Doctoral Thesis

Extended Personal Media Networks (XPMN)

by Dirk Meyer

Submitted to fulfill the requirements for a degree of

"Doktor der Ingenieurwissenschaften"

– Dr.-Ing. –

at Fachbereich 3 (Mathematik und Informatik)

Universität Bremen

November 16, 2009

Reviewers:	Prof. Dr.-Ing. Carsten Bormann (Universität Bremen)
	Prof. Dr. Rainer Malaka (Universität Bremen)
Colloquium (Thesis Defense):	March 4, 2010

Abstract

The recent years have brought many changes to the way consumers interact with media assets. Today, consumers store their media digitally and access them using multiple devices; the role of the mobile phone as a universal control and playback device is still growing. Yet, there is no generic architecture for the various devices at home and on the Internet to interact with each other.

In this thesis, we enhance the *Extensible Messaging and Presence Protocol* (XMPP) to be the core of an *Extended Private Media Network* (XPMN). Essentially, we create an end-to-end security layer for XMPP providing confidentiality, data integrity, and peer entity authentication between clients, keeping usability constrains for both the end-user and the developer in mind. In addition, this thesis covers device management, non-relayed TCP connections between peers in a challenging topology, and a generic device architecture—turning XMPP into a powerful peer-to-peer network for inter-device communication.

To prove the applicability of the specifications, an exemplary implementation has been built within the scope of this thesis. Moreover, the core specifications were developed in an open standardization process and are either published as XMPP extension or Internet Draft.

Zusammenfassung

Die letzten Jahre brachten viele Veränderungen, wie Konsumenten mit ihren Medieninhalten interagieren. Sie speichern ihre Medien heutzutage digital und greifen mit unterschiedlichen Geräten darauf zu, und dabei wird die Rolle des Mobiltelefons als universelle Fernbedienung und Abspielgerät immer wichtiger. Allerdings gibt es keine generische Architektur, die es erlaubt, dass die unterschiedlichen Geräte zu Hause und im Internet interagieren können.

In dieser Arbeit wird das *Extensible Messaging and Presence Protocol* (XMPP) erweitert, damit es als Kern für ein erweitertes privates Medien-Netz (Extended Private Media Network, XPMN) eingesetzt werden kann. Dies beinhaltet hauptsächlich die Entwicklung einer Ende-zu-Ende-Sicherheitsschicht für XMPP, um Geheimhaltung, Datenintegrität und Authentisierung zu gewährleisten. Dabei gilt der Benutzbarkeit für den Endbenutzer, aber auch der Entwickler besonderer Aufmerksamkeit. Zusätzlich behandelt diese Arbeit Geräteverwaltung, TCP-Verbindungen zwischen Clients in komplexen Netztopologien und eine generische Device-Architektur – also eine Weiterentwicklung von XMPP in ein leistungsstarkes Peer-to-Peer-Netz für Geräteinteraktion.

Um die Anwendbarkeit der Spezifikationen nachzuweisen, entstand im Rahmen dieser Arbeit eine prototypische Implementierung. Darüber hinaus wurden die Kernspezifikationen in einem offenen Standardisierungsprozess entwickelt und als XMPP-Erweiterung oder Internet Draft veröffentlicht.

Acknowledgments

The research described in this thesis has been performed at the Universität Bremen in the Arbeitsgruppe Rechnernetze headed by Prof. Dr.-Ing. Ute Bormann. Looking back the last one and a half years, many members of this research group have been involved in discussions or the architectural design of the extended personal media network. In particular I would like to thank Prof. Dr.-Ing. Carsten Bormann for supervising my work and pointing me in the right direction when needed. Furthermore, Kevin Loos for mentioning XMPP as a possible core protocol and the teamwork on various TZI projects. In 2007 and 2008, we worked together on the Scalenet and the M4 projects dealing with mobile media and covering a small subset of the possible use cases the architecture developed within the scope of this thesis can cover. These two projects started my research activities in this area.

Yet, discussions with Volker Wittpahl and Nicolas Pauluhn at the CeBIT 2006 initially sparked my interest on the concept of extended personal media networks. Even though it took two more years until this thesis was started, these discussions provided me with new (sometimes overdrawn) ideas on the way we could deal with our media assets.

During my time at the Arbeitsgruppe Rechnernetze and at the graduate school "Advances in Digital Media" I had the opportunity to work together with many people with various interests and knowledge. They provided me with insight to the fields of networking, security and usability. I would like to thank all of these people for their support, notably Sebastian Feige for a different view on the topic and some very informative usability conversations.

I sincerely appreciate the support I have received from members of the XSF as well as the IETF; first and foremost I would like to thank Peter Saint-Andre, the co-author of the Internet Drafts and some XMPP extensions published within the scope of this thesis. Furthermore, Dave Cridland, Justin Karneges, Alexey Melnikov, and Eric Rescorla for the fruitful discussions on the mailing lists and during meetings in Brussels and Stockholm on XMPP end-to-end security and TLS. In addition, I would like to thank Klaus Hartke for implementing some of the specifications developed within the scope of this thesis in his XMPP library and giving me some helpful feedback.

This thesis would not have been possible without support from the Klaus Tschira Foundation and the graduate school "Advances in Digital Media"—and Prof. Dr. Rainer Malaka for giving me the opportunity to be part of the graduate school. The one-year scholarship gave me the chance to focus on my research.

Finally, I would like to thank my friends and family who supported me over the last years; they may not even be aware of their impact on this thesis. Most notably Niels Pollem, Olaf Bergmann, Stefan Prelle, Benjamin Walther-Franks, Andreas Büsching, and Alina Stürck for reading parts of this thesis and giving me some final advice, and last but not least Birgit Ruopp for always listening to my (sometimes unjustified) complaints and for moral support.

Contents

List of Figures

List of Examples

Nomenclature

ACL	Access Control List
CA	Certification Authority
DNS	Domain Name System
DNS-SD	DNS Service Discovery
DVB	Digital Video Broadcasting
IBB	In-Band Bytestreams; XEP-0047
ICE	Interactive Connectivity Establishment
IM	Instant Messaging
IQ	XMPP stanza Info/Query
ISP	Internet Service Provider
JID	Jabber ID
Jingle	Multimedia Session Setup based on XMPP; XEP-0166
LAN	Local Area Network
mDNS	Multicast DNS
NAS	Network-attached storage
NAT	Network Addresses Translation
NAT-PMP	NAT Port Mapping Protocol
NFC	Near Field Communication
OOB	Out-of-Band Bytestreams; XEP-0265
OpenPGP	Strong public-key and symmetric cryptography; mainly used for email
P2P	Peer to Peer
PEP	Personal Eventing Protocol; XEP-0166
PKI	Public Key Infrastructure
Pubsub	Publish-Subscribe; XEP-0060
RFC	Request for Comments
RTP	Real-time Transport Protocol
SASL	Simple Authentication and Security Layer
SIP	Session Initiation Protocol
SOCKS	Tunnel protocol for TCP and UDP
SRP	Secure Remote Password Protocol
STUN	Session Traversal Utilities for NAT
Teredo	Tunneling IPv6 over UDP
TLS	Transport Layer Security

TURN	Traversal Using Relay NAT
UPnP	Universal Plug and Play
UPnP IGD	UPnP Internet Gateway Device for NAT penetration
VCR	Video(cassette) recorder
VoIP	Voice over IP
XEP	XMPP Extension Protocol
XEP-0189	XEP for Public Key Distribution
XMPP	Extensible Messaging and Presence Protocol
XPMN	Extended Personal Media Network
XSF	XMPP Standards Foundation
XTLS	Jingle Transport Layer Security for XMPP

Chapter 1

Introduction

"The best way to predict the future is to invent it." — Alan Kay
"...or at least posit a vision for others to build." — Gordon Bell and Jim Gemmell

The recent years have brought many changes to the way consumers interact with media assets: broadcast became digital, analog video recorders were replaced by DVB receivers with hard disks, and DVDs replaced the video cassette for selling or renting out movies. Online music stores having music files for sale have started to replace the audio CD while online music streams and podcasts replaced the local radio stations for some users. Digital photo cameras replaced analog ones; some images are now only accessible from the PC. Mobile phones have integrated media players and cameras and they can automatically upload photos to online image galleries and download new podcast episodes. These changes influenced the way consumers interact with their media assets; they want to take their media assets along—want to access them from everywhere.

And the development is still in progress: companies started *Video on Demand* (VoD) services and some *Internet Service Providers* (ISP) already offer television channels over IP broadcast (IPTV). Other companies provide recording services for TV channels (Network-based Personal Video Recorder, NPVR). This development shifts some media-related services from devices inside the user's home onto the Internet.

What is missing is a generic and user-friendly way to access all the personal media assets everywhere at any time. Today, most devices cannot interact with each other. The recordings on the DVB receiver can only be accessed by playing the recording on the device itself and only a few devices provide access to the recordings over USB as mass-storage device; admittedly, one reason for this are restrictions due to *Digital Rights Management* (DRM). Today, set-top boxes for digital TV broadcast, video on demand solutions, streaming clients, DVD or Blu-ray players, and personal video recorders are all connected directly to one TV set. All these boxes must be physically near the TV set, can only be controlled in a line of sight with a remote control, and cannot be accessed by a second TV set without rewiring.

Some devices implement *Universal Plug and Play* (UPnP) to interact with another. UPnP is a local media network: a logical network of devices in the user's home network dealing with media assets. UPnP has a generic core and on top of it profiles such as playback control and a directory service to access media files. Unfortunately, the UPnP core is only designed to be used inside the local network and has substantial security issues [Haq07]; *UPnP Device Security* is optional and most devices do not implement it.

Furthermore, there are several use cases where a link-local media network as defined by UPnP is not sufficient. Today, most users have mobile phones and may want to use these phones to control their media, for instance, schedule a recording or access a file from outside the home network. Even if the phone is physically at home when performing these tasks it may not be part of the home network; the mobile phone may be connected over GSM/UMTS and therefore part of the operator's access network.

Besides the user's own devices, there are many services provided by third parties on the Internet that could be integrated into a media network. Examples for such services are IPTV solutions from ISPs, online TV recorders, online storage solutions such as box.net, and popular media-related websites such as YouTube and Flickr. Consequently, it also should not matter whether or not the user owns a media asset: a friend's photo feed, popular YouTube videos, and podcasts should be accessible from devices in the media network, too.

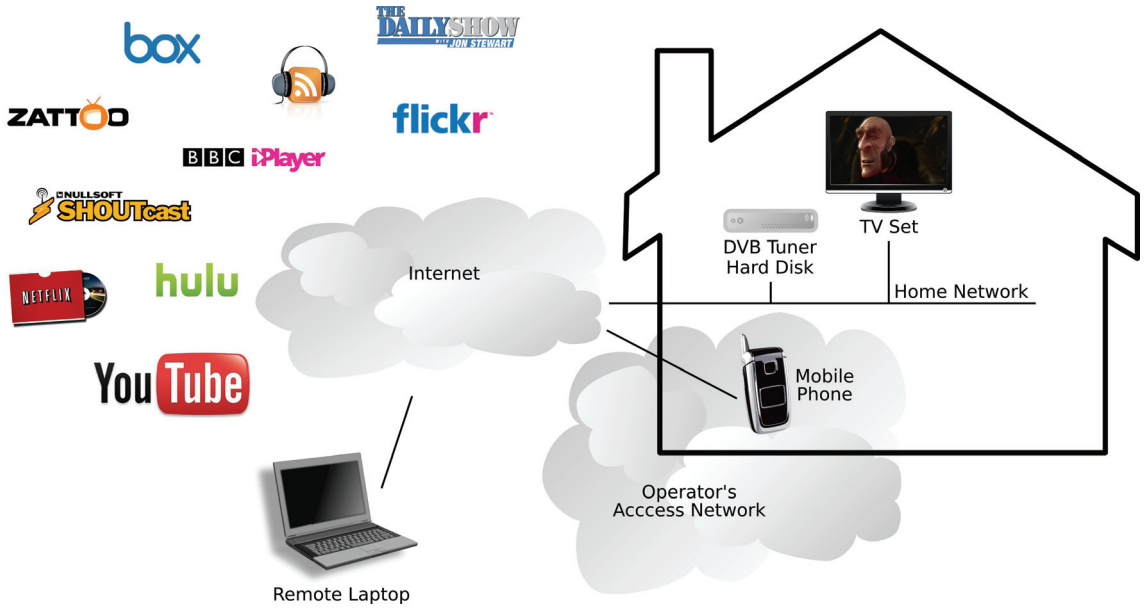

Figure 1.1: Possible XPMN Devices[1]

All devices in the home network, the user's mobile devices, and external services should work together in an **Extended Personal Media Network** (XPMN). Figure 1.1 depicts such an extended personal media network with a remotely connected laptop and multiple Internet-based services.

[1]Most logos used in the image are trademarks of the specific web services or companies.

Unfortunately, most of the external services shown in figure 1.1 are only accessible through a web browser. Even though they provide a service similar as local devices, for instance, an NPVR is used to record a TV show, they do not blend in into the local media network. Furthermore, a user may want to create a virtual TV channel with a mixture of broadcast TV, local video storage, and on-demand content from the Internet [Cho03]. This requires one standardized interface to access media content—we need a convergence layer to access all these media assets. This leads to the main question for this thesis: **How can users interact with all their media through different interface devices in a unified, secure, and user-friendly way?**

Prior to this thesis, the author worked on several media-related projects at the TZI. A common element was the interaction of a mobile phone as the controlling device for the media playback; either the mobile phone was used to control a remote TV set or to resume video playback on the phone started on the TV set at home. These projects focused on a working prototype to show that it is possible—with a simple user interface not designed under today's usability aspects; and security was completely ignored, mainly because security is not something that is visible. Security was considered in the initial design, but not implemented.

Additionally, the author is the maintainer of Freevo, an open source media center software solution for Linux.[2] Freevo is split into a user interface for the TV set, external components for recording, and a web server. The user can schedule a recording with the TV interface as well as the web front-end. Using the mobile phone to control Freevo and its recording component is a personal goal for the next major Freevo release.

The media-related research projects as well as the personally motivated activities on Freevo revealed the following four issues an extended personal media network has to address:

1. **Service Discovery:** XPMN devices need to find and identify each other and must be able to query each others' capabilities, for instance, playback device or media server. The device discovery mechanism must be robust against dynamic IP address changes and consider private subnets as existent in today's home network.

2. **Security:** The privacy of the user must be respected. A rogue node in the media network or an external attacker should not be able to access personal media files it is not supposed to access. Only a subset of all devices should be allowed to access media content and these should only have access to a defined subset of the content.

3. **Usability:** It must be easy for a non-technical user to set up the media network and specify security parameters. The security layer must balance between close-to-none user interaction and a strong level of security.

4. **Extensibility:** Similar to UPnP, a protocol core should be specified with multiple XPMN profiles on top of it. This design makes it easy to extend the media network to future use cases.

[2] Freevo is a modular media center application suite based on the *Kaa Media Repository*, a set of python modules to create media-related applications; `http://www.freevo.org/` and `http://www.freevo.org/kaa/`

Extensibility is easy to achieve by splitting the XPMN architecture into a core and profiles on top of this core. Hence the focus of this research is on the combination of **networking**, **security** and **usability**. Most current research papers ignore at least one of these three topics. Some ignore networking requirements such as dynamic IP addresses and private networks at home, for example, by making the usage of IPv6 and no firewall a requirement [SV03]. Others make the interface to the device different if the user is outside the home network, for instance, a web interface to access UPnP devices located at home [JSL+04, YMH06]. And many ignore security concerns completely.

Creating a new protocol from scratch is always possible, but finding an existing exchange protocol that could be improved is a more appealing option [Ros01a]. Enhancing an existing protocol makes it possible reuse existing implementations, infrastructure, and research on scalability and security. Analyzing protocols not suitable for an extended personal media network and the research that led to these protocols provide an informative basis on additional requirements and possible mistakes.

The XPMN core architecture will be based on the *Extensible Messaging and Presence Protocol* (XMPP) [SA04b]; the reasons for this choice are discussed in section 4.4. With XMPP as the base for the XPMN core most requirements of the networking layer are fulfilled. An important requirement not fulfilled by XMPP is end-to-end security: an entity can neither verify the sender of a message nor that a message was not altered by the XMPP servers involved in the communication.

The central questions for the security layer are:

1. Which devices or services belong to the same media network? To answer this question some sort of device identity and XPMN key management is needed. Beyond that, it should be possible to add and remove devices at any time.

2. Who except the owner should have access to the devices and the media assets? What devices belong to these people? Besides having a list of friends it must be possible to query a friend's personal media network for its list of devices.

3. What are devices allowed to do? A simple authentication layer is not sufficient. Detailed access control is required to restrict access for some devices or users. For instance, when a device is shared between family members, access to a file server or to a TV recorder may be limited for the children in the household.

The XMPP-based approach creates a simple peer-to-peer network of user devices [HYAK+04]. Thus, a centralized authorization node may not be feasible; every device might be offline at any time. The security layer must address the special environment of a peer-to-peer system [WPL08]. There is a lot of research on *Public Key Infrastructures* (PKI) for peer-to-peer systems; some require certificates from a verifiable *Certification Authority* (CA) [BEM04], while some are independent of such an organization [Wöl05]. A possible solution could also be a personal CA for every user to control the private network [LLH+07]. Our research is focused on the specific environment of XMPP and the XPMN scenario.

Still, any security protocol is useless if users do not understand what they are doing and why they are doing it [WT99]. The user interface to the XPMN security core should reduce the interaction to a minimum [Gut03] and usability aspects must be kept in mind when designing it. The aforementioned optional UPnP security layer is not widely deployed, presumably because of usability issues. It is very important at this point not to repeat the mistakes that resulted in commercial products not implementing UPnP Device Security.

Terminology

Various research publications cited in this document as well as standards such as UPnP use a similar, but still slightly different terminology. The following terminology will be used within the scope of this thesis:

- The **home network** is the physical network in the user's home, in most cases one broadcast domain (and connected on layers one and two). The home network is connected to the Internet; today, many home networks are connected with a DSL or cable modem with one public IP address assigned for the whole home network.
- A **media network** is an overlay network. A number of host in the home network are part of the media network. On the other hand, hosts on the Internet may also be part of the **extended media network**; for instance, the user's laptop or mobile phone are always part of the media network independent of their location. Each user has an **extended personal media network** and these personal media networks can interact with each other.
- A XPMN **device** is a logical entity in the media network. It does not have to be a physical device; one physical device may contain multiple XPMN devices. In most cases a device is bound to a specific action such as recording from a DVB card or controlling media playback.
- Some devices provide **services** to other devices in the media network. A service is a specific task the device is capable of fulfilling. In most cases a service is bound to a resource of the physical device. A DVB card in a device may result in the device providing a *TV playback and recording* service.
- A device providing one or more services to others is called a **service provider**. A service provider has an interface to be controlled by other parts of the media network.
- A device controlling services on service providers is called a **controller**. A controller may also be a service provider to others; for instance, a TV set controls a remote file server to browse and access files, but is itself a service provider for a *media playback control* service. A controller may have a user interface for the user to interact with the media network.
- A **profile** is a specification that describes a service. It includes the communication protocol between controller and service provider. A profile is specified on top of the XPMN core.

Structure of this Thesis

This thesis describes the XPMN architecture and the steps leading to it. Chapter 2 starts with an introduction of some use cases to get a better understanding how users might interact with the system and which services they may want to use. Based on these use cases the requirements are derived to get an idea what the architecture must cover.

Based on the list of requirements, chapter 3 describes current media network solutions—even though they are not called that way—and research activities for the home network. The focus of that chapter lies on UPnP and various approaches to add devices and services from outside the home network. Chapter 4 takes a look at protocols outside the home network. The primary use case for these protocols may have nothing in common with a media network, but one of these protocols may provide a solid base for the XPMN architecture. After all, UPnP is based on HTTP which was designed for websites and not devices controlling each other to access media files.

The important finding of the analysis is that the XMPP is a suitable basis: it covers most of the requirements and has successfully been used for inter-device communication [Aur05, RS05]. XMPP as defined in RFC 3920 provides a basic communication layer that can be extended by *XMPP Extension Protocol*s (XEP) [SA08b]. Chapter 5 checks XMPP and its extensions against the requirements from chapter 2 and refines the missing requirements in a way they can be realized. Chapter 6 describes the overall architecture and introduces new XMPP extensions and Internet Drafts based on the requirements; filling the gaps to create the XPMN core architecture.

After defining the architecture, it must be evaluated. Since the core architecture itself does nothing useful on its own, services must be defined on top of it; without services a user cannot control anything. One part of the evaluation is to test one of the use cases; therefore, chapter 7 describes the profiles needed to realize this use case. Thereafter, chapter 8 evaluates the XPMN architecture based on an exemplary implementation, testing the most complex use case, and looking back at the requirements. The thesis ends with a conclusion describing possible next steps for development, deployment, and the standardization process.

Chapter 2

Use Cases and Requirements

We need a list of requirements for an *Extended Personal Media Network* (XPMN) before available protocols and publications in this area can be discussed and analyzed. This list should be independent of existing solutions and only depend on some scenarios describing the use of an extended personal media network. The use cases and requirements only take the user's point of view into account and do not cover restrictions content providers or ISPs may want in such a system. For instance, if a content provider restricts the usage of a video so it can only be played on a specific device, some parts of these use cases cannot be put into practice. Having said that, the music industry already has abandoned *Digital Rights Management* (DRM) for music files. This gives hope that the content industry will adapt to wishes of their customers sooner or later.

This chapter describes five XPMN use cases and derives the requirements for such a media network based on them. These requirements will be the basis for subsequent research and the XPMN core architecture.

2.1 Use Cases

The use cases are based on multiple use cases and scenarios from various papers cited in this document, the TZI media center demonstrator at the CeBIT 2006, multiple discussions with friends and co-workers, and personal expectations regarding such a system. They were combined and reduced to be as simple as possible without removing any important features. The XPMN use cases range from a more or less simple scenario inside the home network to complex personal media networks interconnection. Each use case depends on its predecessors and adds additional complexity.

2.1.1 Resource Management in the Home Network

In the most basic scenario, that can already be found in homes today, most devices of the media network are physically located in the user's home. A TV set is connected to one or more set-top

boxes enhancing its functionality. A typical living room TV set is connected to a DVB receiver, a video recorder, and a DVD player. Pay TV channels or video on demand services add more set-top boxes. While the TV set in the living room is connected to all these boxes, a second one in the bedroom is unable to even access DVB channels or videos without additional devices in the bedroom. Even though the user owns the required set-top boxes, a physical connection to a specific TV set is required to access a service. From the technological point of view it does not matter where a DVB receiver is located, as long as the DVB tuner is somehow accessible from the TV set.

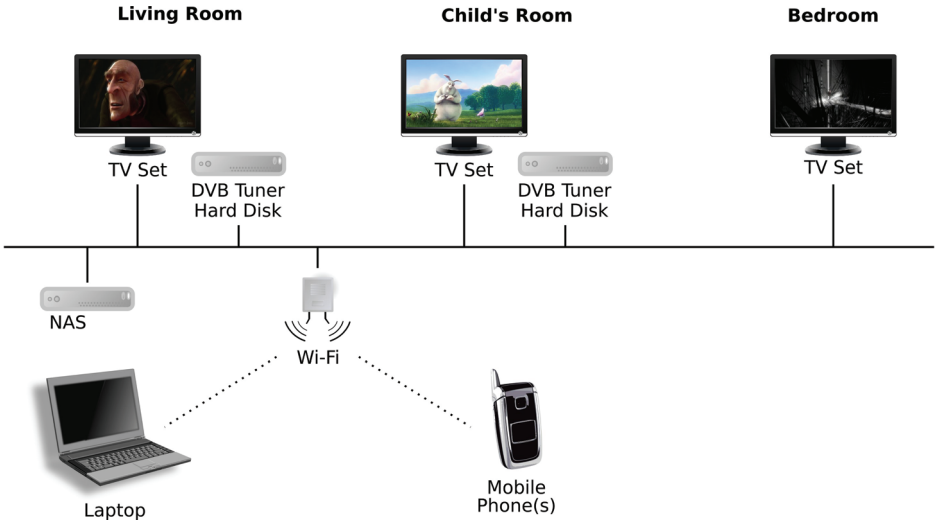

Figure 2.1: Devices in the Home Network

This scenario describes how set-top boxes should interact with each other and share resources. The family in this scenario with one child owns three TV sets: one in the living room, one in the parent's bedroom, and one in the child's room. There are two set-top boxes, one connected to the TV set in the living room and one for the child. Each box is equipped with a DVB tuner and a small hard disk for recordings. Besides the two set-top boxes, the parents own a laptop, one data server (Network Attached Storage, NAS), and each family member owns a mobile phone. Figure 2.1 depicts all these components combined in a media network. Each TV set can access the files on both boxes and the file server.

Similar to the hard disks, the TV tuners in both boxes are shared resources as well. It is possible to watch TV in the bedroom without a TV tuner directly connected to the TV set. Moreover, the media network will automatically choose the best recording device; two simultaneous recordings by the same person would be recorded with one recording on each device. The user does not have to choose which tuner to use, a recording logic will start the scheduled recording on the most suitable device. The recording will be stored somewhere inside the media network: if there is no space left on the hard disk in the box of the selected TV tuner, the recording will be stored on the second box or the file server.

The mobile phones carry personal media assets, too. When a phone has access to the media network, for instance, through a Wi-Fi or Bluetooth link, the phone uploads new photos made

by the integrated camera on the file server and downloads new podcasts. Furthermore, the phones are used to control the media network; an application on a phone can schedule new recordings or start media playback on a specific output device. The laptop is part of the media network similar to the mobile phones. The family members can control their set-top boxes through a browser-based interface.

After a year, the child accidentally breaks the box in its room. The parents buy a new device, a special offer from the local electronics store. The device is manufactured by a different vendor. To reduce the risk of the child breaking the box again, the parents put the new device in their bedroom. The child can still use the functionality of the new box. From the media network's point of view nothing has changed: the number of devices and resources is still the same and the devices from multiple vendors interact like they did before.

Additionally, the new device supports some new kind of interactive service based on DVB-T that was not available before. This new feature can only be used with the user interface on that box; the other XPMN devices do not understand how to use and control it. Still, the new feature does not break compatibility to other services when devices communicate with the new one.

The media network is all about sharing resources between devices. A set-top box should not send the raw video and audio signals via Component, SCART or HDMI to the TV set; devices should communicate over an IP-based network instead to overcome restrictions caused by their spatial arrangement.

2.1.2 Access Restrictions

The previous use case includes three users: the two parents and the child. If the child is not of age, it has restricted access to the media files stored on the two set-top boxes and the file server. Only videos suitable for the child's age are visible when the child is browsing the media collection. This is independent of the TV set the child is using. If the child is in the living room it has the same restrictions as in its own room. On the other hand, the parents have access to all files even when they use the TV set in the child's room.

The child also has restricted access for the TV tuners; the channels it can record are restricted to some channels suitable for children and recordings late at night cannot be scheduled at all. When the parents schedule a recording it may be recorded by the set-top box in the child's room and even stored on the hard disk in that set-top box. The child has no access to the recording on the box in its own room if it is unsuitable for children.

Similar to restricting the child's access to the system, a visiting friend can be granted access for a limited time. Friends can be allowed to control a specific device with their mobile phones or download a limited set of media files. Accessing private data is prohibited and will not work for guests unless the owner of the file allows it.

This use case adds users with various access rights to the system. A device belongs to one or more users and the resources on it may be accessed or controlled by a device owned by another user.

2.1.3 External Services

Today, all devices for a home entertainment system are physically located at the consumer's home. This includes network storage servers and TV recorders, either analog or digital using DVB techniques. With strongly growing high-speed Internet access at home, the Internet Service Providers start pushing multimedia services inside their network. Popular examples are Video on Demand and broadcast TV over IP-based networks (IPTV).

In this scenario, our family is subscribed to the IPTV and Video on Demand service of their ISP. Figure 2.2 shows the ISP's infrastructure and parts of the home network from the previous use cases. The family still owns a DVB-T tuner as fallback since IPTV depends on the DSL connection; if the DSL connection is down, the IPTV service becomes unavailable.

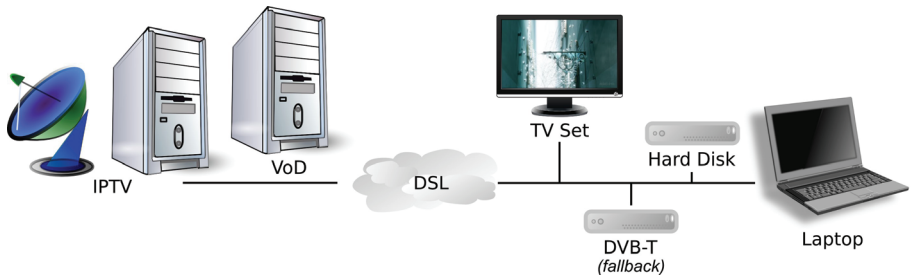

Figure 2.2: ISP Services Integration

There is no difference between the two sources to receive broadcast TV from the user's perspective (except possible variances in the video quality and the number of channels provided). In both cases the user receives digital video and audio streams. There is also no difference between VoD and movies on the home server, except that VoD could cost some extra money. But even that may not be true if the subscription includes access to a limited set of videos per month. An ISP may grant access to some movies for free; the German ISPs Deutsche Telekom and Hansenet have a changing list of videos on their video on demand platform the users can watch without additional fees.[1]

If there is no real difference between IPTV and DVB the controlling device inside a media network should not have to care which type is used. Today, they are handled very differently and the main reason is that each set-top box is only equipped with a TV-out signal and not capable of providing the service in a generic way. If this restriction is removed, an IPTV or VoD set-top box is not required anymore and the ISP provides its services directly to the user's personal media network—to an *extended* personal media network. Recordings can be scheduled on a DVB device or using IPTV. In case IPTV is used, the recording itself can either be stored by the ISP or on a storage device inside the home.

The Internet connection is optional and the media network does not depend on it. When the Internet connection becomes unavailable, recordings are automatically switched to the local DVB-T tuning device if the requested channel is available on a DVB-T frequency.

[1] The "Entertain Comfort Plus" contract from the Deutsche Telekom included a list of ten free movies the user can watch without extra charge in September 2008. The list of movies changes every month.

The ISP is not the only external service provider the family uses. The parents mark some photos as public and they are automatically uploaded to Flickr. Some photos are not even stored on the home network's hard disk and are only available on Flickr; the photos stored externally belong to the personal media network just like photos stored locally. Besides their own photos, the family is subscribed to a friend's photo stream: these photos are automatically part of the family's media network once the friend uploads new photos to Flickr.

Similar to a friend's photo stream, other media-related websites such as YouTube or Internet radio stations are integrated. Each podcast the users are interested in integrates into the personal media network; it does not matter where the content is stored.

Making external services part of the media network raises new security concerns: the ISP would provide services to the media network, but should not be able to control services provided by others. At this point we also cannot ignore the ISP's requirements: the ISP must be able to identify the user for billing purposes. Video on Demand and Pay per View channels require user identification as well.

2.1.4 Remote Control

Just like external services, other parts of the media network can be located outside the local network. A personal laptop is always part of the media network, independent of its location. The extended personal media network is independent of the home network and does not change whether a device is part of the user's home network, is connected through a Wi-Fi hotspot, or is a guest in a friend's home network.

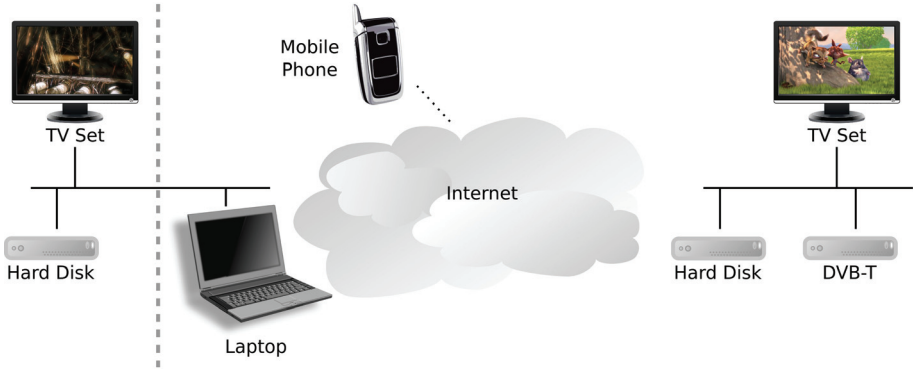

Figure 2.3: Remote Devices

The parents in our use cases own a laptop, and one of them is visiting a friend and attaches the laptop to the friend's home network. The laptop is somehow aware that it is not at home. It may not be able to access services in this local network and prevents its services to be controlled from devices in there as well. It still belongs to the personal media network of its user (see figure 2.3).

The mobile phones are used by the family members to control their media network. At home, the phones are connected through the local Wi-Fi access point; outside, an UMTS connection

is being used to schedule a recording or to access music from the file server at home. The device automatically chooses the best connection method and tries to be reachable from the home network. Furthermore, a mobile phone may be temporary offline, for example, in tunnels or an elevator in the office. Other devices are aware that the phone may be unreachable for a short period of time.

Therefore, a device has to be able to detect if it is at home. It connects to the home network through the Internet if it is at a remote location—it must be able to locate its (home) media network and connect to it. The interface the device provides to the user does not depend on its location and always looks the same. Even physically inside the user's home, a mobile phone may be a remote device from the network's point of view: connected over UMTS, the mobile phone is part of the operator's access network and not inside the home network. This illustrates again that the media network is not defined by IP address ranges; it is a logical network. A device may not even have an IP address. There could be a special Bluetooth profile to include Bluetooth devices directly without creating an IP-layer on top of the Bluetooth link.

2.1.5 XPMN Interconnection

In the previous use cases the media network is described as something logical, independent of the physical network. Since it may include guest devices with limited access, a device may be both a guest in one media network and remotely connected to its home.

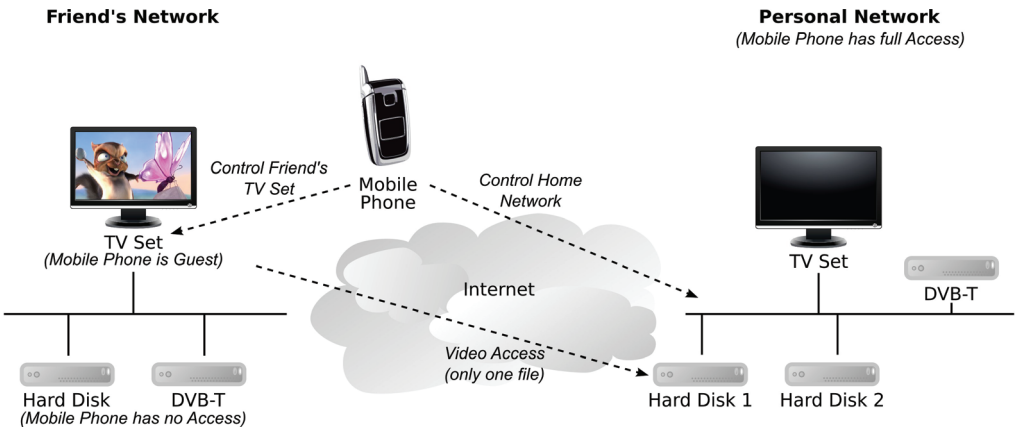

Figure 2.4: Network Interaction

In this last, more complex scenario, our family visits a friend and wants to show the friend a video stored on one of the hard disks at home. Both the family and the friend have personal media networks and a security layer restricts one personal media network access to services on the other. The mobile phone of one of the parents somehow acts as a bridge between the two personal media networks. Figure 2.4 depicts the involved devices and the mobile phone's privileges: it is a guest in the friend's media network while still being part of family's own media network.

To realize the remote playback with a device in two media networks, three steps are necessary to create a trust relationship between the friend's TV set and the file server at home:

1. The friend grants the mobile phone guest access to control the TV set. The mobile phone is now allowed to start video playback on the TV set, but has no access to other parts of the friend's media network.
2. The mobile phone connects to its primary media network, browses for the media file the user wants to show, and adds the friend's TV set to be allowed to access this specific file.
3. The hard disk at home returns an URI for the TV set to access the video file and the mobile phone provides that URI to the friend's TV set.

The mobile phone in both personal media networks creates a trust relationship between the file server in one media network and the TV set in the other. The devices do not know each other, but both recognize the mobile phone as a trusted third party with certain access rights. The hard disk at home already knew the mobile phone because they belong to the same user and the phone has guest access to control the TV set.

Combining external services and guests makes it possible to merge two extended personal media networks and let devices in one access service provided by devices in another. With this sort of interconnection it is also possible to access other media files, for instance, slides for a meeting shown on a presentation computer.

2.2 Requirements

As a next step we reduce the use cases to basic requirements for the extended personal media network. This section lists the main requirements and describes how the requirements depend on each other. Each subsection defines a group of requirements loosely belonging together. Some requirements are not directly based on a use case and are derived from observations on previous ones instead.

2.2.1 Service Provider

The current practice of set-top boxes providing features only for the connected TV set restricts the usage of these resources. Instead, a box should provide its services to the whole media network. A small controlling device is the only component that should be connected to the TV set and future TV sets may have such a controller integrated. Modern TV sets already include memory card readers or can act as UPnP control points (for details on UPnP see section 3.1). Adding a controller for a media network should have a similar complexity as adding UPnP support to allow an easy deployment.

The set-top boxes in the first use case contain two resources: DVB tuners and storage capabilities. These resources should be wrapped into multiple XPMN services the devices provide to

the media network: a recording service, an electronic program guide, and media access. The media access service is provided by both the DVB resource (live TV) and the storage component (recordings). The services are not only accessible for a controller inside the same box; other set-top boxes can use and control these services as well. Hence there are two kinds of devices: *service provider* and *controller*. A physical box is only a device that provides services and may not even have a controller. The DVB service devices and the storage server without a user interface on their own can be located in the attic. Only some services require being placed near the TV set; if the user wants to insert a disc into a DVD or Blu-ray player the device should be easily accessible.

Some services require other services to run. For instance, a DVB device must have access to a storage device to provide a recording service. A service provider may include a controller for other services; a user interface is a controller.

Requirement 1: Access to services is independent of the device they are running on. A device provides services based on its resources to the whole media network.

Requirement 2: Services are controlled by controllers. These controllers are either bound to another service provider or provide a unified user interface. Possible controllers operated by users are set-top boxes, mobile phones, and PCs.

The term set-top box defines a box that provides an external signal or content to a TV set. A set-top box in this terminology is not needed anymore if the TV set is connected to the media network directly. For TV sets without XPMN support, a set-top box with a controller is required to include it. All other boxes do not have to be "set-top" boxes at all. The set-top box is only required for the transitional period of current home environments and is a placeholder for an XPMN-aware TV set.

2.2.2 Interoperability

If access to services is independent from the devices, there must be a standardized API to access them. The new device in the first use case works with the older ones from a different vendor. An open standard is required to increase the chance that a vendor will support that API. A closed solution will only provide service independence for devices from the same vendor. This will not only force the user to buy new equipment from the same vendor, it will also require all vendors to develop a protocol for a media network on their own. This will reduce the acceptance for such a system.

A current trend in the home network are streaming clients: a small set-top box that can be used to play videos, listen to music, and watch images depending on a PC or NAS in the local network to provide the content. The PC provides the content in a format the set-top box can decode and converts media files if necessary. In some cases the whole user interface is provided by a PC. The idea behind this is to use CPU power and storage capacity of a PC on the set-top box. The software used on the PC varies for the different set-top boxes. Some require the *Windows Media*

Center Edition (MCE) and some vendors provide software on their own. Even this simple setup with only two devices causes incompatibilities between vendors. Adding external DVB devices will make the setup more complicated, forcing users to check if devices can interact with each other before buying a new device or application. This must be avoided.

Requirement 3: The media network and the services must be defined in an open standardization process. This includes good documentation and examples to avoid incompatibilities due to misunderstandings. Access to the specification should be granted for all vendors to implement service providers and controllers.

Requirement 4: Each service must have a unique identifier to gain interoperability between vendors. The service name must be the same on all service providers providing the same kind of service.

2.2.3 Extensibility

An XPMN architecture designed today cannot cover all future use cases and some vendors may want to extend some services with additional features. In the first use case, the new set-top box in the bedroom has support for a new interactive service based on DVB-T. This service was not available when the other boxes were produced. It must be possible to provide further functionality to existing services without breaking compatibility with devices not supporting the new extension. To keep vendors from developing a new media network protocol that fits their needs, the XPMN architecture must be flexible enough to be extended with new services.

Therefore, the XPMN architecture protocols are split into two logic layers: a core, providing the basic functionality of the network, and services based on that core.

Requirement 5: The XPMN architecture must have a core protocol and service-specific extensions on top of it. The core is mandatory and fulfills the basic, service-independent requirements.

Requirement 6: It must be possible to define new services without breaking compatibility to the core. A media network may contain a mixture of older and newer devices.

Requirement 7: It must be possible to extend existing services with new functionality without breaking compatibility to clients not implementing this extension.

2.2.4 Accessibility from Outside the Home Network

Presumably most devices are located inside the user's home network, but some service providers and controllers are outside on the Internet. An extended media network must be capable of including these devices. In the third use case, the IPTV and some web-based services are located outside the home network; the fourth use case includes a laptop and a mobile phone

acting as a controller outside the home network. A controller must be able to contact all service providers it is allowed to control.

An ISP typically only provides one dynamic IP address to each subscriber and techniques such as *Network Address Translation* (NAT) [EF94] are used to share this public IP address between all devices in the local network. This practice yields two challenges on the networking layer:

1. It is not possible to address a device inside the home network from the outside. A device from the outside only sees one public IP address for all devices. When IPv6 is widely deployed this problem will be solved; each device can obtain a public IPv6 address.
2. The IP address of the home network changes. ISPs often force a disconnect every twenty-four hours. An external controller must be able to locate the home network independent of its current public IP address. Even with IPv6, it is likely that ISPs will still change the public IP addresses for the home network. Providing a static IP address is an extra service users currently have to pay for. It is unlikely that an ISP sets this extra fee aside when adding IPv6 support.

There are two obvious solutions to locate the home network. The first one is an external service to resolve a unique media network name into the IP address(es) the home network currently owns. A possible protocol for this is the *Domain Name Service* (DNS) and name service providers such as DynDNS[2]. The second solution is to route everything over an external server with a fixed public IP address. The external devices connect to this server instead of connecting to the home network. All devices in the home network connect to the server, too, and the server routes the data to the correct endpoint.

The IP address is no option for addressing a specific device or service if devices or even whole networks change their IP address frequently; the media network must provide unique identifiers for each device. Having unique identifiers independent of the IP addresses and virtual links to route messages over a server or other nodes are two characteristics of an overlay network. An overlay network is a logical network on top of another network—the media network is such a logical network similar to peer-to-peer networks running on top of the Internet. All XPMN devices have a unique identifier independent of IP addresses and valid inside the overlay network.

Requirement 8: The media network is an overlay network and devices should not be identified by their current IP addresses. As shown in the last use case, devices in the same IP subnet do not necessarily belong to the same media network.

Requirement 9: A controller must be able to connect to service providers outside the home network.

Requirement 10: A controller on the Internet must be able to connect to a service provider inside the home network. It has to be aware of the fact that there are dynamic IP addresses and NATs.

[2]DynDNS is a service provided by Dynamic Network Services, Inc. allowing users to have a sub-domain pointing to a computer with a regularly-changing IP address; `http://www.dyndns.com`

Requirement 11: Each service provider and controller must have a unique identifier that does not change. It must be independent of the currently used IP address.

If the data is routed through an external server, it is likely that this routing service is provided by a company and that the user is not the administrator of this server. This means that the user does not have full control over all nodes in the personal media network: the server is outside the control of the user.

Thus the communication between two devices must be end-to-end secure, ensuring confidentiality and data integrity. A malicious routing server may stop the media network from working correctly but should not be able to get control over the devices and access the user's media assets.

Requirement 12: All communication between devices must be end-to-end secure, providing confidentiality and data integrity.

2.2.5 Network Error Tolerant

The Internet connection is a weak spot when communicating with the devices outside the home network as described in the third and fourth use case. All external devices are unavailable if the Internet connection is down. This cannot be avoided, but the XPMN architecture must ensure that the remaining devices in the local network are still operational. In the third use case the media network automatically moves all recordings to the DVB tuner once the IPTV service becomes unavailable. Consumers will not accept a technology where DVB-T and local DVD playback are not possible when the outgoing Internet connection is down.

Requirement 13: All devices in the same subnet must be able to interact with each other without an Internet connection; they should not require external devices to work.

Some services may be unavailable if they need an Internet connection to function. For instance, some bonus features on Blu-ray discs download additional streams from the Internet. While the actual movie playback always works, the bonus features may not.

Over and above, current home networks are often connected with a dynamic IP address and a forced disconnect after twelve or twenty-four hours. Modern DSL routers automatically reconnect without user interaction, but the home network has a new public IP address after the reconnect; open connections to other devices become invalid. A device should detect such network changes and act accordingly.

Requirement 14: Communication between devices must be tolerant against dynamic IP address changes of the local network. TCP connections between two devices may become invalid and must be restarted automatically, retransmitting the data sent during the disconnect.

The network error tolerance is even more important for mobile devices such as the phone in the fourth use case. If the user is on the move, it may happen that the network access is disrupted

and it can take several seconds until it comes back. Current 3G networks do not always work in trains, tunnels and elevators. This may be handled by a special transport protocol taking care of such network errors and reestablishing the connections later. In consequence, it could take some time until a mobile device can receive something from a peer or execute a command. The peer must be aware of that.

Requirement 15: Communication with external devices should be tolerant against small gaps in network reception. All devices must be aware that the other may be incapable of receiving something or answering for a couple of minutes.

2.2.6 Resource Discovery

Service providers and controllers are on multiple devices in the media network and a controller must be able to locate all available services. The devices are scattered in the home network and even on the Internet and a controller must be able to locate all devices it wants to communicate with (*device discovery*). When a device is discovered the controller must be able to query the remote device about its services (*service discovery*).

On start-up of a device, it may take a while due to network constraints until all other devices queried its services and it queried all the others. It should be possible to cache the service discovery results to avoid unnecessary service queries when a known device becomes active again. There are two obvious caching strategies: each client can do its own caching or a client could rely on a centralized service location server where all service providers register their services. If a device provides services based on the availability of others, for instance, a recording service depends on a TV tuner and a storage solution, the list of services a device provides may change during runtime. The caching mechanism must be aware of this.

Requirement 16: It must be possible for a controller to locate all the services it is allowed to use. This includes a *device discovery* and a *service discovery* on these devices.

Requirement 17: It should be possible to skip the service discovery by using a discovery server or by caching the list of services a device provides. If caching is used, a controller must be able to detect whether the remote device still provides the same services as before.

2.2.7 Device Management

One important question for the device discovery as required in the previous section has not been addressed yet: a device needs to find out which other devices belong to the same personal media network. The device discovery mechanism may discover many devices the user is not allowed to use or does not want to use. A security layer must prevent attackers from adding devices to the media network which pretend to belong to the user; a device authentication mechanism is needed. Asymmetric encryption with a device specific certificate is commonly used for such a task. This requires the XPMN core to include some sort of key or certificate management for peer entity authentication.

Requirement 18: A device must be able to detect whether or not another device belongs to the same media network. The discovery mechanism should provide this information. The device must authenticate itself to other devices and request authentication from its peers.

The list of user devices may change over time. Users may add new devices or remove devices from the media network. In the first use case, the broken set-top box is removed and replaced by the new one. Other reasons are the user selling a device or it may get stolen. The latter shows a possible problem with device removal: if an attacker steals a controller, this controller could be used to remove all other controllers from the media network, locking out the rightful user. The user must have a recovery mechanism to prevent such a scenario.

Requirement 19: If a device is added to the media network, all other devices must be somehow notified that this device now belongs to the media network.

Requirement 20: It must be possible for a user to remove a device from the media network. The removal must work even without the device itself to cover broken or stolen devices. All other devices must be made aware that they are not allowed to communicate with that particular device anymore. If a device is not available at the time of the removal of another, it must be notified once it gets online again.

Requirement 21: The network should prevent an attacker from locking out the rightful user.

2.2.8 Interaction with Friends

In the last use case, the user's mobile phone interacts with two personal media networks: it controls the user's home devices and the friend's TV set. The friend may not want to grant the mobile phone complete access to all service providers, only to the service controlling the TV set. Therefore, the mobile phone is only a guest and not part of the friend's media network. The child access control in the second scenario can be treated similarly. The child might not own any service providers itself and only has some sort of guest status on all services in the household—even on the box located in its own room.

With friends and children as guests in the media network, a simple authentication layer is not sufficient. Detailed access control is required to restrict access for the external users. Multiple personal media networks must interact with each other to authenticate devices.

Network identifiers are needed when we have more than one personal media network. Each device has its device identifier as unique identifier inside its media network. One possible globally unique identifier is the combination of the media network identifier and the unique identifier inside that particular network.

Requirement 22: Each media network has a unique name.

Requirement 23: A device has a globally unique identifier besides the unique identifier inside its media network.

Requirement 24: The XPMN architecture must provide a mechanism to define friends/guests that are allowed to access or control parts of the personal media network.

Requirement 25: The service discovery requirements from section 2.2.6 are not limited to the user's own devices. It must be possible to locate and query the friends' devices, too.

Requirement 26: A device needs a mechanism to ensure that a remote device is part of the media network it claims to be. The peer entity authentication must work across multiple personal media networks.

Requirement 27: It must be possible to define some sort of access rights for each guest or group of guests. All devices of the guest may share the same access rights. These access rights may expire, for instance, the family is only allowed to control the friend's TV set for one evening.

The details for the access control depend on a specific service and therefore cannot be part of the XPMN core architecture. For example, a file server may support access control for files and directories, the TV tuning device in the second use case allows the child only access to specific channels, and the TV playback service in the last use case grants the guest full access for a limited time. Thus, the service specifications should include access control details.

2.2.9 Ease of Use

The above mentioned requirements must be fulfilled in a user-friendly way; the media network must be easy to use. If a solution is very complex, this complexity must be hidden from the user. The following requirements are not based on the use cases and are some sort of meta-requirements that must be kept in mind when fulfilling the others.

For the networking layer, an external server may be required to locate other devices outside the home network and for the home network to locate the devices outside. This server has to be configured somehow. The same is true for the security requirements: managing the security should be easy and comprehensible. The best security protocol is useless if users do not understand what they are doing and why they are doing it [WT99]. The user interface to the authorization infrastructure of the media network should reduce the needed interaction to a minimum [Gut03].

Creating an architecture that is usable and secure at the same time is very important: it requires user interaction to fulfill a security-related task and users may have difficulties understanding why the security is even necessary. ISO 9241 Part 11 defines usability as "the extent to which a product can be used by specified users to achieve specified goals with *effectiveness*, *efficiency* and *satisfaction* in a specified context of use." [Tec94, accentuation added] Furthermore, there are several other aspects one should consider. For the usability of security *error tolerance* and the *inducing of awareness* also seem to be of importance. The user has to understand why security is necessary and needs some guidance in this area. Often security is considered as annoyance; many users are "trained" to click on the OK button on pop-up boxes without reading the actual message.

Requirement 28—effectiveness: The user must be able to successfully complete a task and the result should be as expected.

Requirement 29—efficiency: An interface must be simple and it should not take too much time to complete a task. This also includes the learning phase. A user interface should be "suitable for learning" (ISO 9241 Part 10). It should be possible for the user to understand a task without reading an additional manual.

Requirement 30—satisfaction: The user should like the system and the level of options that can be controlled. From the security point of view there may be many options a developer wants to provide to the user; but too many options often overstrain the user. This must be avoided by reducing the available options.

The iPhone serves as a good example here: it is less powerful than other mobile phones but it provides the available functionality in a simple user interface. Other mobile phones with more features often have confusing menus to fit all possible options.

Requirement 31—error tolerance: The system must be error tolerant to avoid that the user accidentally deletes personal files or shares private data or resources.

Requirement 32—inducing awareness: The interface must make the user aware of security issues and why some settings are necessary. After some time, the user will know about possible risks and intuitively handle some tasks with more care.

2.3 Summary

The devised requirements only cover the requirements of the core protocol. The services themselves have additional requirements; a requirement for a TV tuner may be that the device supports live-pause or time-shifting. We need a basic core protocol first before we can go into details of possible XPMN services. When the core protocol fulfills all requirements mentioned in this chapter, we can use the extensibility requirement for section 2.2.3 to add the actual services.

The focus of the following research is on the combination of networking, security and usability. All three topics must be analyzed in one step. Like adding security to an existing networking layer, it is very difficult to add usability at the end when the other layers are done. It is not enough to define a user interface at the end; usability must be kept in mind when designing the other parts. The XPMN architecture cannot be perfect in all three areas: we get the best usability by ignoring security completely, and we require additional user input by adding security. This is not as user-friendly as a system that works without it. We must find a good compromise for concurrent requirements when designing the XPMN core.

Chapter 3

Current Home Networks

With the XPMN use cases and the requirements defined we can now check the situation in today's home networks. Creating a new protocol from scratch is always possible, but finding an existing exchange protocol that could be improved is a more appealing option [Ros01a]. Enhancing an existing protocol makes it possible to reuse existing implementations, infrastructure, and research about scalability, security and usability. Even analyzing protocols not suitable for an extended private media network and the research that led to these protocols can provide an informative basis for additional requirements on the protocol level and mistakes that have been made.

When thinking about the first use case, *Universal Plug and Play* (UPnP) is the first protocol that comes to mind. In fact, most elements of the first use case are based on scenarios in papers about UPnP. This chapter describes UPnP as well as *Zeroconf Networking*, *D-Bus*, and *Mbus* because of various reasons described in the corresponding sections.

3.1 State-of-the-Art: UPnP

Universal Plug and Play (UPnP) allows components to discover and use services in the LAN. Its primary goal is to enable the use of devices and services without complicated setup or configuration. This includes getting an IP address even without a DHCP server and accessing the functionality of a component from another. Today, many routers, network storage devices, and set-top boxes are UPnP-compliant. This section gives an overview about UPnP and lists some UPnP-compliant devices and software solutions. Thereafter, we explore the possibility to enhance UPnP to fulfill the XPMN requirements described in the previous chapter.

3.1.1 Overview

UPnP is not one protocol, it is a set of protocols developed and published by the UPnP Forum which had over 800 members at the end of 2008. One scenario defined by the UPnP forum is

"Digital Entertainment". While UPnP is a generic framework, that scenario is the only one with noteworthy deployment.[1]

The *UPnP Device Architecture* specification [Sco02] is the core of the protocol suite and was initially developed by Microsoft in 1999. It is based on several Internet Drafts written by Microsoft employees and was already integrated into the Windows operating system at that time. It defines a general interoperability mechanism to locate devices, control them, and receive notifications on status changes. In addition it specifies Auto-IP, a protocol to obtain an IP address when no DHCP server is present—the plug and play philosophy states that there should be no need to configure a device before connecting it to the network. Auto-IP is based on an early draft of RFC 3927 (IPv4 Link-Local Addressing) [CAG05] and is fully compatible with it. A component requests an IP address via DHCP and if it gets no answer, it chooses a random IP address in the 169.254/16 prefix that is valid for link-local communication. Then it sends an ARP request to determine whether another component already uses this address. When no answer is received, the component uses this address, otherwise it tries another one. Besides that, Auto-IP periodically checks for a DHCP server.

On top of the core are several profiles for specific services. Each service is standardized by the UPnP Forum in a set of *control* and *device templates*. This design makes UPnP extensible for future services. The list of available templates includes profiles to control a printer or scanner, configure a router, and control media center set-top boxes. Besides home entertainment, it is possible to use UPnP for home automation such as controlling the lights in a room by turning them on and off or dimming them.

UPnP distinguishes between a *device* providing services to the network and *control points* to control these services. A control point can use services from various devices to fulfill a specific task, such as browsing the media server for a video file and providing an URI of that video to a device capable of video playback. This terminology is different from the one used in other sections of this document:

- A **control point** is a controller for the UPnP network. Most control points provide a user interface for the end user. A control point is similar to the functionality of an XPMN controller defined in this document.
- A **device** in the UPnP terminology is similar to an XPMN service provider. The terminology used in this document uses *device* for a service provider and controller; a UPnP device always provides services while a control point is never called *device*.
- A **template** is a specification how a device and a control point interact. Each service is specified by a service template. A template is implemented as device template (server) or control point template (client).

This section uses the UPnP terminology unless stated otherwise.

One disadvantage of UPnP is that it contains a lot of optional features with multiple ways to achieve the same goal. This has resulted in many incompatibilities between components from

[1] Additional information including the UPnP specifications can be found at `http://www.upnp.org/`.

multiple vendors. To eliminate this problem the *Digital Living Network Alliance* (DLNA) has defined standards based on UPnP with fewer optional values. Today, most UPnP-compliant devices are DLNA-certified and can interact with each other.

Actions and State Variables

The interface visible to a developer to control a UPnP device is similar to a library. UPnP provides a request/response mechanism similar to procedure calls and events to get notified about internal changes of the device without the need to poll constantly. Figure 3.1 provides an overview of the protocols used for the service discovery, request/response (actions), and the event mechanism.

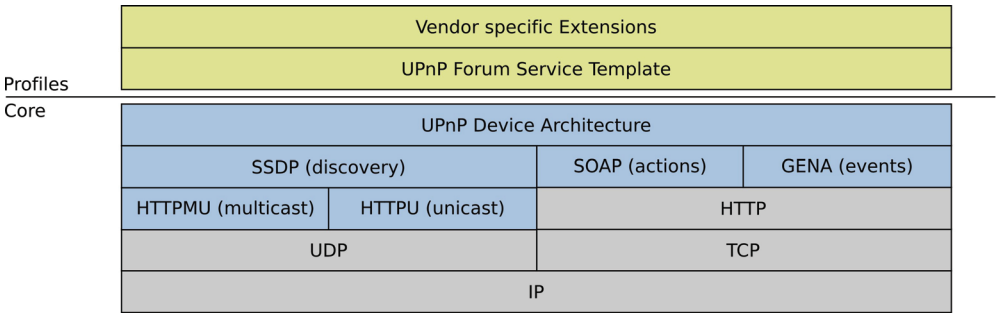

Figure 3.1: UPnP Protocol Stack

A service template defines a list of required and optional *actions* and *state variables* a device provides. An action is sent to the device over HTTP using the *Simple Object Access Protocol* (SOAP) [GHM+07]. SOAP was developed for web services and is based on XML messages for the request and the response. A control point sends a SOAP message to trigger a command on a device and receives the answer in another SOAP message.

To get notified on internal state changes such as the playback status of a media player, a device provides a list of state variables a control point can register to. This is covered by the *General Event Notification Architecture* (GENA) [CA98] which provides a simple HTTP subscribe/notify event architecture. GENA reuses some headers from HTTP and defines the new HTTP methods `SUBSCRIBE`, `UNSUBSCRIBE` and `NOTIFY`. A control point sends a `SUBSCRIBE` request to the device with a callback URI to be used for the event notification. When the internal state of the variable changes, the device sends a `NOTIFY` to the URI provided by the control point. This requires that the device can connect to the control point to deliver events.

The usage of SOAP in web services is often being criticized: “SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment” (SOAP specification) but is in fact very complex and uses a lot of bandwidth [New04]. The same applies to UPnP and it was suggested to remove SOAP from UPnP and replace it with a REST-based interface [New05]. The messages exchanged would be smaller, but it would break compatibility with existing UPnP devices.

Service Discovery

UPnP uses the *Simple Service Discovery Protocol* (SSDP) [GCA+99] to discover devices and services on them; this fulfills requirement 16 for the XPMN architecture. A device sends periodic advertisements in an interval with a minimum of 30 minutes using IP multicast. The maximum value is not defined and could be in the order of days. The advertisement contains an expire time and the device has to send the advertisement again before it expires. Besides the periodic announcement from the device, a control point may send a query using IP multicast to get the list of available services without waiting for all devices to send their next service announcement (which may take several days).

A control point can assume that the services are available as long as the service advertisement has not expired. A device may cancel its announcement before it becomes unavailable. When the user pulls the plug or a device crashes due to hardware or software failure it cannot send such an announcement. A control point does not know that the device is offline until it tries to communicate with it—with expire times in the order of days control points may have an incorrect device list for a long time.

As service announcements are sent over IP multicast and SSDP uses GENA which is built on top of HTTP, some HTTP request and response messages must be exchanged over UDP. This protocol is called *HTTP Multicast UDP* (HTTPMU) [Gol99]. The service description is defined by an XML schema which can be quite complex, but the HTTP request and the response each have to fit into a UDP packet; preferably without fragmentation. To work around this, HTTPMU is only used to discover a remote device and to get an URI to call discovery actions using unicast TCP-based HTTP.

After the control point has discovered the device, it requests the device information using the URI provided in the discovery process. The device descriptions include basic vendor specific information about the physical device, information about logical devices, and a list of services the device provides. This list of services again contains URIs to request the detailed service descriptions. A service description is an XML document that corresponds to a specific *UPnP Service Template*. This service template is defined by a UPnP profile and outside the scope of the UPnP core specification.

The UPnP Audio Video Specifications and Notable Implementations

The *UPnP Audio Video Specifications* (UPnP AV) [RK02] define a set of service templates targeting consumer electronic devices such as TV sets, video recorders, and DVD players. The two main parts are the *AVServer* [PL+06c] and the *MediaRenderer* [PL+06b]. A control point can use these two services to play media content from an AVServer on a MediaRenderer in the local network. The following outlines the AVServer and the MediaRenderer; a more detailed description of the various parts of the AV specifications can be found in the *Intel Technology Journal* Volume 06, Issue 04 [RR02].

The *AVServer Template* defines a service for devices providing any kind of media content to the UPnP network. Many currently available NAS devices for home network usage support this device profile. A control point can browse the media on this device using UPnP and present the result to the user with its own user interface. The media files can be accessed using HTTP, the required URI is provided in the query result. To enhance this basic operation the *AVTransport Template* [PL+06a] specifies playlists, how to provide additional metadata about the media, media streaming using RTP, and media conversion. The conversion is needed if the player cannot handle the format of the media (codec, container). In most NAS devices the conversion feature is not implemented because it either requires considerable CPU power or dedicated hardware support for a wide range of video and audio codecs.

Microsoft Windows supports media sharing using UPnP since *Windows Media Player 10* for Microsoft Windows XP. A Windows PC can act as a UPnP AVServer service provider as well as a control point. Different Windows-based PCs in a network can share their media data over UPnP. A widely deployed but not necessarily well known AVServer device implementation is Twonky Vision[2]. It is pre-installed on some external storage devices and is available for Microsoft Windows XP and Vista, Mac OS X v10.0 or higher, and Linux. Media center software solutions such as the Microsoft Windows Media Center can act as UPnP control points and the Linux application djmount[3] allows mounting the contents of an UPnP AVServer as a Linux file system. While djmount is an interesting solution to provide UPnP support for all applications, many UPnP AVServers also support exporting the data over a standard network file system such as NFS or SMB/CIFS which is outside the scope of UPnP.

The *ScheduledRecording Template* [PL+06d] is another service an AVServer may provide. It offers control over a remote video source such as a TV tuner; including channel listing and recording management. The *GetRecordScheduleConflicts* action is used to show conflicts due to overlapping recordings on different channels. Unfortunately, dedicated DVB tuner set-top boxes available today do not support this template. ZoltarTV by o2media[4] is a powerful set-top box with DVB and UPnP support; however, the specifications indicate that the UPnP support is limited to being an AVServer control point. There are only few software-based media servers implementing the ScheduledRecording Template; one of them is Coherence[5].

The *MediaRenderer Template* defines how to control a UPnP-enabled media player. A control point can start and stop media playback and control some basic settings such as brightness, contrast and volume. Together with an AVServer, a control point has full control over the media and its playback. In theory, a control point on a mobile phone can be used to browse the available media content on the servers and provide a URI to a player. This turns the mobile phone into a universal remote control and the TV set is reduced to only showing the media. This scenario is realizable with mobile phones from Nokia's Nseries (S60). Unfortunately, many UPnP set-top boxes only act as control points to an AVServer hiding the internal MediaRenderer. Sometimes

[2] More details can be found at the product's home page; `http://www.twonkyvision.de/`.
[3] The project is hosted on Sourceforge at `http://djmount.sourceforge.net/`.
[4] Product description at `http://www.o2media.es/en/zoltartv/`.
[5] A UPnP/DLNA framework for Linux, BSD and Windows; `http://coherence.beebits.net/`.

the name MediaRenderer is misused for anything that can play content in the UPnP network; irrespective of the fact that it is only a control point for an AVServer and no device following the UPnP MediaRenderer specification. Unlike the AVServer Template, the MediaRenderer Template is not widely deployed.

Internet Gateway Device

A different device, unrelated to media storage and playback, is the *Internet Gateway Device* (IGD). "The Internet Gateway is an edge interconnect device between a residential Local Area Network (LAN) and the Wide Area Network (WAN), providing connectivity to the Internet" [IW01]. The IGD template is implemented in many home network routers.

The template is commonly known for its actions to manipulate the NAT table. Some applications require being accessible from outside the LAN; for example, the *Windows XP Messenger* or the *Windows Remote Access* open listening ports and wait for incoming connections. If an application on a host behind a NAT provides a listening port, it is only reachable from inside the LAN. The NAT must be aware that an application wants to be accessible from the Internet and forward incoming connections on specific ports to the host inside the LAN. A UPnP-aware application can act as control point to the IGD and configure the required port mapping using the *AddPortMapping* action. This is a useful feature for expanding the media network outside the LAN or access media files from the Internet (see requirements from section 2.2.4).

Most applications only act as an IGD control point to manipulate the NAT table, yet the template allows complete configuration of the router from a host inside the LAN. It is possible to setup the required PPP/PPPoE connection for the router to dial out, activate and configure an internal DHCP server, and even configure the name server to be used. In theory, the user does not need the web interface provided by the router to configure it. Microsoft Windows XP or higher can detect a UPnP-capable router and provide a configuration interface in the look-and-feel of the operating system. In practice, some routers have UPnP turned off by default and the user must at least use the internal web server to turn on UPnP support. Implementations of the device template are available for Linux-based routers; for instance, OpenWRT[6] uses miniupnp[7], a lightweight implementation able to run on routers with 4MB flash memory and 16MB RAM.

3.1.2 Security

The UPnP device template does not address any security aspects. There is neither an authentication of control points to a device nor an authorization whether a control point is allowed to call actions on a device. This problem is addressed in the *UPnP Security Ceremonies Design Document* [Ell03a]; its author Carl M. Ellison describes the security layer in more detail in the *Intel Technology Journal* Volume 06, Issue 04 [Ell02].

[6] OpenWRT is a Linux-based firmware for a wide range of routers; `http://www.openwrt.org/`.

[7] Miniupnp is a lightweight UPnP stack used for embedded devices; `http://miniupnp.free.fr/`.

UPnP Device Security Template

Every security model requires some interaction with the user to permit or deny access of a new device to the system; simple plug and play is not possible anymore. The UPnP Security Ceremonies add a *Security Console* (SC) [Ell03c] to the system, implemented in a control point which provides human input. When a new security-aware device enters the UPnP network it may prevent control points from calling any actions on itself until it is associated with a security console. Besides the templates for the services provided by the device, it supports the *Device Security Template* [Ell03b]. The user can use the security console to locate the new device and take the ownership of it using the *TakeOwnership* action.

For initial peer authentication the user must verify a security ID and enter a password on the security console. The security ID is a subset of the SHA-1 hash sum of the device's public key and should be unique among all devices. The password could be a static password printed on the back of the device, or if the device has an LC display a random password could be displayed to the user. The new device is associated with the SC after the pairing process and they exchange public keys for future authentication, encryption, and to ensure data integrity. A device can be shared between several security consoles by calling *GrantOwnership* with information about the other SC to the device.

A security console announces itself on the UPnP network and control points may attach themselves to it using a similar mechanism as devices; this is a small violation of the UPnP design because a security console is a control point and not a device. When a control point is owned by a security console it queries the security console about certificates the SC issued for it to communicate with devices (*GetMyCertificates*). Beforehand, the SC added the certificates it issued for the control points to the devices. The certificates are used to secure the SOAP messages exchanged between a control point and a device.

When the communication between a control point and a device starts, they negotiate a symmetric session key based on the asymmetric private keys and the certificates. The symmetric key is used to add an HMAC to the XML messages [ERS02]; optionally the XML messages can be encrypted to ensure privacy. The key has an expiration time after which a new session key has to be negotiated.

"The main purpose of security ownership is to establish who is permitted to edit a device's ACL." [Ell03a, section 6]. A security-aware device supports actions for a control point to read the current *Access Control List* (ACL) and to write a changed version back. Only the security console that has ownership of the respective device is allowed to call these actions. What operations the ACLs grant are outside the scope of the security document; each security-aware device defines its own access names it uses in ACLs.

Security Analysis

Device security is an optional feature and products can get a UPnP-compliant logo or a DLNA certification without supporting the secure access. Current consumer electronic routers often

include a web server to configure the device using a web browser. They often implement a password-secured access to the web interface, but expose the UPnP stack without a security layer to the local network. As demonstrated by Pasto and Petkov it is possible to reconfigure a typical UPnP-enabled router with a malicious flash applet [PP08]. The fact that an attacker could use this to change the name server of the router makes it a damaging security risk. Attackers could force the usage of a malicious name server to support phishing attacks. Other attacks involve exposing file servers to the outside by forwarding the required ports. Many home users do not protect the SMB/CIFS network shares with a password making it possible to access all files remotely once the port mapping is in place. An even more serious attack on some routers is to configure it to forward traffic from one host from the outside to another. In this case the attacker uses this relay for illegal activities and the owner of the router is the suspect for this activities [Hem06].

These attacks show that UPnP device security should be mandatory; at least for Internet Gateway Devices and probably for all UPnP devices and control points. However, the "[...] UPnP Security Ceremonies specify a relatively complex set of protocols and procedures, which make designing a secure UPnP-compliant device both expensive and time-consuming." [Sel06]. The certificate handling is the most problematic and confusing part of the specifications—it may be one reason why most devices do not implement UPnP device security.

Moreover, the Device Security Template itself is vulnerable to attacks due to usability issues [D'H06]. An attacker inside the LAN can scan a new device for its services and public keys and use a denial of service attack to prevent it from searching for a security console. The attacker then creates a public key with a security ID similar to the one of the new device. The SC only sees the attacker and displays the false security ID to the user. This points out the importance of usability: the security method itself is not broken. The attacker's device does not have the same security ID, only a similar one. The security ID is an alpha numeric string and most users will not verify every digit.

Other attacks rely on implementation errors of popular UPnP stacks. The Windows XP UPnP stack had a buffer overflow which made it possible for on attacker to take complete control over the host [Haq07]. Since it is not possible to activate UPnP support only when in certain "safe" networks, mobile devices with a vulnerable UPnP stack are at a high risk of being compromised. A laptop running an out-of-date Windows XP version with this buffer overflow has to enable UPnP to control UPnP devices inside the LAN and thus exposes a security risk when the laptop is in a public Wi-Fi hotspot. Many Windows users are not even aware of the fact that the operating system provides UPnP support, nor do they understand what it means. The bug is fixed now, but it is hard to eliminate any possibility of other errors with a similar impact. With a good security layer unauthorized access could be detected at the beginning of the communication, meaning implementation errors in specific templates will not be exposed to everyone anymore.

Another reason why UPnP device security is not widely deployed may be the fact that the documents are very vague about access control lists. Various devices with the same services but from multiple vendors could require a completely different access configuration. While UPnP itself is very user-friendly, setting up access control lists and handling certificates is not.

There are some solutions making UPnP more secure without implementing UPnP Device Security. One simple solution is to use HTTPS instead of HTTP for the SOAP communication. A small change to libsoup, an open source HTTP client/server library, adds HTTPS support to GUPnP, a popular open source UPnP stack.[8] While this makes UPnP more secure, it breaks compatibility with all existing UPnP devices and control points. This solution does not include an automated certificate distribution and requires the user to manually install the peer certificates; this is not very user-friendly.

A different attempt to add some security is integrated in the Windows XP UPnP stack: when a control point first contacts the PC, the user has to manually accept the new control point. This solution provides the illusion of a secure UPnP implementation to the user, but does not prevent simple attacks such as a man-in-the-middle or an attacker posing as a known control point.

3.1.3 Extending UPnP

When extending UPnP to be usable outside the home network we face two major challenges: the SSDP-based discovery mechanism requires IP multicast and the SOAP/GENA-based unicast communications require that devices and control points must be accessible from outside the LAN. The second problem could be solved by requiring IPv6 and a home network with a public IP address for each host. Unfortunately, IPv6 support is still not widely deployed and it is unclear whether the subnet provided by the ISP for private customers will have fixed IP addresses or is a dynamically changing subnet.

In *Remote Access to Universal Plug and Play (UPnP) Device Utilizing the Atom Publishing Protocol* [BS07] the authors use a tunneling mechanism to connect the home network and a remote network. The tunnel can be created with existing *Virtual Private Network* (VPN) solutions and solves the addressing problem for all TCP traffic UPnP requires (e.g. HTTP, SOAP, GENA). The link-local multicast-based SSDP messages however will not work with the tunnel. Even if the two networks can reach each other, they are still two distinct networks and do not share link-local multicast traffic.

The authors suggest a new device called *UPnP Device Aggregator* to create a server in one network handling the service discovery. This aggregator will work with unmodified UPnP clients in the home network, but requires enhanced UPnP clients in the remote network. The remote devices use the aggregator for discovery. The communication between external devices and the home network is realized using the *Atom Publishing Protocol* (AtomPub or APP). The Atom-based service discovery requires external service providers to poll for device changes. It is unclear why the authors do not add a special aggregator in the second network as well to tunnel the IP multicast traffic.

Without going deeper into the details of the aggregator, this setup points out the main problems for expanding a UPnP network outside the home network:

[8] A patch to libsoup is available at the GUPnP bug tracker. As of December 2008, it is not included in the official libsoup source tree. `http://bugzilla.openedhand.com/show_bug.cgi?id=1227`

- A VPN tunnel is required to make sure a control point outside the home network can reach a device inside. UPnP uses IP addresses in the SOAP messages which must be accessible. Additionally, a dynamically changing public IP address of the home network can only be covered by a VPN tunnel providing fixed IP addresses for all devices and control points in the private network. A UPnP control point does not expect IP address changes for a device it is communicating with.
- The paper assumes that there is a VPN tunnel between the two networks but does not cover how it is created. Having dynamic public IP addresses for both networks, an external discovery mechanism is still required. Setting up a virtual private network is also very complicated and not something the average user is familiar with. Besides, both private networks may be using the same IP address range creating routing problems and two clients may have the same IP address. Since many home network routers use 192.168.1.0/24 and choose 192.168.1.1 for themselves such a collision is very likely to happen.
- If a user expands the home network into a friend's home network, the UPnP devices of both media networks are mixed up. UPnP device security is required for all devices to limit the access. Interaction between the two UPnP networks is not possible since the device and the control point must be owned by the same security console.

These observations underline that UPnP is designed to operate in the local home network. Even the solution to eliminate dynamic IP addresses and NATs by requiring IPv6 addresses for all devices does not work without additional changes to the UPnP protocol stack. A new discovery mechanism is required since the IP multicast messages are link-local and will not be delivered to a remote network. Additionally, the ISP may only provide a dynamic IPv6 subnet to the user with dynamic IPv6 addresses for all devices and control points. This must be covered by the discovery mechanism.

3.1.4 Comparing UPnP to the XPMN Requirements

UPnP satisfies many requirements from section 2.2. It has an extensible core and provides many higher level profiles on top of it (requirement 5). It also "just works" without any user interaction if the devices do not implement device security. This is the major criticism of UPnP: the fact that device security is optional and most devices do not implement it makes UPnP vulnerable to attacks. The existing security framework is too complex for both developers and end users. The available attacks on UPnP Internet Gateway Devices show that even link-local operations require a basic security layer; a security layer is mandatory for the XPMN architecture. With the security layer required, a user could add access to friends and restrict access for some family members (requirement 24).

UPnP is limited to the LAN because it uses link-local IP multicast for device discovery. Furthermore GENA requires that devices must be able to open a connection to a control point. In general, this is impossible with current home networks behind a NAT and external devices

or control points. To use UPnP for the media network as defined in the previous chapter, the following steps must be taken to enhance UPnP:

- UPnP device security must be required for each UPnP device on the network. Current attacks have shown that even the home network itself is not secure. Authentication, authorization, data integrity, and encryption are mandatory when a device is exposed to the Internet. For usability reasons the pairing process must be changed; comparing a cryptic security ID is not acceptable.
- IP multicast used for the service discovery is blocked for private customers by most ISPs. This means that a different discovery protocol is required to locate the home network (dynamic IP address) and devices therein.
- Devices and control points must be able to penetrate the NAT.

These three basic changes require massive modifications to the UPnP device template and raise the question whether UPnP should be changed at all, or if it is a better idea to define a new core for the extended personal media network. Changing UPnP to meet the requirements would render the protocol suite incompatible to currently available products anyway.

3.1.5 Summary

UPnP provides a solid framework for a local media network. It defines an extensible core with various service templates on top of it. The defined templates cover most of the use cases including the separation of media server and playback device, making it possible to play media files and schedule recordings from any control point in the UPnP network. One important feature of UPnP is that it keeps its promise: it is plug and play. Unfortunately, the security concept breaks this promise. The user has to verify cryptic security IDs and define complex access control lists.

The lack of a lightweight security protocol is the main drawback. Another often mentioned counter-argument is the fact that UPnP claims to use "Internet-based technologies: UPnP technology is built upon IP, TCP, UDP, HTTP, and XML, among others." [Cor06]. The usage of IP, TCP and UDP does not make a protocol special. Some papers list SSDP, GENA and HTTPMU as three of the "other" technologies; these specifications are only expired Internet Drafts in the IETF, submitted by Microsoft employees outside the context of a working group. The latest versions of SSDP and HTTPMU expired in April 2000; GENA even expired in January 1999. Since then, there is no active development on these drafts to make them standards-track RFCs in the foreseeable future.

Microsoft, one of the founders of UPnP, has abandoned the generic UPnP approach and created a new technology called *Windows Media Center Extender*. The core of this technology is a Windows PC in the home network running the Microsoft Windows Media Center Edition. Viewing devices can connect to this PC to show the media streams and files as well as the media

center user interface. Unlike UPnP, the specifications for Windows Media Center Extenders are not publicly available.

UPnP cannot be used for the extended personal media network outlined by the requirements. Mainly because it lacks an effective and user-friendly security mechanism and it is limited to the LAN. It would require massive changes to the core to overcome these limitations. Nevertheless, UPnP has some good concepts that should be kept in mind when creating the XPMN architecture and specifying its protocols.

- UPnP distinguishes between devices and control points. This creates a client/server structure with a device being the server and the control point the client to this device. Requirement 2 defines the separation of service provider and controller with the same effect.
- UPnP defines a simple core protocol and service templates on top of it. This makes it easy to create new services based on new use cases without redesigning the core.
- The service templates are kept small and simple. Each functionality is defined by a service template. Instead of defining one large media access service, the AV components are split into media server, media transport, and scheduling.

The templates' realization in consumer electronic products differs. There are many UPnP-capable storage devices on the market and the extended personal media network should be able to include them using some sort of bridge to the UPnP-world.

While most routers and network storage devices for the home network support UPnP, it is not implemented in other equipment that should be part of a media network: DVD players, standalone DVB recorders, and IP-TV set-top boxes have no UPnP support and cannot be integrated into a UPnP-based media network. Predictions about interoperability of all components in the home [RET02] have not come true—users still deal with several remotes for the various devices and the usage of a set-top box is limited to the TV set it attached to.

3.2 Other Local Coordination Technologies

There are other coordination technologies in the home network besides UPnP. This section briefly describes Zeroconf Networking, D-Bus, and Mbus; only a small subset of available protocols. They can serve as an model for a future media network, may contain concepts that should be included in the architecture, or have substantial deployment so that including gateways for these services must be considered.

3.2.1 Zeroconf Networking

Requirement 16 demands a device and service discovery and the *Zeroconf* protocol suite is a set of protocols for automatic network configuration and service discovery. Zeroconf is included

in Apple's OS X and modern Linux distributions and thereby widely deployed. It has some relevance to the a media network as well: iTunes uses parts of Zeroconf to enable media sharing between hosts. Therefore, we should take a closer look at Zeroconf.

The Zeroconf set of protocols was initially developed by Apple as an IP-based replacement for AppleTalk, a set of proprietary protocols developed for the Macintosh computer in 1984. Its main purpose is an automatic network configuration of hosts and applications, to locate services in the network, and to open ports for hosts behind a NAT. The *Zeroconf Working Group* in the IETF developed a protocol for *Dynamic Configuration of IPv4 Link-Local Addresses* (RFC 3927) [CAG05]; UPnP uses a draft version of that protocol for Auto-IP (see section 3.1.1).

This section outlines three Zeroconf protocols: *Multicast DNS*, *DNS-based Service Discovery*, and the *NAT Port Mapping Protocol*. More details can be found in *Zero Configuration Networking: The Definitive Guide* by Steinberg and Cheshire [SC05].

Service Discovery

Multicast DNS (mDNS) [CK09] was developed to use link-local multicast for DNS service discovery without a dedicated DNS server in the LAN; the DNS query is sent using IP multicast. Every mDNS-aware host is part of the `.local` domain and can answer DNS queries. This makes it possible to resolve host names to IP addresses in the LAN without a DNS server.

DNS-based Service Discovery (DNS-SD) [CK08] describes a convention for naming and structuring DNS resource records. DNS-SD relies on DNS Service Records (SRV) [GVE00] and DNS TXT Records [Moc87]. The DNS TXT records are used to describe a service; for instance, a printer can be described with attributes such as paper size, its location, and a TCP port to configure the printer using a web browser.

Multicast DNS together with DNS-SD create a very powerful zero configuration service discovery in the LAN. Sometimes the combination of these two protocols is called Bonjour; formerly known as Rendezvous. Bonjour and Rendezvous are trademarks of Apple for its implementation of the protocols. Instead of configuring the device properties and TXT records on a local name server, the device itself answers multicast DNS queries. A host can query for printers in the local network and receives a list of available devices, including the above mentioned additional information. Similar to UPnP, an administrator does not have to reconfigure the existing infrastructure when adding a new host or service.

The proprietary *Digital Audio Access Protocol* (DAAP) introduced by Apple in its iTunes software to share media files between iTunes clients is based on mDNS and DNS-SD. The iTunes software announces itself using mDNS and includes a port to an internal HTTP server in the service description. Additional access restrictions are included in DAAP; while every host can discover a remote iTunes application, it cannot access the music files without prior authorization by the remote user running iTunes. There are only few details known about how the DAAP authentication works; as of September 2009, the iTunes 7.0 DAAP authentication has not be reverse engineered.

Multicast service discovery is included in most network printers available today. Apple includes Bonjour in Mac OS X and their Wi-Fi access point Airport. They released the source code of mDNSResponder, an mDNS implementation for Mac OS X, Windows, Linux, Solaris, FreeBSD and VxWorks. Since iTunes uses mDNS internally to allow music sharing with friends, every Windows host running iTunes already has multicast DNS support, too. Apple provides an SDK for Windows developers to use mDNS in an application. Modern Linux distributions come with the Zeroconf daemon *Avahi*.

DNS-SD is not limited to mDNS and can be used with DNS servers, too. An administrator can add important services such as HTTP servers and printers directly in the domain's DNS server and any client can browse the list of services. This technique is not limited to the local network; it is possible to add service records to any domain or sub-domain. The DNS server software BIND supports remote updates that can be used to update a DNS record for a host including it services.[9] Apple uses wide-area Bonjour in *Back to My Mac* which is part of *MobileMe*, a collection of online tools Apple offers to its customers. The software on the Mac provides the host name and available services to the MobileMe DNS server. The DNS server only covers service announcements and browsing. A host behind a NAT needs to register its external IP address and must forward the respective port on the router.

Lee et al. introduced a different approach how to expand the local service discovery outside the LAN using peer-to-peer networks [LSKD07]. An application in the LAN gathers information about multicast service announcements and uploads that information into a peer-to-peer network. Another client in a different subnet then can download this information to locate external services. The paper neither covers how to connect to external services if they are behind a NAT nor does it address the concept of friends to limit the information about the services to a small group of people. This is a requirement for the XPMN architecture (requirement 24).

The *DNSEXT Working Group* at the IETF standardized *Linklocal Multicast Name Resolution* (LLMNR, RFC 4795) [ATE07] with many similarities to mDNS. LLMNR also uses multicast DNS queries for link-local name resolution. The last call for making LLMNR a proposed standard in 2005 started a heated discussion on the IETF-Discussion mailing list. The discussion was started by Stuart Cheshire, the author of mDNS.[10] In his opinion mDNS is superior to LLMNR and he implies that LLMNR is motivated for company political reasons: Zeroconf is supported by Apple while LLMNR was written by Microsoft employees. Today, LLMNR is an RFC and mDNS is not. "The problem with LLMNR is that it is deliberately limited to prevent it from being used for service discovery, which, you may recall, was the whole motivation for beginning this work in the first place. LLMNR is presented as being the 'official' sanctioned successor to mDNS, as if it were somehow equivalent in functionality but better designed, while

[9] See `http://www.ops.ietf.org/dns/dynupd/secure-ddns-howto.html` for details about secure dynamic DNS to allow clients to change the DNS record and `http://www.dns-sd.org/ServerSetup.html` for a wide-area Bonjour tutorial using BIND.

[10] The mailing list thread of the discussion is titled "Re: Last Call: 'Linklocal Multicast Name Resolution (LLMNR)' to Proposed Standard". Stuart Cheshire's arguments starting it can be found in the IETF mailing list archive at `http://www.ietf.org/mail-archive/web/ietf/current/msg37103.html` (August 2005).

in fact it is so self-contradictory and nonsensical that it actually does nothing useful at all." (Stuart Cheshire in his mail starting the discussion).

The service discovery is required for the media network, a simple name resolution is not enough (see requirement 16 in section 2.2.6). Hence LLMNR does not provide any benefit for the XPMN architecture.

NAT Traversal

The NAT Port Mapping Protocol (NAT-PMP) [CKS08] provides a similar service as the port mapping action of the UPnP Internet Gateway Device. It can be used to forward specific ports of a NAT on the transport layer to a host located behind that particular NAT. Unlike UPnP, configuration of the router is not possible. NAT-PMP does not rely HTTP/SOAP and uses a different very light-weight protocol instead.

The messages are very simple. The host sends a mapping request to the router containing the internal and external port, and a time span how long this mapping should be active. It is possible to remove the mapping before the timeout and also to renew the mapping before it expires.

A router supporting NAT-PMP also broadcasts external IP address changes in the LAN. This is important for hosts inside the LAN to be aware of dynamic IP address changes and to know when to reconnect to a host on the outside. Without that information an application would not notice that it has an invalid TCP connection until it tries to send something or a timeout occurs.

NAT-PMP is supported by most routers from the Apple Airport series and implemented as an additional NAT penetration method in the miniupnp stack used by OpenWRT routers. Most routers do not support NAT-PMP and rely on UPnP IGD instead. For that reason even Apple's Back to My Mac supports both methods for NAT traversal.

Security

Similar to the UPnP core, the mDNS and DNS-SD combination does not include a security layer. But unlike UPnP, Zeroconf is only used to discover the services; the protocol used to communicate with the service is out of the scope of the Zeroconf protocol suite. "Running counter to the basic goals of zero configuration networking, security cannot be automatically configured, but it should still be simple to administer. The Zeroconf WG has identified a set of requirements to ensure that operation using zero configuration protocols will be no less secure than using their correlate configured IETF protocols. Specific mechanisms for simply configuring IP hosts to operate securely (fulfilling these requirements) have been discussed, but no specification has yet been proposed." [Gut01]

If a service needs security, it has to implement it on the application level. A user can use mDNS to discover all HTTP servers in the local network and the servers may require a user name and password using the HTTP authentication mechanisms. Security may not be important for the

service discovery if the service itself is secured. Of course every host in the LAN is able to query the network for services and other hosts. This already may be a security risk for some use cases. For the media network this is not a problem; the access to the LAN is considered secure. If an attacker gets access to the home network, the information how many DVB or storage devices are available is not a security risk; as long as these devices do not grant access without authorization.

Back to My Mac relies on wide-area Zeroconf for service discovery and uses NAT-PMP for NAT traversal. The traffic between two hosts is encrypted by an IPsec tunnel and the authentication is based on Kerberos. However, this is not part of the Zeroconf protocol suite; it is part of MobileMe.

Matching the XPMN Requirements

Multicast DNS and DNS-SD can be integrated in the XPMN architecture for link-local device discovery and service discovery without any configuration. This provides a plug-and-play service discovery similar to UPnP in a very generic way. This approach is much simpler than the complex UPnP protocol stack with HTTP, SOAP and SSDP. "Zeroconf went in a horizontal direction, creating a solid foundation suitable for all IP-based networking protocols. The UPnP Forum went in a vertical direction, creating specific vertical solutions aimed at solving specific vertical problems." [Che]

Zeroconf seems to be a good choice for service discovery in the home network without requiring a special discovery server or any dependencies on an Internet connection. Yet, to expand the media network outside the LAN a second discovery method is required. This second method could rely on DNS Service Discovery with a DNS server without multicast. In this case, a secure way to update a client's service and contact information must be found.

The NAT Port Mapping Protocol can be useful in the media network to allow external entities to connect to a service in the home network. Unlike UPnP IGD, NAT-PMP only handles the forwarding rules, reducing the number of possible attacks on the protocol. This requires a router with NAT-PMP support. However, the XPMN architecture should not require such a router to function since most routers do not support NAT-PMP or an administrator may have disabled it for security reasons. Applications supporting NAT-PMP usually also UPnP IGD to increase the chance of successful port mapping. The media network should not limit the possibilities to penetrate a NAT; UPnP IGD, NAT-PMP, or something completely different. It does not matter as long as a client is reachable from the outside.

3.2.2 D-Bus

Another approach for local coordination is D-Bus.[11] "D-BUS is an interprocess communication (IPC) system, providing a simple yet powerful mechanism allowing applications to talk to one

[11] Specifications can be found at `http://www.freedesktop.org/wiki/Software/dbus`.

another, communicate information and request services. D-BUS was designed from scratch to fulfill the needs of a modern Linux system." [Lov05] There are two major desktop environments for Linux: Gnome and KDE. They used CORBA (Gnome) or DCOP (KDE) as IPC mechanism before D-Bus was developed. These two IPC mechanisms are incompatible to each other, rather slow and complex, and did not really fit the requirements for both desktop environments. D-Bus was designed as a simple IPC mechanism to replace CORBA and DCOP [Wit05]. D-Bus is included in modern Linux distributions and used to control system services as well as user applications such as an audio player and a DVB recorder. Therefore, D-Bus is a possible candidate for the XPMN core.

D-Bus Daemon

Unlike the name suggests, D-Bus is not implemented as a bus architecture; it only supports point to point connections. A D-Bus daemon in a system acts as relay to connect several applications. Each application connects to the daemon and the daemon routes the messages to other applications. With this daemon as relay, the network architecture is rather a star than a bus.

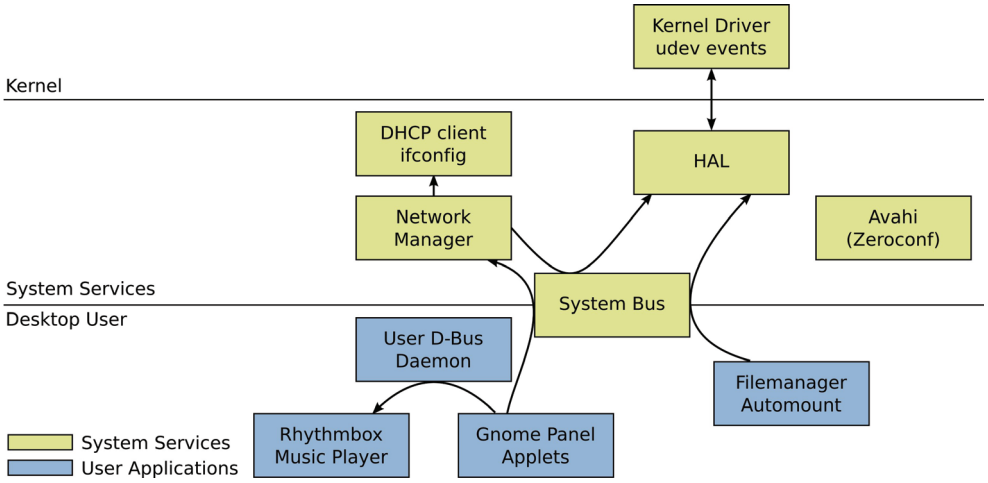

Figure 3.2: D-Bus Daemons and Example Applications

When a user is logged in to the system running the Gnome or KDE desktop environment, two D-Bus daemons are running. One is called the *system bus* to connect user applications with D-Bus-aware Linux services and the second one is the *user bus* connecting several applications of the same user. Figure 3.2 illustrates a typical session with the two D-Bus daemons, services on system level started by the superuser, and some user applications of a Gnome session.

The system bus was a huge step forward for the Linux desktop. It was a new way to interact with parts of the system that require superuser privileges. The first important application using the system bus was the *Hardware Abstraction Layer* (Hal).[12] Previously only the superuser was

[12]More information can be found at `http://www.freedesktop.org/wiki/Software/hal`. Hal is deprecated and in upcoming versions of the major Linux distributions it will be replaced by DeviceKit. DeviceKit is controllable over D-Bus, too.

allowed to mount devices such as USB flash drives and CD-ROM drives. It was possible to allow a user to mount and unmount specific devices, but the superuser had to define this list of user mountable devices. This approach did not work well since it is impossible to predict what devices will be connected to the system in the future. With Hal, a user application can monitor the system for new devices and mount and unmount them. Hal is running with root privileges and is connected to the system bus. A second important D-Bus-aware service is Avahi, a Zeroconf daemon (see section 3.2.1).

Naming, Service Discovery and Communication

An application gets a connection identifier when it connects to a D-Bus daemon. This identifier is unique during the lifetime of the daemon. The application gets a new identifier if it disconnects from the D-Bus and reconnects again. It may register a *well-known* name at the D-Bus daemon, for example, Hal uses the well-known name `/org/freedesktop/Hal/Manager`. The application can be addressed with that name by other applications without knowing the unique identifier given by the Hal daemon. If an application wants to communicate with the Hal daemon, it asks the D-Bus daemon for the unique identifier of Hal and either gets an answer (Hal is connected) or an error (Hal is unavailable). There is no service discovery for D-Bus; an application can neither browse the list of connected applications nor browse the list of services an application provides. The knowledge how to communicate with a service using the well-known name must be added to the application by the programmer.

The remote application API exposed over D-Bus is based on the programming paradigm of Glib, the base library for Gnome: it provides (asynchronous) remote procedure calls and events. An application can connect to a remote event and receives the event message once the remote application emits it; for instance, an application registered to the Hal event `DeviceAdded` gets notified once a new device is added to the system. All commands and events are fully typed; the available types include bytes, integers and double, as well as UTF-8 strings and key/value pairs. Similar to objects in object oriented programming languages a list of commands and events can be grouped; this group is called an interface. For example, the `DeviceAdded` signal provided by Hal is part of the `org.freedesktop.Hal.Device` interface. The interface name is used to avoid naming conflicts between distinct applications. Example 3.1 is a small Python script to list all available devices from Hal (synchronous operation).

```
import dbus
bus = dbus.SystemBus()
hal = bus.get_object('org.freedesktop.Hal', '/org/freedesktop/Hal/Manager')
manager = dbus.Interface(hal, 'org.freedesktop.Hal.Manager')
print manager.GetAllDevices()
```

Example 3.1: Python D-Bus Example

There is no standardization group for communication above the D-Bus core. A developer can choose an interface for the application based on its internal structure similar to defining a local

API using header files for C libraries. As a result several applications with the same purpose have multiple well-known D-Bus names and interfaces. If an application wants to control a running music player it has to use a distinct D-Bus API for each media player it wants to support. A generic media player API is missing.

Security

The security layer is tied to the Linux security model based on users and groups. The daemon uses UNIX domain sockets by default. POSIX access restrictions for that socket restrict access to the user bus to applications run by the same user. The system bus knows the user identifier of the connecting application and can use UNIX groups and configuration files to determine whether or not an application is allowed to communicate with a specific service. D-Bus was written for the Linux desktop and not for servers and allows most operations for the local user. An administrator can restrict access to specific users.

Because UNIX domain sockets are host-local only, they prevent attackers from the outside from compromising the system. In theory it would be possible to connect applications to the D-Bus daemon using TCP sockets, but there is no security concept for this scenario. D-Bus relies completely on the user and group access control of the operating system.

Comparing D-Bus to the XPMN Requirements

Obviously, D-Bus as it is now is not suitable for the envisioned media network because it is limited to host-local communication. Unlike the other protocols presented in this chapter it does not even support link-local communication. Yet, this restriction makes the security layer very simple, easy to implement, and very user-friendly. D-Bus "just works" without any configuration from the user; modern Linux distributions provide reasonable default security settings.

To use D-Bus for the XPMN core it must be opened to applications outside the host using TCP sockets. As this would compromise the security layer of D-Bus, a complete redesign of that layer is needed. Moreover, the star architecture may be a good solution for host-local communication, but creates a single-point-of-failure when several hosts are involved. If the host running such an XPMN D-Bus daemon is offline, the other hosts cannot communicate anymore.

Even though D-Bus is unsuitable for the XPMN core, D-Bus and the D-Bus API should be kept in mind when specifying the XPMN core architecture. The concept of commands, signals and groups is an important part that makes adding D-Bus support easy for developers because it is similar to an API. Furthermore, there are many D-Bus-aware applications and these applications could be integrated in the media network using some sort of gateway; the media network should not aim to replace D-Bus for desktop IPC.

D-Bus lacks a central standardization group to define vendor independent interfaces and there is hardly any documentation of an exposed API available. A developer is often forced to read the

source code or the *Glib XML Introspection* file for Gnome applications. A vendor independent system requires standards, detailed examples, and descriptions.

3.2.3 Message Bus

The *Message Bus for Local Coordination* (Mbus, RFC 3259) [OPK02] was developed at the TZI and the UCL for interactions between components of a larger, distributed system. Until 2008, the various parts of the media center software Freevo used Mbus to interact with each other. The Freevo integration covered some use cases from section 2.1, making Mbus worth to look at, even without Mbus being implemented in any well-known software product at this time.

"The local Message Bus (Mbus) is a light-weight message-oriented coordination protocol for group communication between application components. The Mbus provides automatic location of communication peers, subject based addressing, reliable message transfer and different types of communication schemes." (RFC 3259). Mbus is based on IP multicast similar to UPnP and Zeroconf. It was used at the TZI for distributed *Voice over IP* (VoIP) applications splitting the signaling, audio and controlling components into separate programs [OK00]. All entities on the bus have the same permissions and there is no client/server-based infrastructure. Remote procedure calls and an event mechanism were developed on top of the Mbus core in the *Guidelines for Application Profile Writers* [Kut01].

Overview

Each entity on the Mbus has a unique address, optionally containing the entity name and the role the entity has in an application. A VoIP application could have three entities with the roles *signaling*, *audio* and *controller* [OKM01]. To make sure the Mbus address is unique, it contains an identifier generated from the IP address of the host running the application and the operating system's process identifier on that host. A valid Mbus address of the Freevo TV server with the process identifier 1035 running on the host `134.102.218.67` would be `(module:tvserver app:freevo id:1035@134.102.218.67)`.

A message contains the sender of the message, the address of the receiver or group of receivers, a message name, and a list of parameters. The parameters consist of integers, floating point values, strings, and any arbitrary data using Base64 encoding. Mbus relies on UDP and is designed for small messages; the whole message has to fit into a UDP packet.

The receiver address supports application layer multicast by using a part of the address all recipients share. A message to `(app:freevo)` is handled by all Freevo components on the bus. Reliable transport is only possible for unique receiver addresses. The receiver must send an acknowledge message when the message is received. If the acknowledgment is not received by the sender after 100ms the message will be sent again; the message is considered lost without an acknowledgment after a second retransmission.

An entity sends special *Hello* messages to notify others of its existence. The interval depends on the number of entities on the bus to avoid flooding the bus with *Hello* messages when too many entities are available. The minimal interval is specified as one second. If an entity fails to send five *Hello* messages in a row it is considered offline.

Security

Mbus has a small security layer to make sure that only authorized components can communicate with each other. Each message contains a hash sum of the message calculated with a pre-shared key. Optional symmetric encryption can be used to encrypt the message content. There is no existing authentication method defined for Mbus if no shared secret between the devices exists. The user must manually add this shared secret to the configuration file of each entity. This creates a very simple security layer and an application must trust all other applications knowing the shared secret; it is not possible to verify whether the sender contained in the header is correct. A friend's device in the network as described in the last use cases (see section section 2.1.5) will have this shared secret in order to communicate with one other device and as a consequence is able to communicate with all devices without restrictions. To work around this limitation several Mbus multicast addresses could be used, using multiple secrets for the different tasks. The shared secret for the different Mbus communications is exchanged using a bus with a well-known shared "secret". This bootstrapping mechanism is explained in *Dynamic Device Access for Mobile Users* (DDA) [KO03] and implemented in a project done by the TZI.

Using Mbus as the Media Network Core

Mbus was integrated as a communication protocol for the components in the development version of Freevo in 2004 to provide easy link-local communication. The communication setup was reduced to copying the shared secret to all hosts and removed the requirement of providing IP addresses and ports of other components to all applications.

Together with a company manufacturing set-top boxes, Freevo was ported by the TZI to run on a set-top box. Unlike a PC, the set-top box has lower CPU performance and handles all video and audio decoding with special hardware co-processors. Even with Mbus designed for low performance devices, the set-top box sometimes was unable to acknowledge messages in time. Increasing the timeouts in violation to the RFC made Freevo work on that specific set-top box, but the vendor decided to remove Mbus support completely and use a proprietary protocol based on XML-RPC for the components. The set-top box now implements UPnP to interact with other devices in the network.

The same effect was discovered on low performance recording servers by the Freevo community. A dedicated DVB recording server with less than 500 MHz and many recordings scheduled may not send acknowledgments on time. It is possible to solve this by changing some internals

of Freevo, but it points out that Mbus tempts the application developer into using specific programming paradigms such as threads. An application based on a main loop can run into timeout problems quite easily.

The application layer multicast feature of Mbus could not be used because the components required reliable message delivery. A third problem with Mbus was the limitation of a message to the size of one UDP packet. The complete listing of all scheduled recordings including metadata can be larger than the allowed message size. As a work-around the Freevo TV server had one command to list all identifiers from scheduled recordings, and a second command was called for each identifier to retrieve the metadata. This is a very intricate solution and not very developer-friendly.

To summarize, Mbus as defined by RFC 3259 is unsuitable for an extended personal media network outlined by the use cases and the requirements from chapter 2. The limitation to link-local communication, the too simple security layer, the timing problems, and the limited message size require too many changes to Mbus to fulfill the XPMN requirements.

After these experiences Mbus was removed from Freevo in 2008. Mbus was replaced temporarily with a Python RPC solution as transition until the XPMN core developed within this thesis can be integrated.

Kutscher discusses other technologies such as JINI and CORBA in his PhD thesis introducing the Mbus [Kut03]. He comes to the conclusion that while these protocols describe interesting concepts, they have some disadvantages that led to the development of the message bus. Arguments such as the Java dependency of JINI or the complexity of CORBA are strong arguments against these protocols for the media network, too. Especially CORBA, which was part of the Gnome desktop, lacks support from developers which led to the development of D-Bus.

3.3 Lessons Learned

Our analysis of existing protocols used for inter-process communication in today's home networks showed that these protocols are mostly limited to the local network or even only host-local such as D-Bus. While it is theoretically possible to expand some of these protocols outside the home network, the security defined by these protocols is not sufficient to expose the services to the Internet. Security seems to be the number one problem when creating an extended personal media network.

UPnP is a very good example to show the challenges with security versus usability. The UPnP core is simple and "just works" without any user interaction. This makes UPnP very user-friendly—but it provides no security. The complex UPnP device security template makes it difficult for developers to implement security and makes it also complicated for users to make wise security decisions. It is secure, but not usable anymore.

Some ideas and protocols seem to be very useful for the XPMN core architecture: Multicast DNS and DNS-based Service Discovery can be used for local service discovery and the NAT

Port Mapping Protocol can be used to connect to local devices from outside the home network. The UPnP philosophy with a core and profiles on top of it makes it easy to extend the media network in the future.

The analysis of existing protocols leads to the following observations that must be kept in mind when developing the XPMN core architecture:

1. IP multicast provides an easy service discovery in the LAN and should be used for not having to rely on a specific server in the home network. The whole media network does not function anymore if such a server is not available. Multicast DNS and DNS-SD seem to be a reasonable choice for this task. For additional external components an external (maybe server-based) discovery mechanism is required. This external mechanism could be based on wide-area DNS-SD.

2. The XPMN logic should be divided into service providers and controllers to create a client/server infrastructure similar to UPnP devices and control points. A control point uses a service provider to fulfill a specific task similar to a client using a server.

3. A security layer must be mandatory and integrated into the core. If access control lists and certificates are used, the complexity of them must be hidden from the user. The user should not be forced to check cryptic security IDs. The user should either verify something human readable or provide a short password.

4. A service should define some sort of interface a controller can use to interact with the service provider; in the UPnP terminology these commands are called actions, Mbus calls them remote procedure calls, and D-Bus calls them methods. For internal changes of the service an event mechanism is required. All analyzed protocols have these two mechanisms and they are often used programming paradigms.

   If actions specify a timeout for the remote entity to react, this timeout has to consider lower CPU performance and delays in the network. The Mbus integration into Freevo has shown that small timeouts can be problematic even inside the LAN.

5. UPnP-aware Network Attached Storage devices should be integrated into the media network using a gateway to UPnP services. It took several years to widely deploy UPnP AVServers and Internet Gateway Devices and it is necessary to support at least UPnP AVServers to include the external storage devices available today.

   Many desktop applications on Linux systems provide a D-Bus API which can be used to connect these applications to the media network. Since there is no standard D-Bus API for a category of applications such as media players, support for each application must be added separately.

Since the protocols used in a the LAN cannot be easily changed to fit the XPMN requirements we have to search beyond the home network. The next chapter will describe some protocols and ideas used outside the LAN. The purpose is to find a suitable core protocol for the media network that can be expanded with profiles similar to UPnP.

Chapter 4

Beyond the Home Network

The previous chapter covered some protocols and products available in home networks today. None of these protocols lends itself to be expanded outside the local home network. We have to take a different approach and look at protocols used outside the home network to determine whether they can be used as XPMN core protocol.

This chapter describes four possible technologies for the XPMN core: based on available web technology, peer-to-peer networks, the *Session Initiation Protocol* (SIP), and the *Extensible Messaging and Presence Protocol* (XMPP). The focus of the study lies on the requirements the protocols from the previous chapter cannot solve when expanded outside the home network: peer discovery, the message exchange between clients even though one or both clients may be in a private subnet behind a NAT, and end-to-end security.

4.1 Web-based Applications and Services

Most PCs and many mobile phones have a web browser installed, and a web-based interface would allow the user to reuse an existing application to control the media network—even if the media network itself is not built upon web-based technology. Additionally, there are many media-related websites and web-based applications that could be integrated. This section describes some web-based applications, protocols, and research activities similar to the extended personal media network. Afterwards, we discuss how to integrate external web-based services.

4.1.1 Orb

Orb[1] is a server-based solution to access media files from an "always-on" PC at home running the Orb software. The PC connects to the *MyCasting* server on the Internet. The server provides a web-based interface for mobile devices to access the files from the home server. Orb makes it

[1] Developed by Orb Networks, Inc.; `http://www.orb.com`.

possible to access media files from the home network at any place; the user only needs a web browser. The data between the home server and the Orb server is encrypted using a proprietary protocol. The communication between the mobile devices and the Orb server is based on HTTP, HTTPS and RTP. Orb does not provide an end-to-end security layer and the user has no knowledge about what data is transferred between the home server and the Orb server. Thus, the user has to trust the Orb server completely. If an attacker gets control over the Orb server (which is outside the control of the users), the attacker can access all media files of all customers.

The media files are re-encoded on-the-fly and streamed to the mobile device. If the remote device is a PC, the media will be encoded using Microsoft Windows specific formats: *Windows Media Video* (WMV) for video files and *Windows Media Audio* (WMA) for audio. The data is sent from the MyCasting server to the client over an HTTP connection. Orb provides special web pages for mobile phones and the videos will be encoded in a format suitable for the mobile phone and sent using RTP. A special client for the iPhone even allows the user to access the home network without using the web browser. This makes the iPhone a powerful media player, capable of watching live TV and all video files on the home server. In addition, special Orb clients for popular game consoles such as the Nintendo Wii, Microsoft XBox 360, and the Sony Playstation 3 are available. These clients are used to play content stored on the always-on PC in another room. The closed architecture prevents third parties from developing additional clients for other platforms.

The user can mark a file as public to share it with friends. While this makes sense for small files such as images, sharing videos is impractical in most cases. Current DSL connections cannot handle many parallel video uploads and the server needs to be very powerful to create multiple on-the-fly encodings at the same time. Orb acts as a client to a UPnP Internet Gateway Device to access the data directly from the PC behind a NAT. As a fallback, Orb can route everything through the Orb server. This fallback is a bottleneck in the communication. In several tests between May and August 2007 the bandwidth to the Orb server was not high enough.[2] The video encoding parameters are fixed and did not adjust to the problem, resulting in interruptions during video playback.

Besides local files, the software allows to access attached TV cards. Orb provides an electronic program guide (EPG) in some countries, but for most countries this feature is missing. Without direct EPG support the user has to use external sources as TV guide and copy the information to Orb. There is no API for external EPG providers to include their service directly.

Supplementary to content on the home PC, Orb provides access to well-known web-based video sources such as YouTube, AOL Video, and Yahoo Video. The videos will be re-encoded on the home PC to be suitable for the chosen viewing device. The software always uses the same encoding parameters; YouTube videos and TV recordings will be encoded the same. The video

[2] The always-on PC was in a private network behind a NAT in a simulated home network with a 100 Mbit/s link to a laptop with a public IP address on the outside. The bandwidth problems occurred either between the home network and the Orb server or from the Orb server back to the laptop. UPnP Internet Gateway Device support was disabled on the router for security reasons (see section 3.1.2).

will be scaled down and re-encoded by the home server even if the available bandwidth would be sufficient for the unchanged video.

From the consumer's point of view Orb provides most of the features outlined in the scenarios in section 2.1. The implemented security layer is the main criticism: there is no end-to-end security and the user has no idea what information is exchanged between the PC at home and the MyCasting server. The latter could be solved by making the specification open and allow others to implement additional functionality to the system; yet, this may undermine Orb's business model. The lack of end-to-end security would still be an issue, though.

4.1.2 Web-based UPnP Control Points

Orb presents an interesting solution for accessing media files stored in the home network with a web browser. Instead of limiting the access to the always-on PC and relying on a central server, a personal web server in the home network with an integrated UPnP control point could allow a remote user access to UPnP devices at home. A solution by Jimeno et al. allows complete control using a PDA without installing any additional software on the device [JSL+04]. Using any device with a browser to control the media network is very convenient for the user, but does not satisfy requirement 10 stating that a control point on the Internet must be able to access a service inside the home network with the same interface as if it would be in the home network, too. The UPnP SOAP interface is not available for devices outside the home network. Additionally, the solution does not address dynamic IP addresses and requires the web server to be accessible from the outside—most home networks are behind a NAT.

The same is true for Freevo and MythTV, two of the largest media center solutions for Linux. Both include a web server to schedule recordings and access files remotely; how to deal with dynamic IP addresses and a NAT is not covered by the applications—the average user has no idea how to configure the home network to access the web server from the outside. Nevertheless, a user might want such a web-based control point.

Web-based remote control of UPnP devices was investigated further by the TZI and described in Arendt's diploma thesis [Are08]. A control point interacts with UPnP devices and has an internal web server for the user to browse and access the media files. The solution is not limited to UPnP devices and can be expanded to any other protocol, for instance, to controlling D-Bus applications.

One goal of the web-based control architecture was the ability to access the web server without having to create port forwarding rules to make it work with dynamic IP address changes of the home network. This seems to be the main problem with web-based solutions. To solve this, a developed prototype makes public IPv6 addresses for each host in the local network a requirement. Most ISPs do not provide IPv6 addresses for private consumers; therefore, *Teredo* (RFC 4380) [Hui06] is used to tunnel IPv6 traffic over UDP/IPv4 as fallback. Teredo is integrated in Microsoft Windows starting with Windows XP Service Pack 2 (disabled by default,

enabled by default for Windows Vista or higher), Linux, and Mac OS X[3]. "Teredo is designed to provide an 'IPv6 access of last resort' to nodes that need IPv6 connectivity but cannot use any of the other IPv6 transition schemes." [Hui06, section 3.2]. The protocol uses an external Teredo server to provide IPv6 connectivity. Teredo can operate through a NAT if the NAT behaves in a certain way; the technique is similar to *Simple Traversal of User Datagram Protocol through Network Address Translators* (STUN) and *Traversal Using Relay NAT* (TURN) used by Voice over IP solutions (see section 4.3.2 for details about SIP NAT traversal). Teredo uses a relay to route messages between the peers if NAT traversal is not possible. An additional link-local multicast discovery allows two clients in the same local network to communicate without an external server.

A web interface is used to access the UPnP devices in the home network. A UPnP-aware web server supporting IPv6 and IPv4 in the home network can be used to browse and access media files on the UPnP devices only supporting IPv4. The video and audio files are streamed to the client using RTP; images are transmitted over HTTP.

The prototype has proven to be a good starting point for a web-based UPnP control point, but has some limitations:

- The solution only provides an interface to the user; there is no HTTP-based API for a UPnP control point. It is not possible to tunnel the UPnP SOAP messages over IPv6 because they contain IPv4 addresses for subsequent communications—IPv4 addresses that are from a private subnet and unreachable for a remote UPnP control point.
- The remote access is secured using a user name and password combination; encryption and ensuring data integrity is not supported yet. The UPnP network at home is still insecure without UPnP Device Security.
- The web pages on the web server are designed for a UPnP AVServer. Each UPnP template needs special support—there is no generic solution for all possible UPnP services. A controlling website must be individually designed for each UPnP template. If the internal web server does not support a service from one device in the home network, it cannot be controlled.

The big advantage of a web-based UPnP control point is that the user does not need to install anything on the remote controlling device. This makes it possible to control the home network from anywhere without special software to be installed—except IPv6 support. Hopefully, IPv6 will be available for private consumers in a couple of years. Until then, Teredo is a suitable interim solution. The extended media network may include such a control interface for a user. However, the focus of the development in this thesis lies on device communication without human interaction and therefore web pages mapping UPnP actions are not sufficient for the XPMN core.

For the XPMN architecture design, the outcome of the UPnP gateway prototype is that the Teredo IPv6 approach works very well and is a suitable solution to connect to devices in the

[3]Mac OS X does not support Teredo by default, a third party application must be installed.

home network. All modern operating systems support IPv6 and Teredo can be used to obtain an ISP independent public static IPv6 address. On the downside, users have to enter IPv6 addresses which are very cryptic and not user-friendly; a name server entry for the host or a device discovery mechanism would solve that problem. Teredo is an alternative for NAT penetration to a UPnP Internet Gateway Device or a NAT-PMP-capable router.

4.1.3 Web-based External Services

Some services or media content in the media network may be provided by third parties. One important aspect of the so-called *Web 2.0* is the concept of user-generated content, turning ordinary users into content providers, and by that, making some consumers independent from the local TV broadcasters.

Two popular web-based media-related services are YouTube and Flickr. YouTube is a video sharing platform where users can upload and watch short video clips. The videos are shown within a Flash application embedded in the website. Flickr is an image hosting site and one of the first successful Web 2.0 applications. Users can upload their photos and Flickr provides multiple photo streams (RSS/Atom feed) for each user. Furthermore, they can form groups and apply photos to feeds with a specific topic. YouTube and Flickr are only two examples, there are several copycats providing similar services.

Let us take a closer look at Flickr to identify how web services may be integrated as XPMN service providers. Flickr provides an API for developers to integrate the service into their applications, for instance, a photo manager software can use the API to allow the user to upload photos directly from inside the application. The API provides search methods on the Flickr image database and RSS/Atom feeds from users, groups and tags. This makes Flickr a very useful media network add-on:

- Flickr is just another place where photos are stored. The user marks some photos stored somewhere in the media network as "publicly available" and they will automatically be uploaded to Flickr.
- Photos that belong to friends can be integrated in the user's media collection by adding the friends' personal photo streams. For the consumer these photos are no different than locally stored photos.
- Publicly available photos can serve as supplemental metadata to another media asset. For instance, when viewing vacation images, the user can enhance the collection of photos by photos from the same location. Technically this can be realized based on GPS coordinates stored in the images (geotagging).

Besides user-generated content, broadcasters and professional studios provide some videos on their websites. Popular examples with an API for developers are the *BBC iPlayer*[4] and the *ZDF Mediathek*[5]. The Linux video player *Totem* includes a plug-in for users in the U.K. to browse

[4]BBC iPlayer channel selection; `http://www.bbc.co.uk/iplayer/tv`.

[5]The Mediathek provides access to shows produced by the ZDF itself; `http://www.zdf.de/ZDFmediathek/`.

and watch videos provided by the BBC. The ZDF Mediathek provides an RSS-based API for developers on request. It is possible to browse parts of the Mediathek and besides the actual video, access metadata such as descriptions and thumbnail images. Unfortunately, most video sites restrict the playback to be used with a browser and do not support external control and downloading of the video files over a web API.

At the CES 2009, several manufacturers introduced TV sets and set-top boxes with integrated support for selected web services. While some rely on an internal browser engine, others added special support for the specific services. The latter makes updates to support additional sites in the future more difficult. YouTube and Flickr are supported by most TV sets and set-top boxes; in the U.S., *Hulu* and *Netflix* are planned to be supported as well. Hulu[6] is a web-based streaming service supported by the major TV broadcasters where users can watch their favorite TV shows online; sometimes even before the official broadcast. Hulu finances its service with advertisements similar to the traditional TV broadcast. Netflix[7] on the other hand is a video on demand service. Originally started as a DVD rental service, the company offers a subset of their movies to be watched on *Netflix ready devices*. These devices include the TiVo HD DVR, a popular digital video recorder in the U.S. with a subscription service, and the Xbox 360. Users can watch as many movies as they like with the *Unlimited Plans* subscription for $.8.99 a month.[8] Due to licensing issues, Hulu and Netflix are not available worldwide and restricted to the U.S..[9]

Similar to TV broadcasters and studios, most radio stations offer a live-stream of their program. There are many Internet-only radio stations besides traditional radio broadcasters available. Some provide an API to receive a personal radio stream based on the preferences and mood of the user. Other web services that may be useful for an extended media network are online TV recorders and online storage solutions. Their technical restrictions must be considered when integrating such services: a web-based storage solution may have a similar interface as a local hard disk, but one can be used to store a live DVB stream while the other is limited by the bandwidth of the Internet connection.

All these web-based services should be integrated into the media network. The central issue with web APIs is that every website defines its own; each mentioned service has a completely different interface.

The Linux-based UPnP stack *Coherence* includes an UPnP AVServer that does not expose the media files stored on the device and uses Flickr instead. It acts as a normal AVServer to the UPnP network and uses the Flickr API in the background to retrieve metadata and a list of available photos based on the user's photo stream settings on Flickr. Besides Flickr, Coherence supports various Internet radio stations and presents the radio listing and the radio stream as UPnP audio objects to a UPnP control point and a UPnP MediaRenderer.

[6] Hulu includes a large selection of videos from more than 100 content providers; `http://www.hulu.com`.
[7] Netflix DVD rental and Video on Demand; `http://www.netflix.com`.
[8] Price from November 2009.
[9] A very limited set of videos on Hulu is available worldwide.

4.1.4 Media Center Web-Pages

Some web services are very user-centric and are never automatically accessed by controllers without a user interface. A weather forecast for the TV set is such a service: its purpose is to provide the information to the user. Any such service can be integrated into an extended media network by including a web browser in the TV set, making use of existing technologies such as (X)HTML, CSS and JavaScript.

There are two major differences between a browser on a PC and on a TV set. First of all, the user is very probably not as close to a TV set as to a PC monitor. This makes it harder to read the content and the websites have to be rendered with a larger font size and images have to be scaled accordingly. For analog TV sets this is even worse because an analog PAL or NTSC signal not only has a small resolution compared to HDTV and computer monitors, the signal quality is also very poor compared to a VGA or a DVI signal used to connect a monitor and a PC. The image is always a bit blurry, making it even harder to read text.

The second difference between a TV set and a PC are the unequal input capabilities. On a PC, the user has a mouse and a keyboard as input devices. The mouse can be used to select a specific element on the website based on x/y-coordinates. It is very fast and precise at the same time. A typical TV set only has a remote control as input device which is inferior to mouse and keyboard. It has less buttons than a keyboard and as a mouse replacement the buttons must be used to move the mouse cursor across the screen. With the processing of each button press in the order of milliseconds, the mouse control either looses speed or precision. For character input such as a keyboard, the remote control lacks the needed number of different keys. Future remote controls could include a small touch pad similar to laptops to compensate for the mouse; the missing alphanumeric keyboard will still be an issue.

Web4CE [DS07] (also known as CE-HTML, part of the CEA-2014 standard) defines how to include web-based technology into the DLNA framework. UPnP devices can use a Web4CE-capable TV set to render a configuration interface. Web4CE defines special profiles for consumer electronic devices, most notably mobile phones and TV sets. It introduces the <op> element to map remote control keys, a *NotifySocket* scripting object to open a TCP stream between client and server to not rely on polling, alpha-blending, and overlaying parts of the website on top of the video. These extensions make Web4CE incompatible to current web browsers.

Philips, one of the companies developing Web4CE, includes this technology in some of its products. Some TV sets can also render web pages not relying on CE-HTML, but lack support for Flash and Java. The Yahoo TV Widgets[10] are based on Web4CE and can be used to access some websites from the TV set. This system is very closed: it seems that Yahoo has to approve every widget and then the individual TV set manufacturers will decide whether or not they offer the widget to their costumers.

The Web4CE specification is not free of charge: developers must pay $200 for it. While this is no obstacle for manufacturers and companies running websites, it may prevent private sites from

[10]Further information can be found at `http://connectedtv.yahoo.com/services/tv-widgets`.

supporting Web4CE-based TV sets. The whole idea is reminiscent of the *Wireless Markup Language* (WML) for mobile phones: it was not widely accepted and is of no relevance anymore. It should be possible to reuse the existing protocols as they are instead of defining something new.

The TZI integrated a web browser into a set-top box during the second phase of the ScaleNet project [BKML08]. The developed browser extensions are based on an implementation by Ebersbach done in the scope of his diploma thesis [Ebe08]. They include keyboard navigation, web page zooming, and mapping remote control keys to special events. A plug-in for video playback in web pages was created enabling the user to watch embedded videos full screen on the press of a button. In addition, Kasten created a framework for web services to be used with set-top boxes in his diploma thesis [Kas07]. His prototype aggregates information from sites such as YouTube and presents them on web pages suitable for the media network: the framework uses large fonts and images, videos can be shown inside the web page or full screen, and the user can navigate on the site with keyboard combinations that can be mapped on a remote control. This solution is compatible with current web browsers. Compared to Web4CE this solution only lacks the special overlay support; typical browsers to not support alpha values for the background color.

4.1.5 Lessons Learned

The analysis of Orb, UPnP control points, connecting web services to UPnP, and integrating others with a web browser shows that web-based services are an important part of an extended personal media network: user-generated content, recommendations, and social networks provide an additional benefit. Some services already provide an API to integrate the service into an external application, but there is no generic API usable for multiple services.

The following observations should be kept in mind when designing the XPMN core architecture.

- Web 2.0 services such as YouTube and Flickr highly depend on user-generated content. Users who are interested in such content when browsing the web may also want to include the content into their extended media network. For instance, there should be a seamless integration of Flickr photo streams; from the user's perspective they are similar to locally stored photos.
- The social structure of some Web 2.0 services provide an additional benefit. Sometimes it is not only the actual content that attracts for the user; personal recommendations, user groups, and user comments are often used and should be preserved when integrating a service.
- While it is possible to use the web APIs and create a gateway to the media network, a much better solution is a standardized API. This would make it possible to add services similar to existing ones without modifying existing controllers. Forcing the user to update a TV set's firmware to include a new external service is unacceptable.

- Some web-based services can be added by integrating a web browser on a viewing device. User-centric services can be created using existing web-based technology; the websites should be enhanced to be used with a remote control on a TV set. Web4CE or the Yahoo TV Widgets are a possible solution, but an open solution compatible to existing web browsers would be preferable. The browser-based integration is outside the scope of the XPMN core; sharing bookmarks between controllers is a possible XPMN service.
- A web-based controller is a useful add-on. A user can access services everywhere at any time; only a browser is required. The media network itself does not need special support for the web front-end; a web server acts as controller to available services without special handling in the core protocol.
- A closed solution such as Orb or Yahoo TV Widgets limits its usefulness. Web services not supported by the Orb software cannot be added by a third party. It is not possible to create an alternative client for the always-on PC because the protocol used to communicate with the Orb server is closed and the security even makes reverse engineering illegal in some countries.

An additional finding, unrelated to web-services, is that Teredo is a suitable solution for NAT traversal and overcoming some restrictions resulting from dynamic IP addresses. As a consequence, HTTP could be used for devices to interact with each other; thus, UPnP based on IPv6 addresses. But as discussed in section 3.1, HTTP lacks a proper subscribe/notify mechanism, UPnP lacks a usable security layer, and the UPnP service discovery is based on IP multicast; therefore, restricted to the home network. Yet, a device discovery mechanism could rely on IPv6 with Teredo as fallback to access a device independently of the actual network topology. In that case the usage of IPv6 should be hidden from the user; dealing with IPv6 addresses directly is not user-friendly.

4.2 Peer-to-Peer Networks

Recently, peer-to-peer (P2P) networking has become a widely discussed topic; often in context with illegal music, movie and software downloads. But P2P is not about file sharing; it is a concept. Unlike traditional client/server communications the nodes are equal and exchange data directly. A peer-to-peer network requires peer discovery, identification, and has to penetrate the NAT for clients in the home network—the XPMN core also has these requirements. A media network can be described as a P2P network with each device being an independent peer.

4.2.1 Architecture of Peer-to-Peer Networks

Harjula et al. grouped peer-to-peer networks into three generations [HYAK+04]: the first generation is very server-centric, the second generation completely distributed, and the third generation is a hybrid of the previous ones. The classification into the three generations is based on

the development of new P2P concepts and does not mean that a first generation P2P network is inferior to a third generation network—which generation to prefer depends on the use case.

When the file sharing application *Napster* was published 1999, it was the first file sharing application based on a peer-to-peer network.[11] Originally, Napster depended on centralized servers for clients to publish the list of files they wanted to share. Searching for specific files was done on that centralized server and the server returned contact information of clients providing the files. Up to this point, this is a classic client/server-based approach. In contrast, the actual file transfer is completely independent of the server and only between the clients (peers). Today, Napster is of no importance anymore and *BitTorrent* is mainly used for (illegal) music and video downloads on the Internet. BitTorrent can also be categorized as a first generation P2P network: each file or group of files requires a server (tracker) to coordinate the clients.

The next generation is the exact opposite. Napster with the index on centralized servers was sued by copyright holders. As a counter-movement *Gnutella* was developed. It does not rely on centralized servers and all peers are equal. Without the server, peer discovery is a big issue and a client requires a small list of peers to bootstrap the peer discovery. It tries to connect to clients known from a previous session (which may not be available anymore) and queries them for a list of other clients. After a while, a client gets to know a part of the available network, but it will never know all hosts in the network. Gnutella suffers from the overhead to announce clients and to search for files. Each client only has a limited view of the P2P network and a search may not return all possible results available. This design is impractically for the XPMN architecture due to the device and service discovery requirements.

These first two generations cover both a very server-centric network and a network where all nodes are equal. The third generation is a mixture between these two. Today, the most popular third generation P2P protocol is *Skype*. Unlike Napster and Gnutella, Skype's purpose is not downloading files; it is a Voice over IP solution. Skype has two kinds of nodes: super nodes and edge nodes. The super nodes are not behind a NAT or Firewall and have a high bandwidth. They form a second generation network among themselves. The edge nodes use the super nodes as access point to the P2P network; they use the super nodes as server. Gnutella added support for super nodes in the protocol at version 0.6; yet, it never dropped out of draft status.

4.2.2 Popular P2P Networks

There are several protocols for peer-to-peer networking. Some are based on a very specific use case (e.g. file sharing or telephony), others are generic overlay protocols. This section lists some popular P2P networks and checks them against three basic XPMN requirements: peer discovery, NAT traversal, and end-to-end security. Maybe one of them is a suitable XPMN core or provides some insight how to design the core.

[11] Napster's brand and logo is now used by a commercial music store which has nothing to do with the technology described here.

BitTorrent

Today, when peer-to-peer networks are mentioned, many people think of BitTorrent. BitTorrent uses a central server (tracker) for each set of files (torrent) providing peer discovery. There is no relationship between torrents or servers to allow searching for available files. Instead, websites are used to exchange BitTorrent files which describe a file and its tracker.

One outcome of the European Union funded P2P research project *P2P-Next* was an extension to BitTorrent to work with super nodes and without a central server for the tracker [Gan08]. Each client may temporarily become a tracker for specific files; no matter whether or not the client is even interested in the torrent.

NAT traversal is not included in BitTorrent directly, some clients have NAT-PMP or UPnP IGD support built in to penetrate the NAT. If the router does not support any of these protocols, it is not possible to connect to the client from the outside. For the given use case this is not a big issue: a user starts the BitTorrent client to download a file, not to upload something. The download will complete because the client can connect to other clients to request a chunk of data. Yet, the download will be slower: BitTorrent rewards peers with a high upload by preferring them in the download queues. One reason is that such a client can upload the new chunks fast to other clients and a second reason is to encourage people to upload. Without uploading clients, BitTorrent does not work; the tracker is not in possession of the files.

The security layer in BitTorrent only provides data integrity, confidentiality is of no importance. The users do not care what peers they use to download a file from as long as the file is correct. Data integrity is provided by the tracker by distributing check-sums for the file chunks.

Skype

Skype is listed here because it is a well-known application, solves the challenge of a suitable NAT traversal, includes a strong security layer, and is still usable for the average user. Unfortunately, the protocol specification is closed and some details described here are only based on observations by Baset and Schulzrinne [BS06].

Skype has both super nodes and a central server which is only used for the login process. Each node must have a list of other nodes on the network to bootstrap the peer-to-peer communication. The super nodes are used for NAT traversal: first Skype tries to punch holes into the NAT (there are similarities to STUN; see section 4.3.2 for details about STUN/TURN) and if this does not work, an edge node uses a super node to route the data to its peer. This is a very efficient way and does not rely on additional routing servers. However, users can neither choose not to relay for others nor do they have a way of knowing whether or not their client is used for relaying data.

The developers put a huge amount of effort into a security layer, encrypting the whole communication between all nodes and to protect the application itself from being reverse engineered.

Since it is a closed network and closed source, it is easier to achieve end-to-end security. One possibility is that each user owns a certificate signed by the central user server. It is unclear how Skype works in detail and whether or not there is a back door. A user has to trust Skype completely. This security concept violates requirement 3, to have an open protocol where users are not forced to trust a server operated by a third party.

JXTA

The previously mentioned P2P protocols are designed for specific purposes. BitTorrent is for downloading pre-recorded files and Skype is a Voice over IP protocol. A generic protocol that could be used for the XPMN core is JXTA [TAA+03]. It is not used in popular products, but often integrated in research prototypes.

A JXTA peer is a virtual communication point with a unique identifier. Several peers form a peer group and only peers inside the same group can communicate with each other. Each personal media network can be described as such a group. JXTA defines special *rendezvous* peers for device and service discovery. The rendezvous peers organize themselves into a loosely-coupled network similar to super nodes. Peers, peer groups, and services are described in an *advertisement* XML document and shared between the rendezvous peers.

JXTA uses special gateway peers to provide NAT traversal for edge nodes. Each peer may become a gateway and can announce this service using the rendezvous peers. Direct communication with NAT-PMP, UPnP IGD or Teredo is possible, depending on the client's functionality.

The security layer relies on *Transport Layer Security* (TLS) and X.509 certificates. Each peer must have the root certificate of all the other peers in a group for a secure communication. How the peers can obtain a certificate the others can verify is not specified; distributing root certificates is not a trivial task to make user-friendly.

BEEP / APEX

The *Block Extensible Exchange Protocol* (BEEP, RFC 3080) [Ros01b] is a framework providing basic connection management, message encoding and decoding using MIME, authentication, and encryption. New protocols can be defined by using the BEEP core and creating a profile on top of it. BEEP itself only defines the communication between two clients and has no support for device discovery, message routing, or NAT penetration. Its generic architecture makes it possible to multiplex several streams into one TCP connection.

The *Application Exchange Protocol* (APEX) defines BEEP profiles for message routing (RFC 3340, RFC 3341, RFC 3342) and presence (RFC 3343). It is an extensible best effort datagram relaying protocol. One possible use case for APEX is instant messaging. End-to-end security for messages routed over APEX relays can be realized with TLS; how the certificates for such a session can be verified by a peer is outside the scope of APEX or BEEP. Furthermore, APEX does not address NAT penetration.

In 2007, Rose, one of the authors of BEEP and APEX, suggested to reclassify the APEX RFCs to Historic: "To the author's knowledge, no implementations of these RFCs were ever deployed. To the author's knowledge, all of APEX's functionality is currently provided by XMPP, which enjoys extraordinarily robust deployment." [Ros07a] Although the APEX RFCs are still in the standards track, the *Extensible Messaging and Presence Protocol* (XMPP) may be more suitable for the media network. It is described in section 4.4.

Windows Peer-to-Peer Networking

Windows Peer-to-Peer Networking [Mic06a] is not a well-known protocol, but integrated into Microsoft Windows since Windows XP Service Pack 2. Therefore, the protocol is widely deployed. Some applications shipped with Windows Vista use this protocol for direct client-to-client communications. The most popular example is *Remote Help* using *Easy Connect*: the user only has to enter a password and has to provide this password and a user-selected host name to a friend. The friend can remotely connect to the PC with these two values.

Like the remote control web server for UPnP devices developed by the TZI (see section 4.1.2) Windows Peer-to-Peer Networking relies on IPv6. This allows NAT traversal using Teredo which has also been part of Microsoft Windows since Windows XP Service Pack 2.

The *Peer Name Resolution Protocol* (PNRP) [Mic06b] specifies the device and service discovery process and uses the P2P network itself. A peer can be a host, a user or a service. The PNRP peer identifier is generated based on a user-defined name, the service location, and optionally a certificate with a digital signature. A remote client needs all this information to calculate the PNRP identifier of the peer it wants to communicate with. The secure identifier based on a certificate requires the remote peer to know the certificate in advance. This forces the user to choose between an insecure human-readable name to provide to a friend and a cryptic secure name. A user needs a signed certificate to create a secure identifier; how to obtain such a certificate is out of the scope of the specification. Therefore, Easy Connect relies on a password to authenticate a remote connection. End-to-end security is not covered by Windows Peer-to-Peer Networking. For secure PNRP identifiers the included certificate can be used with TLS.

PNRP is covered by US patent 7065587, which makes it difficult to support other platforms such as Linux and Mac OS X. Thus, Windows Peer-to-Peer Networking may provide some ideas on how to create the XPMN core, but cannot be used directly.

4.2.3 Lessons Learned

The aforementioned peer-to-peer networks differ in their internal design and they constitute only a small subset of available protocols. For the media network the following statements sum up the lessons we learned from looking at these protocols:

1. The **device and service discovery** is simple with external servers and more complex in a P2P network without. This affects the complexity of the client: a BitTorrent client receives everything it needs to connect to a peer from the tracker, while a JXTA or PNRP network with edge and super peers makes the device and service discovery very complex for a client. A client needs information about the network to bootstrap the communication, super nodes can vanish, and the clients must constantly monitor the network.

   It would be better to rely on central servers to handle presence information, keeping the clients more simple. After all, an XPMN controller should be able run on a mobile phone. To save battery power, a mobile phone client should not be forced to check the network topology all the time. This is a requirement we missed in section 2.2. Additionally, a complex client reduces the acceptance by vendors due to complex design, and by consumers due to possible errors a complex system is more likely to have.

   On the other hand, we do not want to have a central server for all users as implemented by Orb and Skype. It should be a federated network of servers similar to e-mail where users can choose between e-mail providers or even set up an e-mail server on their own.

2. **NAT traversal** always requires other peers or servers to help setting up a connection. The Microsoft Peer-to-Peer Network shows again that IPv6 and Teredo as helper protocol provide a solid base for setting up an end-to-end connection between clients. If a central server is used for the device discovery it can map client names to IPv6 addresses, hiding the complexity of these addresses from the user. Once IPv6 is widely deployed, Teredo can just be removed without changing anything else.

   Skype's super nodes relaying traffic are an interesting way to traverse the NAT, but a client cannot choose to whom this service should be provided. A user may only want to help out friends and not give away bandwidth to strangers.

3. The **end-to-end security** seems to be the biggest challenge for the media network. None of the protocols besides Skype, which is a closed network, provides any mechanism to bootstrap the trust relationship. Some rely on Transport Layer Security which is a well-known technology and provides confidentiality, data integrity, and peer entity authentication. However, they all lack a mechanism to exchange the required client or root certificates.

   A pure password-based authentication solution does not work for devices in the media network interacting without human input. For instance, there is no user involved to enter a password if a TV tuner wants to store something on a remote hard disk.

Therefore, we should build upon a protocol that uses a federated group of servers for the discovery process and allows direct connections between clients. One choice for this kind of session management is the *Session Initiation Protocol* (SIP); the second possible protocol is the *Extensible Messaging and Presence Protocol* (XMPP) mentioned in the proposal to move the APEX RFCs to Historic.

4.3 SIP-based Approach

The *Session Initiation Protocol* (SIP, RFC 3261) [RSC+02] is a signaling protocol mainly used for Voice over IP telephony. In some aspects SIP can be considered to be a first generation P2P network. The servers are required for peer entity discovery and the audio data is transmitted directly between the clients using the *Real-time Transport Protocol* (RTP, RFC 3550) [SCFJ03]. Even though SIP is mainly used to initiate voice sessions between clients, it is possible to create any kind of session and therefore it may be a suitable XPMN core protocol.

RFC 3261 is an IETF standard developed in the *SIP Working Group*; the current name of the SIP Working Group is SIPCore. Besides the SIPCore Working Group working on SIP itself, several others are working on SIP extensions for conferencing, instant messaging, media control, and much more. At the end of 2008, there were around 200 RFCs regarding SIP[12] and many active Internet Drafts.

Prior to SIP, the H.323 protocol suite developed by the *ITU Telecommunication Standardization Sector* (ITU-T) was used for IP-based telephony. Unlike SIP, H.323 has its roots in the telecommunication industry and incorporates standards also used in ISDN. "H.323 met with reasonable success in the marketplace. [...] Development experience led to complaints about its complexity, in particular the complex setup procedure of H.323 version 1 and use of binary message formats for signaling." [Per06]. Even with some deployment in closed company networks, H.323 was replaced by SIP and therefore this thesis will not go into further detail about H.323.

4.3.1 Overview

SIP defines a client/server architecture with clients, called *SIP User Agents* (UA), and *Proxy Servers* relaying SIP messages between them. A client sends a SIP message to a proxy server and the proxy server will either deliver the message to the recipient or another proxy server. If the proxy knows several possible destinations, it may relay the message to all these destinations (forking proxy). The client registers at a *Registrar* which provides this information to a *Location Server*. The location server is used by clients and proxy servers to locate their peers. Depending on the complexity of the topology and the use case, there may also be other kinds of servers involved. Many implementations include multiple servers in one program, for instance, registrar and location server.

A SIP message has a header and a body similar to HTTP. The header contains the SIP URIs of the sender and the recipient, the type of the content, the route (proxy servers), and other content-independent information. The message body contains the content as described in the header and can be empty if no additional content is needed.

A session setup can be very complex with redirect servers and forking proxies. In a simple session setup for a VoIP call, the initiator sends an INVITE message to the recipient over a proxy

[12] A list of SIP-related RFCs can be found at `http://www.sipknowledge.com/SIP_RFC.htm`.

server. Figure 4.1 depicts a session initiation with a forking proxy server and two possible recipients. The INVITE message contains a *Session Description Protocol* (SDP) body describing possible audio codecs and UDP port numbers for the actual media stream. Once the recipient or proxy receives the message, it acknowledges it with a 100 Trying response, a 180 Ringing, and finally a 200 OK when the recipient has accepted the session (picked up the phone). The Trying response may contain a SDP message (early media); the actual media stream negotiation is exchanged in the OK response. At the end of the session initiation, the initiator sends an ACK to the recipient.

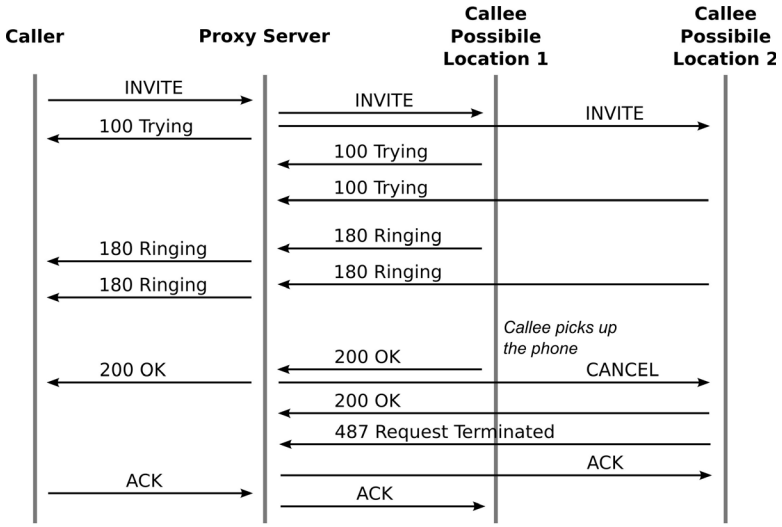

Figure 4.1: Exemplary SIP Session Initiation

The *SIMPLE Working Group* defined RFCs for instant messaging, presence, and a subscribe/notify mechanism for SIP. SIMPLE stands for *SIP for Instant Messaging and Presence Leveraging Extensions*. A newly introduced MESSAGE command is sent over the SIP network similar to an INVITE and contains a small instant message as content. This works without opening a session first and provides a service comparable to SMS in cellular networks. For longer instant messaging conversations the *Message Session Relay Protocol* (MSRP, RFC 4975) is used to set up an instant messaging session over SIP. The clients open a TCP stream to a relay host and use this host to exchange SEND messages. The same TCP connection can be used for several instant messaging conversations. The main drawback of SIMPLE are scalability issues [HAP+09].

The *3rd Generation Partnership Project* (3GPP) uses SIP in their *IP Multimedia Subsystem* (IMS) for the next generation of cellular networks. IMS is designed as a convergence layer to not only support voice calls, but other multimedia services as well. “If care is not taken, the operator networks may become mere ‘bit pipes’ in the business chain and not achieve the proper return of investment. [...] IMS is designed as part of the network operator architecture and to be controlled and owned by it. [...] This includes also the creation of an operator driven ‘trust’ centre: the operator will become the account holder for the customers by offering various mercenarily services and service bundles [...]. The business model proposed by IMS [...] centres the network operator again in the core of the business chain.” [CMVE06] A Video on Demand

service can be built on top of IMS using SIP for session negotiation and RTP for delivering the actual video to the end point. The IMS network is a closed network run by the operators and therefore all services have a trust relationship with each other. This allows additional services such as billing for Video on Demand.

4.3.2 NAT Traversal

The SIP messages are routed over the proxy servers and the actual audio or video data is sent directly between the clients. If one client is behind a NAT, the RTP packets cannot reach that host: the client behind the NAT cannot be addressed from the outside and the SIP messages contain the private IP address and not the public one of the home network.

One of the first solutions to this problem was to include an application-layer gateway into the NAT/firewall. Instead of only mapping the port numbers, the router inspects the SIP message itself, opens the required ports, and rewrites the IP addresses. Besides the fact that this solution requires SIP-aware routers in the home network, it breaks SIP security mechanisms. The application-layer gateway acts as a man-in-the-middle: it needs to read and rewrite SIP messages. The purpose of a security layer is to prevent this.

The *Session Traversal Utilities for NAT* (STUN[13], RFC 5389) [RMMW08] take advantage of specific ways address translation is implemented inside a home network router for UDP traffic. The SIP client sends a STUN request to a STUN server on the Internet and the server returns the source IP address and port number of that request. First of all, the SIP client now knows its public IP address; secondly, the STUN messages to the STUN server created a port mapping in the NAT and the client knows the port number on the other side of the NAT.

Some NAT implementations do not include the source IP address for incoming UDP packets in the mapping (Full Cone NAT); therefore, the client is reachable by all clients outside the NAT on the public IP address and port number in the server's response. Even though the mapping only depends on the IP address and port of the internal client, some routers additionally check the source IP address (Restricted Cone NAT) or even the source port (Port Restricted Cone NAT) of incoming messages. By exchanging messages between the clients, STUN can create the port forwarding for these NAT implementations, too. STUN fails only if the mapping depends on the internal and the peers' IP addresses and port numbers (Symmetric NAT).

STUN opens the port forwarding on the NAT by exchanging the STUN messages. Thus, the actual media must be transmitted over the same ports and the application must be able to distinguish between STUN and RTP packets.

If STUN fails to establish the RTP streams between the clients, the traffic can be routed over an external server. *Traversal Using Relay NAT* (TURN) [RMM09] is a companion protocol for NAT to route the messages over a TURN server. This mechanism is sometimes called reflexive

[13]Originally STUN stood for "Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)". This name was used in RFC 3489, which is now obsolete and replaced by RFC 5389.

STUN. A client requests a forwarding port from the TURN server and includes the server's IP address and this port number in the SDP messages. The TURN server will forward messages on that port to the client. Though TURN works in more cases than STUN does, it relies on an external server. Depending on the location of the server this may increase the latency of voice calls. Furthermore, the voice quality depends not only on the available bandwidth of the two peers, but on the TURN server's bandwidth as well. However, for voice calls TURN is an acceptable solution for NAT traversal if STUN does not work. For the media network TURN may not be possible: High Definition video streams have a very high bandwidth requirement and TURN server operators may limit the available bandwidth per user.

With STUN and TURN integrated into VoIP stacks, one important question remains: the clients need to know when one of these protocols should be used. A direct connection is possible if both clients are behind the same NAT and TURN should be used if STUN is tried and does not work. *Interactive Connectivity Establishment* (ICE) [Ros07b, Ros06] defines a technique to determine the required NAT helper protocol. Before call setup, a client gathers information about IP addresses and ports it can use for the communication. This includes the addresses of the actual interfaces (host candidates), IP addresses and ports determined by contacting a STUN server from each interface (server-reflexive candidates), and addresses obtained by contacting a TURN server (relayed candidates). The gathered candidates are prioritized: a candidate based on a TURN server has a lower priority than direct links, and some interfaces may be known to be slower or more expensive than others, for instance, a UMTS connection.

The call initiator sends all its ICE candidates in the INVITE message. The recipient performs the same gathering and prioritizing steps, and sends its candidates to the initiator. The recipient must do this before sending a RINGING message. For usability reasons, the phone should not ring before ICE completes and the clients know that a voice call will be possible.

Now that both clients have the candidates of their peer, they try every combination of local and remote candidates sorted by the priorities. A client sends a STUN message to the peer and if it receives an answer, it knows that this pair of candidates will work for voice communication. One of the clients, typically the caller, sends the final pair to be used to its peer to be sure both clients use the same pair for the media.

4.3.3 Applying SIP to the Problem

With SIP gaining popularity around 2003, some people thought that SIP is the solution to virtually everything; clients can exchange any kind of message using the SIP routers. This is the reason why the SIP Working Group published the *Guidelines for Authors of Extensions to the Session Initiation Protocol (SIP)* (RFC 4485) [RS06]. "SIP is a protocol for initiating, modifying, and terminating interactive sessions. [...] SIP is not a general purpose transfer protocol. It is not meant to send large amounts of data unrelated to SIP's operation. It is not meant as a replacement for HTTP. [...] SIP is not meant to be a general Remote Procedure Call (RPC) mechanism. None of its user discovery and registration capabilities are needed for RPC, and

neither are most of its proxy functions." (SIP's Solution Space, RFC 4485 section 3.1.). From this it follows that, while SIP may be used in a media network for the device discovery and to open an XPMN session between clients, a different protocol must be used for the direct client-to-client communication. The SDP message in the INVITE may contain an IP address and a TCP port number for it.

Let us take a closer look at the NAT traversal to determine whether SIP provides a benefit here for the XPMN scenarios. Unlike the common SIP use case with media data streamed over UDP/RTP, the media network requires a TCP connection between the clients for message exchange: commands and events must be send reliable between a controller and a service provider.

If the clients are in distinct private networks and no helper protocol such as NAT-PMP, UPnP IGD or Teredo is available, opening a TCP connection will fail. ICE and STUN only cover NAT traversal for UDP-based protocols; TCP connections are not covered. Yet, the basic ideas of ICE still apply: each client has a list of candidates and a TURN server can be used if a direct connection is not possible. *TCP Candidates with Interactive Connectivity Establishment* (ICE-TCP) [PR09] provides information about these additional candidates. ICE-TCP uses the STUN messages for the initial binding and a framing to separate the STUN messages from the rest of the stream (Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport, RFC 4571). This framing ties ICE-TCP to RTP and is no generic solution.

To allow file transfer between clients Garcia-Martin et al. proposed to use MSRP to exchange the data [GMICL06]. The clients open an MSRP session using SIP and MSRP SEND messages to send chunks of data from one client to another. This solution could be adapted to XPMN commands and events. Yet, it does not benefit from ICE-TCP and uses a relay where a direct connection may be possible.

In *Remote Service Usage Through Sip with Multimedia Access as a Use Case* [HGR+07] the authors have built an architecture similar to the media network on top of IMS. In their architecture a *Service Discovery Gateway* (SDG) is used to connect the home network with the outside world. When two SIP clients open a TCP session, the SDG automatically makes a corresponding mapping in the NAT table. This design deliberately adds a man-in-the-middle, a design the SIP VoIP use case eliminated with the development of STUN and ICE since it prevents end-to-end security. In an IMS network most of the infrastructure components are operated by the network operator and are therefore always assumed to be trustworthy—an assumption we cannot make for the media network.

In summary, SIP has some issues opening TCP connections between clients if network address translation is involved. On the other hand, SIP can easily be used to open an RTP session between two peers for audio and video streaming. There are standards for SIP voice mailboxes and a UPnP AVServer is very similar to such a device. The media server requires a re-encoding engine since it is impossible to stream all media container formats and codecs over RTP.

Let us ignore the NAT traversal for now and assume that it will be possible; maybe by making IPv6 a requirement with Teredo as fallback similar to the Windows Peer-to-Peer Network (see

section 4.2.2). The next question to ask is how devices communicate with each other since SIP should not be used for RPCs. One solution is to use SIP to connect home networks and use UPnP or a similar technology between devices. In *Remote Service Usage through SIP with Multimedia Access* [HG07] Haber and Gerdes connect two UPnP home networks to one large extended media network by opening a TCP connection between *Service Discovery Gateways* (SDG)[14]. An SDG forwards UPnP messages to the remote network. The UPnP SOAP messages contain IP addresses for further actions or callbacks; these IP addresses are private IP addresses and unresolvable in the remote network. Media server proxies in both local networks connected to the SDG are required. A UPnP controller in the remote network sees the proxy as a media server and the real media server in the home network is controlled by the proxy. The proxy has to rewrite the URIs in all messages to allow external services to access the media. This is very complicated, not easy to implement correctly, and again prevents end-to-end security.

Roychowdhury and Moyer proposed the usage of SIP instant messaging for a scenario similar to the extended media network [RM01]. The subscribe/notify mechanism is used to subscribe to events on a remote host. In the media network, such an event could be "recording started" from a TV recording device. Each device has its own SIP address and a local SIP server in the home network is used. To send a command or query to a remote device, a new SIP command DO is introduced which is similar to the MESSAGE command for instant messaging. The command has an XML document as payload with the action to call on the remote device. The answer is sent back in the payload of the corresponding OK message.

Besides the fact that the DO command violates RFC 4485 by adding an RPC mechanism to SIP this approach has some design flaws:

- The commands are routed though the SIP proxy servers and each proxy could modify the payload. As a security measure, the complete payload must be encrypted. How this can be done is not described. SIP lacks an end-to-end security layer; even with the body encrypted each proxy can modify the header.
- Each device has its own SIP URI and consequently its own password to register to the SIP network. A user-friendly device setup is required to provide the necessary information to a new device.
- Multimedia playback is limited to RTP due to limitations of the NAT traversal. Other forms of media access (e.g. image files) need a TCP stream between the two peers. Video streaming over HTTP such as used by UPnP requires additional work.

This approach suffers from not looking at the problem from all angles: it describes a basic solution for the networking layer for video content, touches the problems of security, and ignores the usability completely. However, analyzing it points out generic problems SIP has for a scenario such as the media network.

[14]The SDG in the paper by Haber and Gerdes has nothing to do with the SDG defined before. The different authors use the same name and abbreviation for distinct things.

4.3.4 Lessons Learned

SIP is designed to initiate a session and SIMPLE adds presence information. Everything else is outside the scope of SIP; SIP should not be used for RPCs. Using SIP for session setup is a possible solution, but it carries a huge ballast. Reusing SIP would cause more problems than providing benefits.

The mentioned research activities reveal some important findings:

1. The concept of connection candidates in ICE can be used for NAT traversal. Together with NAT-PMP, UPnP IGD and Teredo a direct connection may be possible. Making IPv6 addresses a requirement allows a direct connection for all NAT behaviors except if both clients are behind a symmetric NAT. In that case, a TURN server is a possible solution.
2. Routing the commands and events over the servers may be a good solution to avoid creating a TCP connection between each pair of clients for simple messages. While SIP does not allow such a behavior, a different protocol might; for instance, SIP could be used to open MSRP sessions between the peers.
3. We need a way to deal with multiple devices and more than one account on a server. If all XPMN devices share the same SIP URI, they cannot be addressed individually; if they have multiple URIs, they all need an account with a password on the SIP server.

SIP is a suitable protocol to open a session between two peers, but a different protocol must be used between these two for the XPMN profiles; instant messaging seems to be an interesting way of solving the problem. All events and even the request/response mechanism can be reduced to sending instant messages between peers.

SIP started out as a protocol to open a multimedia session between peers and instant messaging support was added later. Maybe a different protocol, designed for instant messaging is more suitable as XPMN core protocol.

4.4 XMPP-based Approach

The *Jabber* project was founded in 1998 by Jeremie Miller with the goal to create an open platform to replace existing, centralized, commercial instant messaging services such as ICQ, the AOL Instant Messenger, and the MSN Messenger. “Jabber was designed from the get-go for peer conversations, both P-P (between people) and particularly A-A (between applications), and for real-time as well as asynchronous/offline conversations.” (Jeremie Miller in Conversational Technologies [Mil01]). In 2001, the project released its main product jabberd, an open source instant messaging server. The developed communication protocol formed the basis for the *Extensible Messaging and Presence Protocol* (XMPP, RFC 3920) [SA04b]. Most end users are not familiar with the name XMPP and know the protocol as Jabber or Google Talk.

4.4.1 XMPP Core

XMPP uses a client/server architecture and the clients connect to their servers and communicate with other clients indirectly using the servers as relay. RFC 3920 defines the basic communication between entities independent of the instant messaging use case. It supports *Transport Layer Security* (TLS) between a client and a server and between two servers. A client authenticates to its server using the *Simple Authentication and Security Layer* (SASL).

XMPP is an XML stream-based communication protocol with the XML stream being the container for XML stanzas, the first level children in that stream. When the client connects to its server, it opens an XML stream by sending an opening `<stream>` start-tag. The server responds by opening a stream back to the client using the same TCP connection. Inside these two unidirectional streams the XML stanzas are sent for client/server communication. A stanza contains 'from' and 'to' attributes and is routed by the servers to the destination. An XMPP server is responsible for a specific domain similar to an e-mail server. If the stanza destination is not handled by the sender's server, it opens a stream to the destination server which then delivers the stanza to the destination. These server-to-server streams (S2S) are kept open for subsequent communication.

The XMPP core defines three main stanza types:

1. The <message> stanza is used to send a message to another client.
2. The <iq> stanza (Info/Query, IQ) is a request/response mechanism for a client to request something from a remote client or the server. The remote node answers with an IQ result or an IQ error. The request will be handled by the client's server if the 'to' attribute is omitted.
3. The <presence> stanza is a very simple publish/subscribe mechanism for exchanging presence information. Users can subscribe to each other's presence information, allowing friends to receive the presence status of each other. The client sends a new presence stanza to its server when its presence status changes and the server will forward this information to all subscribed clients of the user and friends.

These three core stanza types are only the containers to provide a simple delivery mechanism for near real-time communications. The content of a stanza may be qualified by any XML namespace. XMPP can be used to exchange any data that can be represented in XML; binary data is supported by using Base64 data encoding.

The JID (Jabber Identifier)[15] is similar to a SIP URI or an e-mail address: it contains a user name and a domain part (user@domain). This form of a JID is called a *bare JID*. A user can log in with several clients at the same time and all these clients share the same bare JID. An optional third part in the identifier specifies the resource (client) of the user (user@domain/resource). A JID with a resource identifier is called a *full JID*. This concept makes it possible to send a stanza to any client belonging to a user by using the bare JID and addressing a specific client

[15]The name JID is the last remaining reference to Jabber in XMPP. It was kept for compatibility reasons.

```
<message from='juliet@capulet.com/balcony'
         to='romeo@montague.net'>
  <body>Art thou not Romeo, and a Montague?</body>
</message>

<message from='romeo@montague.net/orchard'
         to='juliet@capulet.com/balcony'>
  <body>Neither, fair saint, if either thee dislike.</body>
</message>
```

Example 4.1: XMPP Instant Messaging

with the full JID. Example 4.1 shows a simple conversation between Juliet and Romeo.[16] The first message is sent from Juliet's client (full JID) to any of Romeo's clients (bare JID). The answer from Romeo is sent from a full JID to the full JID of Juliet's last client. Yet, if Juliet would have switched from the balcony to her chamber she would not have missed Romeo's message. A message stanza addressed to a nonexistent full JID is handled as if it is addressed to the bare JID. This does not apply to IQ and presence stanzas.

4.4.2 Extensions

On top of RFC 3920, RFC 3921 (Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence) [SA04c] defines instant messaging and contact list (roster) handling using XMPP. Besides the RFCs, there are several extensions defining additional functionality. These extensions are defined as *XMPP Extension Protocol* (XEP)[17]. A XEP is developed by the *XMPP Standards Foundation*, an independent, non-profit standards development organization. "The mission of the XMPP Standards Foundation (XSF) is to build an open, standardized, secure, feature-rich, widely-deployed, decentralized infrastructure for real-time communication and collaboration over the Internet."[18] The list of extensions covers a wide range of use cases including file transfer, user mood indication, electronic business cards support (vCard), and XMPP over HTTP to bypass limitations of mobile phones and web browsers.[19] The standardization process of a XEP is a mixture between an IETF Internet Draft and an RFC. A XEP starts as *Experimental* and expires like an Internet Draft after six months of inactivity to *Retracted* or *Deferred*. In *Draft* status, the XEP cannot expire anymore and developers are encouraged to implement it. A *Final* XEP is not changed anymore, but can be assigned a status of *Deprecated* or *Obsolete* (similar to an RFC becoming historic or obsoleted by a new version). For a XEP to get assigned Final status it must have at least two independent implementations with at least one being open source. In practice, there are much more implementations. The complete procedure is described in *XEP-0001: XMPP Extension Protocols* [SA08b].

[16] XMPP standards and extensions often use characters from Shakespeare instead of the usual characters Alice and Bob from example.com. This practice is adopted in this document.

[17] Some older documents refer to the extensions as JEP (Jabber Extension Protocol). The term JEP is deprecated.

[18] XSF Mission Statement; `http://xmpp.org/xsf/mission.shtml`

[19] The list of official extensions is located at `http://xmpp.org/extensions/`.

Describing all extensions is beyond the scope of this document; most of them are very specific for instant messaging and not useful for a media network. This section discusses extensions for end-to-end security, media stream setup, and application layer multicast.

ESessions

There is a collection of extensions defining encrypted end-to-end sessions (ESessions) between two clients. XEP-0200 (Stanza Encryption) [Pat07] specifies how to encrypt a message stanza. ESessions create a *Short Authentication String* (SAS) during session initiation for peer authentication. The algorithm generates a five-character SAS and the two communication partners need to compare whether their clients calculated the same SAS to ensure the end-to-end characteristics of the session. This comparison must take place using a different communication channel such as phone, e-mail, or a meeting in person. The encryption only works for message stanzas and does not protect IQ stanzas for request/response operations. Yet, it might be possible to add IQ support to ESessions.

Only one XMPP client is known to implement ESessions and in May 2008 the status of the XEP was set to Deferred because is was not updated for half a year. Many users of the one client implementing ESessions do not validate the SAS;[20] inexperienced users might even exchange the SAS over the same XMPP channel the ESession is supposed to secure. But the main reason why ESessions were abandoned by the XMPP community is that the protocol itself is not analyzed by security experts. There are many examples in the history of encryption where the authors missed a weak spot in the design, for instance, WEP for wireless LAN.

As of May 2008, S/MIME (End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol, RFC 3923) [SA04a] is the only end-to-end security protocol for clients and there are no XMPP clients known to support it. When the XMPP community contacted security experts in the IETF, they suggested using a well-known technology such as TLS.

Jingle

XMPP is designed to send messages between nodes inside XML streams. While this is a good solution for instant messaging, it does not work for audio and video communication. Members of the XMPP community were interested in such a way of multimedia communication and in 2005, a basic signaling protocol on top of XMPP was developed [SA07]. When a first draft was published, Google employees contacted the authors because the Google team also developed a basic signaling protocol for Google Talk, their XMPP-based "Instant Messaging and Voice over IP" product. The teams merged their specification and the XMPP Standards Foundation published a first version at the end of 2005.

[20] This information was provided by one of the developers during a discussion about end-to-end security on the XMPP security mailing list; `http://mail.jabber.org/pipermail/security/2008-August/000365.html`

Jingle (XEP-0166) [LBSA+09] defines a generic mechanism for session initiation, negotiation and management similar to the basic SIP functionality. An extension using Jingle is called a *Jingle Application*. Figure 4.2 depicts a generic session from initiation to termination.

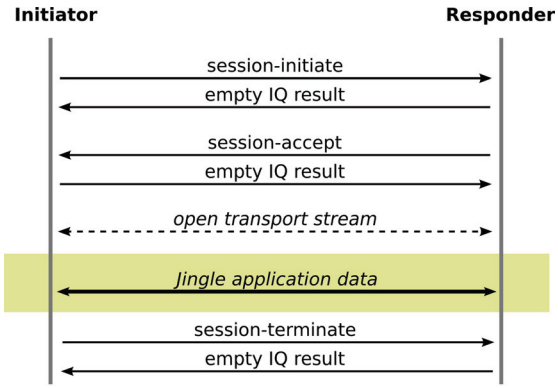

Figure 4.2: Jingle Session Flow

The first extension using Jingle was Jingle RTP Sessions (XEP-0167) [LSAE+09] for Voice over IP. Later *Jingle File Transfer* (XEP-0234) [SA09a] was developed. Unlike VoIP which is based on UDP/RTP, this extension requires characteristics of a TCP stream such as reliability and ordering. The extension does not care how this stream is created inside Jingle and it simply requests a reliable stream. The Jingle layer tries multiple existing XMPP extensions for client-to-client streams: *SOCKS5 Bytestreams* (XEP-0065) [SMSA07] uses the syntax of SOCKS to set up a TCP stream between two clients and *In-Band Bytestreams* (IBB, XEP-0047) [KSA09] send binary data Base64 encoded in message or IQ stanzas to the peer.[21] IBB is an always-working fallback for exchanging small pieces of data, but is unusable for larger files. Most servers limit the bandwidth a client is allowed to use to avoid users abusing the service. Sending media data over IBB will definitively trigger the throttling mechanism.

Publish-Subscribe

With growing deployment of XMPP, the need for a generic publish/subscribe mechanism arose from several use cases. Users wanted to extend the presence information, include RSS/Atom feed support, share bookmarks, and many other scenarios requiring some sort of event mechanism.

The basic idea is for a client to subscribe to a node and when another client publishes information to that node, this information is sent to all subscribers. *XEP-0060: Publish-Subscribe* (pubsub) [MSAM09] defines such a service and the corresponding pubsub server. A client can create a node with items on the pubsub server. The XEP uses a blog as example: the blog itself is the node and the owner of this blog can publish new entries. Other clients can subscribe to that node—the blog—and receive updates once the owner publishes a new entry.

[21] This text describes the usage of the Jingle transports as of May 2008. During the development of the XPMN architecture, all reliable Jingle transport methods have changed. See sections 6.1.4 and 7.1.1 for details.

A pubsub node has several configuration options for various use cases. It is possible to allow friends in the roster to publish items, limit read or write access to a group of friends, restrict the access to the owner, or allow the owner to moderate.

Today, most XMPP servers support XEP-0060 and provide this service to their users. Furthermore, a pubsub server may be a regular XMPP client providing this service to others.

4.4.3 "Jabber going Social"

The success of the so-called Web 2.0 with its social structure and user generated content gave the idea to include such concepts in XMPP. With pubsub as base technology, users can publish information and web services could be accessed via an XMPP gateway. The integration of web services into XMPP is called "Jabber going Social" in the XMPP community.[22]

Personal Eventing

When the first pubsub draft was published in 2002, the XMPP community agreed that it is a powerful mechanism that should be incorporated in many other XEPs. But it took three more years until pubsub got its break through with *Personal Eventing over Pubsub* (PEP).

The publish/subscribe mechanism described in XEP-0060 is very powerful and admittedly, very complex. Often users want to share some information based on a specific use case; for instance, a user listing to music in the background. A client needs the JID of a pubsub server to publish something. On the other end, the subscribers need to know which server the publisher uses; they need to know where to register. Besides the JID of the pubsub server, the publisher and the subscribers need to agree on a node name on that server. Moreover, the clients need to know that the desired information (e.g. the music playing in the background) is published all.

The Personal Eventing Protocol (PEP, XEP-0163, previously know as Personal Eventing over Pubsub) [SAS09] introduced auto-creation of nodes and auto-subscription based on the service discovery from XEP-0030. PEP defines a virtual pubsub server based on the bare JID for each user, and the namespace of an extension is the node name for events belonging to that service.[23] For instance, `http://jabber.org/protocol/tune` defines a namespace for music and a client uses the namespace as node name to publish which music its user is currently listening to. A friend's client in the contact list is automatically subscribed to the PEP service if it supports the `http://jabber.org/protocol/tune+notify` feature.

PEP made the usage of pubsub much easier for both client developers and users. With auto-subscription, a client does not need to know what friends publish events for specific services

[22] See the mailing list archive at `http://mail.jabber.org/pipermail/social/` for discussions in this area.

[23] Similar to Jingle, the pubsub and PEP specifications evolved during the time this thesis was written. This text describes PEP as of May 2008. During the development of this thesis most parts of PEP were included in the pubsub specification, making PEP a profile on top of pubsub.

and where to subscribe to. The music example used before is specified in *XEP-0118: User Tune*; other extensions based on PEP are *XEP-0107: User Mood*, *XEP-0108: User Activity*, and *XEP-0197: User Viewing*. These extensions are very small and mainly define the XML schema of the information to publish.

XMPP and Web 2.0

RSS/Atom feeds from web services range from site-wide news feeds such as blogs to a personal photo stream such as those offered by Flickr. Unlike XMPP, an HTTP server cannot send events to a client; to receive updates, the client must frequently poll the server and check the feed for changes. While a server can handle many parallel HTTP requests in the operating system and the web server, the server may require resources to create the feed or to perform database look-ups.

Elliott-McCrea, employed at Flickr, provided some insights about the traffic the HTTP-based API produces and how an XMPP solution could change the situation. He only takes the traffic from friendfeed into consideration; friendfeed is a site that aggregates RSS/Atom feeds and uses APIs from multiple web services. "On July 21st, 2008, friendfeed crawled Flickr 2.9 million times to get the latest photos of 45,754 users of which 6,721 of that 45,754 visited Flickr in that 24 hour period, and could have **potentially** uploaded a photo.[...] But it's worse. If any of those 6000 people uploaded **lots** of photos, friendfeed didn't see that either, because our buffer size on RSS is 20 items; anything more is lost." [HPEM08, accentuation adopted from source] 2.9 million accesses per day with 20 items per stream and 2 kbyte per photo metadata sums up to around 100 Mbit/s. This example only covers the traffic from friendfeed, there are many more. For instance, Google Reader is a popular web-based RSS reader and the Google servers poll external RSS feeds such as Flickr photo streams. The fact that a client has to poll the server to receive updates is the main problem.

XMPP using publish/subscribe can reduce the traffic enormously. The XMPP network is built as a federated network with asynchronous communication that already has identity and presence built in—pubsub was created for such a use case. In the same time frame mentioned above, Flickr had 60 photo updates per second as peak and an XMPP entry for each photo would be around 2 kbyte without stream compression. That results in less than 1 Mbit/s if someone is subscribed to all Flickr uploads. A site such as friendfeed could aggregate all photo streams over a DSL connection.

The latest development in the XMPP community around the publish/subscribe service created pubsub chaining to reduce the number of subscribers to one pubsub server. Users may subscribe to a server mirroring popular pubsub nodes instead of subscribing to Flickr directly. This equates a service such as friendfeed adjusted to XMPP.

The users will still use the current HTTP-based REST interface to upload their images to Flickr and use the XMPP network to get notifications about changes. XMPP uses persistent connections to provide a near real-time notification. Unlike the HTTP-based polling, the users' clients

have an open connection to their XMPP servers and only the servers have a direct connection to Flickr. This reduces the load on the Flickr server and also enhances the service for the users by providing near real-time notification.

Another example for a web service that would benefit from XMPP is Twitter. Twitter introduced the concept of *micro-blogging*. The concept is similar to blogs, but the number of characters is limited to 140 characters in length; similar to an SMS. Unlike blogs where a user adds long post in an interval of days, Twitter users add several new comments per day. The user base of Twitter rapidly grew and the service could not handle the load anymore. An XMPP-based API was one openly discussed solution. In the end, XMPP was not added in favor of updating the servers and rewriting some parts of the code. It may only be a matter of time before the service struggles with the same problems again. Another micro-blogging site is identi.ca which is based on a federated network of servers, and uses XMPP in its core and for user notification. A web front-end is also available.

4.4.4 Applying XMPP to the Problem

There is a product called LobsterTunes that seems to use XMPP to remotely access audio files from UPnP AVServer in the home network.[24] LobsterTunes uses the Lobster XMPP server to control a client in the home network to browse the media collection. The audio files are then streamed directly to a mobile client. Details about LobsterTunes works internally are not publicly available.

Aurell prototyped the idea of controlling devices using XMPP [Aur05]. “Instant messaging has good scalability properties, in that it is independent of location or geographical distance, and it is easy to scale up and down on server level to increase or decrease capacity”. He created XMPP gateway-clients to simple X10 devices used for home automation. His solution allows to turn on the heater (commands) or check whether a door is open (query) using IQ stanzas. He points out that the lack of an end-to-end security layer is the central open issue in his prototype.

Choi and Yoo use a similar technique to create a “human-thing” communication based on XMPP [CY08]. Devices such as a printer or locations such as a conference room are part of the user’s contact list and the user can communicate with them. The XMPP printer-client connects the user to the device driver. The user can send a message to the printer to print out a file and the printer can communicate with the user that the print job is finished or that it is out of paper. A conference room can be reserved or queried whether a laptop or a video projector is available in the room.

These two prototypes show the potential of instant messaging and XMPP in particular to control devices. The personal media network is just a network of such devices. Some are controlled by a user (human-to-device) and some are autonomous (device-to-device). XMPP provides the basic technology needed for the XPMN core:

[24] The white-paper can be found at `http://www.lobstertunes.com/Lobster-Whitepaper.pdf`.

- Presence and contact lists can be used to manage devices and friends.
- XPMN profiles can be defined in XML namespaces and elements of these namespaces can be added to messages and IQ stanzas for notification (eventing), controlling and querying.
- Jingle can be used to establish a media stream between two clients for profiles such as a media server or a TV tuner device.

The major drawback is the lack of end-to-end security. The SAS used in ESessions is unsuitable for XPMN devices because it is very user-centric: XEP-0200 requires the user on both clients to verify the SAS; some devices such as DVB recorders may not have a user interface to display the SAS. More importantly, it has scalability issues. An ESession is between two clients only, each device pair in the media network has their own two strings for the user to verify. To create a media network with ten devices, the user has to verify 45 pairs of authentication strings—this is unacceptable.

4.4.5 Lessons Learned

XMPP fulfills most of the requirements for the XPMN core with RFC 3920, RFC 3921, and some XMPP extension protocols. XEP-0030 and XEP-0115 provide a good device and service discovery, the contact list provides a list of friends, XEP-0174 a way for link-local device discovery based on Zeroconf technology, and the extensibility of XMPP makes it possible to define the missing requirements and XPMN profiles.

It is a very lightweight protocol for a client and puts the burden of device discovery on the XMPP servers. The distinction between bare JID and full JID opens a new choice how to assign devices to an extended personal media network. An XMPP-based media network may use the bare JID to relate to a specific personal media network; each user has a distinct bare JID. The various devices in one personal media network all share the same bare JID and use a distinct resource names, thus having a unique full JID. The resource part of a full JID can solve the entity naming problem outlined in section 4.3.4.

Some XMPP servers restrict the number of clients each user is allowed to use in parallel. This restriction is based on the server policy and must be adjusted to the XPMN use case. Rémond explained that a default value of ten in ejabberd was added because of bad client behavior [Rém09]. He encountered a badly programmed client that registered over thousand resources for one bare JID causing an enormous presence flow in the server. During a private conversation, he suggested a default value of fifty for the XPMN scenario.

The presence mechanism in XMPP automatically provides device discovery for each media network. Once a client sends its presence information to its server, clients sharing the bare JID will automatically receive this information and the client receives their presence status. The XMPP contact list from RFC 3921 enhances this functionality to friends and their devices. A new device of a friend shares the bare JID of the already known clients and therefore will automatically be known, too. This makes device management easy for a client.

Security is the central issue when using XMPP as core for the media network: the communication between client and server is secured using TLS, but there is no end-to-end security between clients. Similar to Orb, the user has to fully trust the server. Most likely, Lobster does not add end-to-end security in their closed network as well. Furthermore, when using serverless communication between two clients in the home network, the clients have no easy way to identify a peer.

This leaves us at the same point the UPnP forum was before the UPnP Security Ceremonies Design Document was published: we want to add end-to-end security to an already existing insecure core protocol. It is important not to repeat the mistakes made by the UPnP forum and the security layer should be as simple as possible for the end-user with a reasonable end-to-end security.

4.5 Summary

This chapter only covers a small subset of available protocols and solutions. There are several other peer-to-peer and instant messaging protocols; with an open specification or a closed system. Nevertheless, the protocols introduced in this chapter are the ones with the widest deployment in their field, solutions developed within the IETF, or often used in research activities. It does not seem wise to build the XPMN core on top of a closed system.

As a last solution, the *Advanced Multimedia System* (AMS, H.325) should briefly be mentioned because it has a similar use case as the media network. In July 2007, the ITU-T formed a new Question, the ITU-T's term for working group, to define a successor to H.323 and SIP. One year later, the Question to study AMS was formally approved [Jon07].[25] "AMS will truly enable multimedia communication that goes well beyond just voice and video, and address QoS, security, and NAT/FW issues from the outset." They include scenarios similar to the ones described in sections 2.1.1 and 2.1.4. AMS is in a very early stage of development without any details publicly available; a first release is targeted 2011 or 2012.

It looks like XMPP seems to be the most suitable protocol as base for the XPMN core. The extensibility of the protocol and the lightweight client requirements make it a good choice. Though, it does not provide everything needed—most of all, an end-to-end security layer is missing. Some of the other protocols we took a look at use TLS for a secure end-to-end communication. Maybe this practice can be adapted to XMPP. Nevertheless, none of the peer-to-peer protocols using TLS define a user-friendly way to bootstrap the TLS communication: a peer must be able to verify the certificate. There must be either a simple way to distribute certificates among the clients or an easy way to get a certificate signed by a CA. TLS does not provide any additional security without a verifiable certificate.

Another thing missing in XMPP is a direct TCP connection between clients. There is an extension to use SOCKS5 to open TCP streams, but it lacks the flexibility of ICE. Both clients must

[25] The H.325 information site is located at `http://www.packetizer.com/ipmc/h325/`.

be able to present possible candidates and together with protocols such as NAT-PMP, UPnP Internet Gateway Devices (even though it is considered a security risk), and Teredo the chance of getting a direct connection between peers is very high. If the security layer works on top of the stanza exchange, clients can communicate securely based on their connection to the server and a direct connection may not be required for the XPMN core. Nevertheless, a media transport profile will require a direct client-to-client stream.

The examination of web services indicate that an integration should be done one two levels: some services should be integrated directly in the media network by getting an XMPP interface, others should be integrated through a web browser. The latter is outside the scope of this thesis; there are working solutions such as Web4CE that can be used.

As a next step we must check XMPP against all of the initial requirements from section 2.2 and create an XPMN core architecture. On top of that core, profiles can be defined.

Chapter 5

Refining the Requirements

Now that we have chosen XMPP as the underlying protocol for the media network, XMPP and the available extensions need to be checked against the initial requirements from section 2.2 to determine what additional extensions are needed.

Requirements not met can be grouped into two main categories: network, for client-to-client communication, and security. Besides technical aspects, the introduced ideas must be user-friendly according to section 2.2.9. The findings are the basis for the media network's core architecture developed in chapter 6 and the media transport defined in section 7.1.

5.1 Initial Requirements

Looking at the initial requirements from section 2.2, most of them are already fulfilled by XMPP and available extensions. A device can be mapped to an XMPP client and each provided service will be defined by an XMPP extension. Similar to the pubsub extension, the media network profiles have a client (controller) and server (service provider) part. This fulfills requirements 1 and 2. Each device has a unique JID based on the user's bare JID, which is the unique media network identifier, and a resource name, which is a unique identifier inside a personal media network (requirements 8, 11, 22 and 23). The bare JID refers to a personal media network and the full JID is a unique device identifier.

All XMPP clients connect to their XMPP server and the federated network of servers route the XMPP stanzas between clients. This server-based routing can penetrate any NAT and allows two devices to communicate even if they are both inside a different private network behind a NAT (requirements 9 and 10). Inside the home network serverless messaging based on Zeroconf technologies is used (requirement 13).

XMPP presence stanzas together with XEP-0030 (Service Discovery) [HMESA08] and XEP-0115 (Entity Capabilities) [HSATK08] are the foundation for device and service discovery, including service discovery query caching (requirements 16 and 17). The roster contains the

list of external users, and presence subscriptions allow enhanced device discovery to locate devices of friends and external services (requirements 24 and 25).

After refining the requirements we have to specify the media network core and build exemplary profiles on top of it (requirements 5, 6 and 7). Each profile should be defined as XEP with its own XML namespace inside the XMPP Standards Foundation (requirements 3 and 4).

This leaves the following initial requirements open to be solved on the networking layer:

1. XMPP cannot handle disruptions on the IP layer. If the home network changes its public IP address, a client behind the NAT may not detect this and a stanza sent through an invalid TCP connection may be lost (violates requirements 14 and 15).
2. While the XMPP server allows clients to exchange XML stanzas independent of the network topology, a different approach for NAT traversal is required for media data transfer. It is impractical to send a video file Base64 encoded in the XML stream through the XMPP network.

On the security layer end-to-end security and device management has to be designed:

3. XMPP does not provide end-to-end security; right now, the XMPP servers must be trusted as part of the media network (violates requirement 12). Before designing an additional security layer for end-to-end security, we must perform a security analysis to determine what additional security is needed.
4. The end-to-end security layer may require the devices to perform their own device management and authentication independent of the XMPP server. Requirements 18, 19, 20 and 21 are not met by XMPP and any existing extension. The same is true for device handling of friends in the roster (requirement 26).
5. The roster only defines the list of friends and does not include access control lists to restrict the access to service providers (violates requirement 27). It also relies on the XMPP server to authenticate a friend. For security reasons the server should not be trusted with that task—the user has no trust relationship to the servers of friends.

These missing requirements are refined in the following sections. First, the media transport and the disruption tolerance on the networking layer will be analyzed. After that, we take a look at the requirements on the security layer and go into details how to design it by considering usability aspects.

5.2 Networking Layer

XMPP meets most of the requirements for the networking layer. Clients only need to connect to their corresponding XMPP server to communicate with each other. This works for most of today's home networks as a client only needs to penetrate the NAT from the inside. The servers on the other hand must have a public IP address. We assume that they are operated by third

parties and that this is of no concern of the user. An experienced user may set up a private XMPP server; the average user will choose one of the existing servers.

Bidirectional-streams Over Synchronous HTTP (BOSH, XEP-0124) and *XMPP Over BOSH* (XEP-0206) define how to tunnel XMPP streams over HTTP connections. Originally designed for mobile devices that are only capable of opening HTTP connections and JavaScript clients, BOSH can also be used to control the media network from a device behind a more restricted firewall found in large corporations. However, the media network design should respect company policies and not try to punch holes in firewalls. Thus, we assume that there are only three possible problems on the networking layer:

1. A home network or the company Intranet may be in a private network behind a NAT. A client is allowed to connect to any host on the Internet, but it is impossible to connect to the client from the outside. Additional NAT assisting helper protocols such as NAT-PMP or UPnP IGD should be supported, but they may not be available.

2. The home network may have a changing public IP address, for instance, when the ISP enforces a disconnect after a couple of hours and the router automatically reconnects. After the reconnect, open TCP connections to the outside become invalid. A client must be aware of this to reestablish its connections. NAT-PMP broadcasts the external IP address in the LAN and keep-alive messages on the XMPP layer can be used to detect whether a TCP connection is still valid.

3. A client or the whole home network is unreachable between disconnect and reconnect. A disconnect may happen more often for a mobile client when it repeatedly changes between a 3G connection and Wi-Fi, or when it has no reception for a short period of time (e.g. elevators, tunnels). Due to an abrupt disconnect, a sender has no knowledge whether the last sent packets reached the destination or whether they need to be retransmitted after the reconnect.

The first item is already addressed by routing XML stanzas over the server. A direct client-to-client connection is required to exchange media data. This is not part of the XPMN core; nevertheless, profiles such as a media server must have a solution for this to offer *media transport*. The second and third items are both in the area of *disruption tolerant networking*.

5.2.1 Media Transport

It should be possible to exchange media data between clients directly to reduce the load and the bandwidth requirements of a relay. A direct connection is possible when one of the clients involved is not behind a NAT or firewall, or if one can manipulate the NAT to forward incoming connections to it; unlike supported by TCP, that client may be the initiator of the connection. For instance, if a user wants to access media files with a mobile phone from the home server, the mobile phone initiates the media transfer, and the home network is behind a NAT while the phone has a public IP address. Opening a TCP connection from the mobile phone to the media

server is not possible due to a missing mapping in the NAT. The XMPP connection is the logical choice for the negotiation layer to exchange possible candidates to open a TCP connection or to initiate a video or audio transfer.

The two previous chapters introduced multiple techniques to exchange data between two hosts behind different kinds of NATs without routing it over an external server. NAT assistant solutions such as NAT-PMP or the UPnP Internet Gateway Device installed in only one network will make direct data streams possible. Other protocols such as Teredo or ICE require support on both sides and are additional solutions for NAT traversal.

Reliable Media Streams

The TZI analyzed various methods of distributing video streams to a client in the first phase of the ScaleNet project [BKML07b]. Most Internet video solutions use pre-recorded content and the transport depends on the video container format. Sites such as YouTube rely on Flash Video[1]; other sites such as the *Apple Movie Trailer* site use the MPEG 4 container format (MP4)[2]. Some sites give the user the choice whether they want *QuickTime* movies (MP4), *Windows Media Video*, or *Real Video*. The latter uses the *Real-time Transport Protocol* (RTP, RFC 3550) [SCFJ03] for video streaming. RTP is also used for voice and video data in SIP and Video on Demand solutions based on IMS demonstrated by partners in the ScaleNet project.

RTP is intended for unreliable transport and is typically used with UDP. RTP is often used with "live" media streams such as Voice over IP (VoIP) and live TV streaming. The sender splits the media stream into small chunks and adds a time stamp to each packet. Optionally, the stream can be enhanced with forward error correction (FEC) to compensate for lost packets.

RTP is designed for streaming media data with real-time constraints. Reliable retransmission-based protocols are considered unsuitable for real-time transport due to the potential delay caused by retransmission and sequenced delivery. This means that RTP provides real-time instead of reliability. A packet is dropped if it is too late to be played.

Unlike telephone calls playback of pre-recorded files requires more reliability and less real-time communication. An initial delay of one second does not matter; however, a packet loss in the video stream will result in notable artifacts or a wrong picture. The error might be visible several seconds depending on the video encoding parameters. For this reason a reliable transport protocol should be used for audio and video assets in the extended media network. RFC 4571 (Framing Real-time Transport Protocol and RTP Control Protocol Packets over Connection-Oriented Transport) [Laz06] defines a simple framing format to encapsulate the RTP packets into a TCP stream to achieve the desired reliability. Yet, RTP over TCP is not widely deployed.

Adobe developed the *Real Time Messaging Protocol* (RTMP) sometimes used for Flash Video. When this thesis was started, the RTMP specification was not publicly available but several

[1] The Videos are played using a player for Adobe Flash. A Flash browser plug-in is required to play the video. More details can be found at `http://www.adobe.com/devnet/video/`.

[2] The MP4 container format is based on the Apple QuickTime container and specified by the *ISO/IEC Moving Picture Experts Group* (MPEG) as *MPEG-4 Part 14* (ISO/IEC 14496-14).

reverse engineered open source implementations existed. Adobe filed a DMCA removal request against one of these clients[3] and had no license model for third parties. Besides legal concerns, RTMP is no option for the XPMN media transfer because it is designed for the Flash Video container format and supports only a limited set of audio and video codecs. In June 2009, the protocol specifications was made public.[4]

Alternatively, Flash Video and MP4-based solutions can use HTTP as media transport; so does the UPnP AVServer. The underlying TCP layer guarantees a reliable transport. However, not all video containers are suited to be streamed over HTTP, such as when the video and audio streams are not interleaved or the player requires information stored at the end of the file. Independent of the container format, seeking inside the stream is difficult. The player could use the HTTP 1.1 range header for seeking, but most players only support HTTP 1.0. The UPnP AVServer defines RTP as alternative video and audio transport method. Since RTP is optional, UPnP itself is based on HTTP, and HTTP is required for images anyway, HTTP is used in most cases. For RTP streaming the UPnP server has to understand the video container and the video and audio codecs to stream the data over the network; using HTTP, the server can just provide the file as it is. The ScaleNet demonstrator developed in the TZI worked around HTTP streaming issues with various container formats by converting all media files and storing them in a streamable container format—this is no option for a widely employable solution.

The main issue is how to set up the TCP connection between the two clients. Whether the clients use HTTP, RTP or something different over that TCP connection is a secondary issue. Jingle as defined by XEP-0166 was designed for XMPP voice calls based on UDP/RTP but is a generic solution that can negotiate transports with TCP characteristics, too.

Jingle Transports

XMPP has two reliable transports that can be used in Jingle: *In-Band Bytestreams* (IBB, XEP-0047) [KSA09] and *SOCKS5 Bytestreams* (XEP-0065) [SMSA07]. Their integration into Jingle is not as smooth as the ICE-UDP and RTP; they do not take full advantage of the transport element inside a Jingle session description. The reason for this is simple: they are older than the Jingle specification.

XEP-0047 tunnels a byte stream through the XML stream over the server. The data is Base64 encoded and is sent inside IQ stanzas to the recipient. This solution always works: if two clients can communicate, they can exchange IBB stanzas. The drawback is that it does not work for larger media files: most servers restrict the bandwidth a client is allowed to use—sending a Base64 encoded video file surely requires more bandwidth than a typical server permits. For instance, the default configuration of ejabberd, a widely used XMPP server, restricts the traffic to 1 kbit/s for each client.

[3] See `http://www.chillingeffects.org/anticircumvention/notice.cgi?NoticeID=25159`.

[4] The RTMP specification and the license to deploy the protocol are available at the *Adobe Developer Connection*; `http://www.adobe.com/devnet/rtmp/`

SOCKS5 Bytestreams is the use of SOCKS5 (RFC 1928) adapted to XMPP. The initiator of the stream sends a list of *streamhosts* to the responder. The initiator can be a streamhost itself and external relay hosts are used in case the initiator is behind a NAT. The responder tries to connect to the streamhosts using the SOCKS5 connect-method. This does not take advantage of the whole range of possible ways to open a stream. It is only possible for the recipient to connect to the initiator and not the other way around.

XMPP lacks a Jingle transport extension that takes full advantage of all ways to connect without a relay server:

1. If either initiator or responder is not behind a NAT or firewall a connection is possible. Both clients should try to connect to their peer. IPv6 should be tried as well even with IPv6 not yet widely deployed in consumer's home networks.
2. NAT assistant technologies such as NAT-PMP or UPnP IGD should be used if they are available. This makes direct connections possible similar to the first case. If possible, a client should reuse one port mapping for multiple communications. This avoids creating several port forwarding rules that will never be used.
3. Teredo provides public IPv6 addresses and uses a technique similar to ICE-UDP for NAT traversal. The traffic is tunneled over UDP. The Microsoft Peer-to-Peer Network fully relies on IPv6 and Teredo and works just fine without any additional NAT assistant protocols. A relay is used for communication between a Teredo-host and a host on the "native" IPv6 Internet. The Jingle negotiation should include the information whether or not the IPv6 address is provided by Teredo.
4. Since it is easier to penetrate a NAT with UDP as transport protocol, a TCP stream could be set up on top of UDP. There are several different specifications to do this. The Internet Draft *UDP-Encapsulated Transport Protocols* [DC08] describes how to merge the TCP and the UDP headers. As a disadvantage, this solution requires either a TCP stack in user space or a modification to the operating system's kernel.
5. Similar to STUN, TCP connections may be able to penetrate the NAT. If the NAT mapping is only based on the internal IP address and port, and is independent of the peer's IP address, one client can punch a hole in its NAT which the other one can use to open a TCP connection [Epp05].

ICE-TCP as introduced in section 4.3.3 is a suitable protocol to negotiate candidates. Unfortunately, ICE-TCP is not finished yet and does not include NAT-PMP, UPnP IGD, Teredo, or TCP over UDP. Making ICE-TCP an extensible framework may overcome this limitation [RL08]. While ICE-UDP is implemented in some SIP stacks, ICE-TCP is not widely deployed, yet.

5.2.2 Disruption Tolerance

Disruption tolerance is an important aspect of the media network. The dynamic IP address of a client or the whole home network may change during the communication. Messages sent to

the clients may get lost. Until an XMPP client behind a NAT becomes aware of the fact that the TCP stream to the server is invalid due to an IP address change of the home network, it will not reconnect to the server. Most likely it will detect this when it tries to send something over the stream. Many XMPP clients implement the so called "whitespace ping". They send a whitespace on inactivity to the server to keep the connection alive and probe whether it is still valid. A whitespace between XML stanzas is ignored by all XML parsers. Alternatively, XEP-0199 defines a ping IQ stanza that can be used to query the connection between two entities; even end-to-end connection probing is possible. A better way to detect IP address changes is the use of NAT-PMP. NAT-PMP broadcasts the public IP address in the LAN when it changes. NAT-PMP suggests that the router should send ten address change notifications in an increasing interval to gain reliability. A client can instantly reconnect.

The server does not get notified if the home network got a new IP address. Until a TCP timeout occurs or the client reconnects, the server treats the TCP connection as valid. A message sent over such a connection is lost. XEP-0198 (Stream Management) [KHSAF09] adds reliability on XMPP stanza level. An XMPP entity counts the number of sent stanzas and the recipient sends an acknowledgment after a negotiated number of stanzas. On reconnect, a client can resume the previous stream and the missing stanzas will be sent again. During the time this thesis was written, the authors of XEP-0198 made many changes to the specification and the XMPP council even advanced it to Draft status, making it ready for wide deployment.

If the connection of a mobile device changes often, it may be useful to tunnel the XMPP connection through a protocol designed to handle disruptions. The TZI developed the *Persistent Connection Management Protocol* (PCMP) [OK04] in the Drive-thru Internet project.[5] "The idea of Drive-thru Internet is to provide hotspots along the road—within a city, on a highway, or even on high-speed freeways such as autobahns. They need to be placed in a way that a vehicle driving by will obtain WLAN access for some (relatively short) period of time" (Drive-thru Internet web page). PCMP allows the usage of existing Internet technologies by hiding IP address changes and disruptions from both the mobile application and the peer on the Internet.

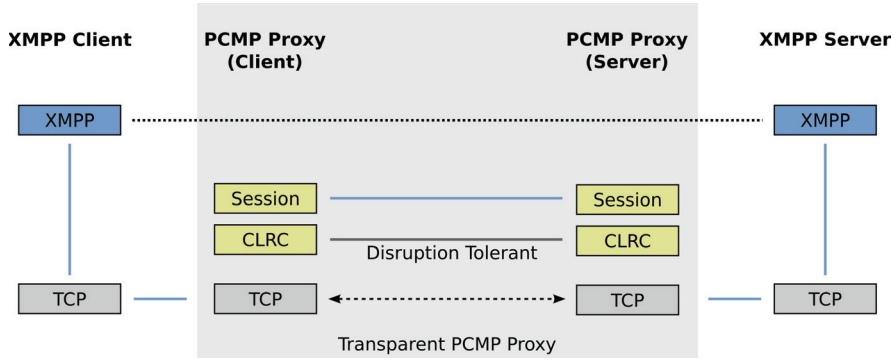

Figure 5.1: XMPP over PCMP

[5]The PCMP specification and papers submitted in this area can be found on the Drive-thru Internet website at `http://www.drive-thru-internet.org/`.

The end-to-end characteristic of a TCP stream is replaced with one TCP connection between client application and PCMP proxy and one between the second part of the PCMP proxy and the original destination. The PCMP proxies tunnel the TCP connections (sessions) over a *Connectivity-loss Resilient Connection* (CLRC). The PCMP proxy takes care of setting up the CLRC after a disruption or a network address change. In the case of XMPP, neither the XMPP client nor the server is aware of IP address changes of the client or that the client may be unavailable for a short period of time. Figure 5.1 depicts a TCP connection between an XMPP client and the XMPP server tunneled through a transparent PCMP proxy. A test with an unmodified XMPP client and server showed that PCMP is a suitable solution for short disruptions or changes on the networking layer. Yet, if the client sends a white space or uses XEP-0199, it may detect that it is offline because its peer does not answer in a specific time frame.

A different solution is being developed by the *Delay-Tolerant Networking Research Group* (DTNRG) in the *Internet Research Task Force* (IRTF). The DTN architecture is designed to handle occasionally-connected devices and changing routes to the client. Instead of tunneling TCP streams or UDP datagrams, the application data is encapsulated in DTN bundles and routed through the DTN network.

The CHIANTI project[6] uses a slightly modified version of PCMP as external module for its *FlexProxy* as one possible method to handle disruptions [OSC+09]. The FlexProxy intercepts TCP connections or UDP flows from the application on the mobile client and tunnels them to the server part of the proxy. Besides PCMP, the FlexProxy can use DTN for tunneling the data. Either way, the tunnel mechanism is transparent for the application and its peer on the Internet.

5.2.3 Summary

The XPMN core requires no additional XEP at the networking layer; everything is covered by the core protocol and the existing extensions. Optionally, XEP-0198 can provide disruption tolerance. A more disruption tolerant transport protocol such as DTN or transparent PCMP proxies may be a reasonable enhancement, but it is not required for the media network to function.

Ironically, the practice to detect connection problems by sending white spaces or XEP-0199 ping stanzas conflicts with transparent solutions such as PCMP. Both the XMPP and the PCMP layer are trying to provide disruption tolerance. The transparent approach makes it hard for the application to detect whether or not it should probe the connection. A different transport protocol may be a better solution for the future; XMPP over DTN could provide the required disruption tolerance on transport level.

Nevertheless, the media transfer as profile on top the XPMN core has additional requirements on the networking layer:

1. An extension to enable support of direct TCP streams in Jingle is needed. Existing solutions such as NAT-PMP, UPnP IGD and Teredo should be integrated.

[6] Detailed project information and the publicly available deliverables of the EU project CHIANTI can be found at `http://www.chianti-ict.org/`.

2. The data should be routed over a relay server if a direct TCP stream is not possible. Possible candidates for a relay server are the existing XMPP SOCKS5 proxies as well as TURN-TCP servers.

3. Similar to ICE-TCP, a protocol trying multiple candidates is required. Jingle already defines a transport negotiation logic that can be enhanced. Furthermore, ICE-UDP is already supported by Jingle and provides a working example for a technology similar to ICE-TCP.

After the XPMN core is specified, we will improve the Jingle transport methods in section 7.1.1. It is only required for some profiles on top of the core and therefore postponed until after the core is defined.

5.3 Security Analysis

XMPP uses *Transport Layer Security* (TLS) and the *Simple Authentication and Security Layer* (SASL) to secure the communication between client and server. The usage of these security protocols is optional; many XMPP servers allow clients to log in without using TLS but require at least SASL for authentication. Server to server communication may be secured using TLS as well but also works without, using other techniques for peer authentication such as *Server Dialback* (XEP-0220)

If two clients from different domains communicate with each other, both clients are connected to their servers. A client only knows whether its link to the server is secure and it has no way of knowing whether the peer uses TLS or the servers authenticated each other and encrypt the traffic. An attacker on the server or on a path with an insecure link can read any message sent between the clients (passive attack) and may even modify or inject messages (active attack). For link-local operation TLS is the only available authentication mechanism; however, it is unclear how a client can verify its peer's certificate. In short, XMPP does not provide an end-to-end security mechanism.

In this section we take a look at security concerns and possible attacks to find detailed requirements for an end-to-end security layer. We need to determine what the security layer needs to provide and what security aspects are optional. Usability aspects must be kept in mind; the best security layer is useless if it is unusable. The solution has to be a compromise between high security and usability of the system.

5.3.1 Security Objectives

Different experts outline various goals for a security analysis. The *Guidelines for Writing RFC Text on Security Considerations* (RFC 3552) [RK03] list three major categories: confidentiality, data integrity, and peer entity authentication. A fourth possible category is the availability of

a service that can be compromised by denial of service attacks. In *Practical Principles for Computer Security*, Lampson compares IT security to the real world: "Real world security is not about perfect defenses against determined attackers. Instead, it's about value, locks, and punishment. The bad guys balance the value of what they gain against the risk of punishment, which is the cost of punishment times the probability of getting punished" [Lam06]. With computers connected to the Internet, the risks of an attacker getting caught are very small; value and strength of the security layer (locks) are important. The values in the media network are the media assets: the personal files, scheduled recordings, and information devices share. The devices themselves are meta-objects. It might not be necessary to hide their existences and capabilities unless it undermines the security of the media assets.

When security is brought up, some people claim they have "nothing to hide" and ask "Who should be interested in my media files?". A good counter-argument is the PGPCoder trojan: it encrypts files based on their extension and the user has to pay $100-$200 to get the decryption tool. If all photos of a new born child or all wedding photos are lost, people may be willing to pay to get them back. Even if they have nothing to hide, the average user is a worthwhile target for an attack and it highlights the need for security.

Confidentiality

Confidentiality is what many people have in mind when they think about security. An attacker should not be able to read the data or decrypt encrypted data. The privacy of the user should be respected and this affects the media data itself. Media files stored on XPMN devices should only be accessible for the user and friends. Even the information what media files the user possesses should not be revealed without permission. People have something to hide even if they do not think so: a compromising party photo being drunk or erotic photos or videos couples made for themselves require confidentiality. A different scenario is copying media files from a user that do not seem to be personal, for instance, audio files from online stores such as iTunes. Some of these stores add a watermark to all the files. If these files appear in file sharing networks, the music label might sue the user for distributing the files. Even if the user could prove that an attacker is responsible, it could be a long and stressful process.

Ideally, the server does not run many services besides the XMPP server application to minimize the possible attack points. An XMPP server is an interesting target if many media networks use the same server. By getting control over one server, the attacker has a node in the media networks of all its users and may even get access to devices of friends. No server administrator will guarantee that the server has no vulnerabilities. "I don't think that jabber.org is a trusted third party, and I'm in charge of jabber.org." (Peter Saint-Andre in a discussion on the XMPP security mailing list about OpenPGP and trusted third parties).

Similar services such as ORB have a special paragraph in the policies the user has to agree to: "Orb may employ procedural and technological measures [...] to protect your personally identifiable information from unauthorized access. However, Orb cannot guarantee that unauthorized

third parties will never be able to defeat these measures or use your personally identifiable information for improper purposes. Therefore, you should consider any communication that you transmit to Orb (such as data [...]) as non-confidential" (Orb Privacy Policy, Section Data Security[7]). Ironically, the user has no idea what the Orb software is doing (see section 4.1.1 for details about ORB), whether the company actually employs procedural and technological measures, and if it does, how effective they are.

From this it follows that the core must provide an end-to-end security layer that ensures confidentiality for messages sent between clients independently of the security measures installed on the server. Media streaming is not part of the XPMN core protocols and covered by profiles. The secure message communication can be used to exchange encryption parameters for secure data streams. This ensures confidentiality for the media assets.

Besides profile stanzas and the media assets, clients exchange presence information for device and service discovery. This information is available to all clients of the user, the clients of all friends, and the XMPP servers involved. An attacker can get inside knowledge about the media network itself based on that information, for instance, to detect a vulnerable client in the media network such as a specific device with a known implementation error.

Admittedly, confidentiality and service discovery are difficult to achieve at the same time; they are more or less mutually exclusive. Most problematic is link-local serverless messaging where the presence information should only be broadcasted in trustworthy networks—only the user can decide this and every user interaction adds additional usability concerns. To make it even more complicated, the wireless LAN *Service Set identifier* (SSID) is only an indicator and an attacker can easily set up a network with the same SSID. Using WPA for encryption makes it more secure by using the pair SSID/password as unique identifier. Unfortunately, it is not always possible for an application to get the required information from the operating system's wireless network manager. This leads to the conclusion that confidentiality for the discovery mechanism is a desirable goal, but does not justify making the system too complex.

Data Integrity

Unlike confidentiality, an attack on the data integrity is an active attack: an attacker may delete, modify, insert or replay a stanza sent between clients. The requirements for data integrity are similar to the ones defined for confidentiality: the media data itself must be protected, for presence information the security of XMPP is adequate.

If an attacker has the potential to modify or insert XML stanzas, the confidentiality of the media assets may be in jeopardy. An inserted command could instruct a file server to upload a file somewhere or modify files similar to the strategy of the PGPCoder trojan. Therefore, we need end-to-end data integrity for XML stanzas exchanged by profiles on top of the core.

[7] Orb Privacy Policy; `http://orb.com/privacy`.

If an attacker compromises the data integrity of service discovery messages, the media network may not function as expected. This is an annoyance for the user but does not compromise the private media files—it is an attack against availability.

Availability

The availability depends on the data integrity of the service discovery. A compromised server may not send presence updates to clients or not route messages. This is only an annoyance for the user and the users could simply move their accounts to a different server; the media assets were never compromised.

A different way to attack the availability of the server is a *Distributed Denial of Service Attack* (DDOS) affecting the server on the networking layer. The server itself is not compromised, but unreachable for its users. This will disturb external devices and the communication with them will be limited. A DDOS can also be mounted on any mail and web server and preventing such an attack is outside the scope of the XPMN specifications. Additionally, the complete home network can be a DDOS victim and there is nothing the XPMN architecture can do to prevent it; the home network is more or less offline. If the server is unavailable, external devices are unable to communicate with each other and with devices in the home network. Requirement 13 already states that the media network must be operational on link-link level in case the DSL connection is down. As a result we accept this threat but ignore it: it is not a critical attack. An attacker does not gain much, especially no access to the media assets; therefore, it is unlikely to happen often.

Peer Entity Authentication

Peer entity authentication is required to ensure confidentiality and data integrity. Without it a client cannot be sure it is communicating with the expected peer. For instance, if a file server receives a request to stream a file somewhere, confidentiality and data integrity is useless if the file server cannot verify that the request was sent by a legitimate peer.

The server cannot be trusted with user authentication since an attacker may compromise it. If a client does not trust the server, it has to validate by itself that a peer belongs to its personal media network. The client-to-client security layer must have an authentication mechanism for clients to check the identity of its peers. This raises the question how a client can get the knowledge of a new device added by a third client. The XPMN security layer must provide some sort of authentication management.

5.3.2 Attacks on Devices

Once a device is part of the media network it should be able to interact with others without further interaction by the user regarding the security layer. If a device is compromised after

it has been integrated, the user must be able to remove that device. Every device can expose vulnerabilities due to implementation errors, for instance, buffer overflows in the XML parser.[8] XMPP limits this threat by making it possible to only route stanzas from clients of the same user or from contacts in the roster. An attacker must have control over the server or a known device to exploit such a security vulnerability. If a client reduces the communication with an unauthenticated peer to a minimum, the risk of malicious stanzas from the server is small. A client should only allow device and service discovery and setting up a secure end-to-end communication before the peer has to authenticate to the client.

Besides general implementation errors, the various types of devices have the following vulnerabilities:

- A mobile device such as a phone or a laptop has a higher risk of getting lost or stolen than the devices in the home network. There is nothing the XPMN protocols can do to prevent a device from getting stolen, only to provide an interface to banish these devices from the media network afterward. The media network is vulnerable in the time span between the device getting stolen and the user removing that device. As a workaround the device may require the user to enter a password to access the media network; a laptop may always request a password when it is turned on again. An effective security layer on the laptop is an effective security measure for the media network, too. For mobile phones this is more complicated: the phones are kept turned on, and it is not very user-friendly to enter complex passwords using the keypad the phone provides. As a result the mobile phone might have no protection and then can be used by the thief without further knowledge.
- A computer such as a PC in the home network or a laptop can be affected by a virus or a trojan. This can compromise the whole system, because the attacker gets access to whatever verification mechanism the media network uses. If a virus rewrites some parts of the XPMN application such as adding a back door, the media network itself cannot detect this. If a user can access the media files from the PC, so can the virus. This problem has to be solved outside the XPMN architecture. If the PC is secure in the first place, it does not only benefit the media network. Up-to-date anti-virus software, a firewall, and a good system password should be mandatory for a PC in the media network.
- A closed box only providing services to the media network is the least vulnerable kind of device. Unlike a PC, its actions are very limited and different vendors having their own firmware reduces the risk of large scale attacks.

A successful attack on a device will compromise one of the main security objectives: confidentiality. The only action the media network can provide is a way to remove such a device. Unlike the XMPP server, a client is not such an interesting target: an attacker with control over a server is a risk to every user on that server while attacking one device will only provide access to one personal media network.

[8] There are several applications vulnerable due to errors in XML parsers; even though patches exist, they are still at risk if the system is not kept up-to-date with security updates. This includes popular browsers such as the Internet Explorer, Mozilla Firefox and Opera as well as the Apache Tomcat XML Parser installed on several web servers.

It is very unlikely that the same attacker can compromise both the server and a client. The user should be able to remove a rogue client with the assistance of the server by denying it access to the XMPP network, and end-to-end encryption provides the required security against a rogue server. As a compromise to usability concerns, the security layer should prevent an attack on either server or client and may still be vulnerable to the scenario where one attacker has control over both.

5.3.3 End-to-End Security Layer

An end-to-end XPMN security layer must provide end-to-end authentication, confidentiality, and data integrity. The SASL-based client authentication to the server with one shared password is not sufficient for XPMN profiles. There are various peer entity authentication methods based on two basic principles:

- Two clients have a common knowledge only they share: a password, pass phrase, or the symmetric encryption key that will be used for encryption and signing. A simple security layer such as the one included in Mbus has one shared secret for all clients on the bus. Client authentication is impossible with only one shared secret. If each pair of clients has their own shared secret, this secret can be used for client authentication. If this secret is an encryption key, only the legitimate peer can encrypt and decrypt messages.
- A client proves its identity to others with authentication information no other client can provide. A commonly used method for identification is based on asymmetric encryption with two distinct keys. A message encrypted with one of these keys can only be decrypted with the other. Each client owns such an asymmetric key pair and keeps one key to itself (private key) and the other one is available to all other clients (public key). If a client encrypts something such as an authentication token with its private key, the peer can verify the identity by decrypting the message with the public key of the client. Only the expected peer could have encrypted the token with the appropriate key.

Distributing shared secrets for each client/client combination is impractical in terms of scalability. If a new client is added to a media network already having ten devices, a mechanism must be found to generate ten new shared secrets and distribute them to the corresponding clients. This is one reason why XMPP ESessions (see section 4.4.2) do not work in the XPMN scenario.

Identification with asymmetric key pairs is more practical. With this method, only the public key of the new device must be made available to all devices, or one already trusted client may be used to retrieve the public key of another. The private key can be used to sign a message to verify the sender and to ensure the integrity of the message. The message can be encrypted to ensure confidentiality using the public key of the recipient. Unfortunately, asymmetric encryption is very complex in comparison to symmetric encryption and requires much more CPU power. It is too time-consuming to use it for all messages, especially for devices with a low CPU performance such as a set-top box or a TV set. Thus some protocols use a hybrid encryption: they use asymmetric keys for the initial authentication and a strong symmetric key for

confidentiality and data integrity. The symmetric session key is exchanged using asymmetric encryption.

There are several symmetric and asymmetric encryption algorithms. An algorithm considered secure today, can be a considered to be too weak in the future. The end-to-end security layer should be flexible about the algorithms used; this applies for the asymmetric identification keys as well as the symmetric session encryption algorithms.

A self-made security protocol for the media network would create the risk of doing something wrong. Other specifications such as *Wired Equivalent Privacy* (WEP) have shown that it is easy to make mistakes when designing a new protocol—mistakes that were discovered after wide deployment. For this reason an existing security protocol should be used.

Transport Layer Security (TLS, RFC 5246) [DR08] provides authentication with X.509 certificates (asymmetric encryption) and uses a symmetric session key. TLS is widely used in the Internet for secure communication with web servers (HTTPS), mail servers (IMAP START-TLS), some of the peer-to-peer protocols described in section 4.2, and is already included in most XMPP clients to communicate securely with the server. A TLS server proves its identity with a certificate and can request a certificate from the client. Client certificates are not widely used because a web browser or mail client possesses no certificate a server can validate. For XPMN end-to-end security one client acts as TLS client and the other as TLS server requesting a client certificate. Since TLS is widely deployed, many security experts have looked at the specifications for possible security flaws and into open source implementations for possible programming errors. Therefore, TLS seems to be a good solution to secure the media network. The *Transport Layer Security Working Group* at the IETF takes care of adding new ciphers to TLS and removes weak ones in newer versions. The XMPP end-to-end security extension can leave the burden of defining ciphers to security experts in the IETF. An XPMN application can use existing TLS libraries that will include new ciphers and the application developers do not need to keep up-to-date with the TLS development.

If each XPMN device has its own X.509 certificate there needs to be a way to manage the certificates for authentication—a client has to know what client certificate belongs to its media network. There are two different ways to manage certificates:

1. The certificate is signed by a *Certificate Agency* (CA) all clients trust and all clients have the certificate of this CA. A new client must send its public key to the CA for the certification process (X.509 Certificate Request) and a client verifying the client certificate must check the CA for revoked certificates. The JID can be stored as *subjectAltName* in the X.509 certificate (see RFC 3920, section 5.1).
2. The X.509 certificates are self-signed. This does not provide any authentication based on the X.509 certificate chain. The list of device certificates belonging to the media network is stored somewhere inside the network and is independent of the X.509 certificate infrastructure.

The usability section of this chapter will explore X.509 certificate management further. Both ways are acceptable for the security layer; the choice depends on how usable they are.

The X.509 certificates may also be used to log in on the server. This provides a login method for the client without knowing the password. A device can easily be removed from the media network by removing its certificate from the server. If the client does not know the password and the server does not list the client's certificate as valid, the client is unable to connect to the XMPP server and thereby excluded from the media network. Yet, the client can still claim to be part of the personal media network in the home network using IP multicast.

5.3.4 Access Control

Setting up a secure TLS session between two clients with authentication is not enough if personal media networks interconnect. The XPMN core needs to define access control for a friend's client. The details of the access control have to be defined by the profiles; the core does not know about files and directories on the media server or how to list recordings on a TV tuner. The core has to provide the framework for access control by defining common requirements of all profiles. This includes the names or groups of clients that are allowed to access a function and a time span. The profiles have to define the details. If possible, the access control should be specified in a generic form that can be understood by every client, even without supporting the corresponding profile itself. Depending on knowledge of a profile to configure it reduces the number of clients that can be used to do so.

Detailed access control is necessary for a strong security layer. It would be nice to be able to restrict the access of a device to certain functions; for instance, a TV recorder could be restricted to write files on a media server but not read them. However, setting up these detailed access controls may be too complicated for the average user. Moreover, the devices with the highest security risk are the devices that need to call most actions: a laptop should be able to access all other devices and has the risk of getting stolen or compromised by a virus.

Therefore, the access control of the media network will not discriminate between devices and only manage rights per-user. Each device of a user has the same rights. From this it follows that all devices in one personal media network can access all the others without constraints and a user only defines additional access for a friend.

Similar to the UPnP Security Template, the access control should be stored on the device it affects. This makes the access control independent of the server or an access control device; the device itself knows best what configuration options it provides.

5.3.5 Summary

To summarize the security goals, the end-to-end security layer must ensure confidentiality and data integrity for XPMN profiles and any kind of media transfer. This means the XMPP message and IQ stanzas must be encrypted (confidentiality), the encrypted stream must be secure against being altered (data integrity), and must contain an authentication mechanism (peer entity authentication). This can be achieved by setting up a TLS session between two clients.

The outlined security concept is far from being perfect: it may expose discovery information to unauthorized clients and does not allow access control for each device individually. However, it provides confidentiality and integrity for the media assets which is the main security requirement. The XPMN core must specify the following XMPP extensions to meet the refined requirements:

1. An extension to send message and IQ stanzas over a TLS connection between two clients. This meets requirement 12 and provides confidentiality and data integrity for profiles. The extension must work over In-Band Bytestreams to guarantee that clients can communicate.
2. A way to add TLS to the media data transport as defined in section 5.2.1.
3. An extension to manage X.509 certificates of the user. The certificates must be readable for all clients of the user and friends in the roster. The certificates and the corresponding private keys should also be used to authenticate against the XMPP server. This will meet requirements 18, 19 and 20.
4. An extensible access control mechanism must be added to the core and profiles need to specify the details (requirement 24).

An XMPP server can be configured by the user to block any communication with devices not owned by the same user or friends in the roster; the only exception is the presence request to be added to the roster. This reduces the list of entities in the XMPP network knowing of the device's existence and who can communicate with it to the user's devices, the friends' devices, and the XMPP servers used by the user and the friends.

5.4 Usability Considerations

The media network must be easy to use or it will not be accepted. Unlike UPnP it requires user interaction for device and friend management to address the basic security issues. This interaction should be as simple as possible. The fact that the media network uses TLS with X.509 certificates should be hidden from the user. The user should not be forced to check X.509 certificate fingerprints or signatures to add a new device. This section refines the previously made security requirements from the usability point of view.

5.4.1 Bootstrapping

The first task for the user is to set up the media network itself. XMPP requires the user to have an account at an external XMPP server; the user has to provide a user name, a password, and the server name. XEP-0077 (In-Band Registration) provides a mechanism to create an account over XMPP and the user does not need to create an account using a web browser or any other kind of software.

Setting up an account is not a problem for instant messaging, the use case XMPP was designed for. A chat client always has a way for user input and displaying chat messages—in most cases a PC with an alphanumerical keyboard and a monitor. For the media network the situation is more complicated: some devices such as a DVB tuner have no user input or output, and some devices such as a TV set have a remote control with limited textual input capabilities. We can safely assume that the user always has at least one controller to interact with the system or the XPMN devices will do nothing. Possible devices are the TV set, a mobile phone, and a PC. The devices without user interface cannot contribute to the bootstrapping process. We need a simple way for the user to provide the credentials with these three types of controllers.

- The **PC** is the easiest controller for the XPMN bootstrapping process. The software can ask for a user name, server name, and password for an existing XMPP account or to create a new one. This is similar to setting up an XMPP chat client or an e-mail account. The XMPP server name may be set to a suitable default value so the user only has to choose a user name and a password.
- The **mobile phone** is a bit more complicated. It has a decent display to show information to the user, but relies on the inferior keypad for user input. Providing a user name and password must be done with this limited interface. We assume that users who use the mobile phone to interact with the media network are familiar with that keypad. Other operations on the mobile phone such as managing the address book or writing a short message require the use of this input method as well. Mobile phones with a touchscreen can display a keyboard on the screen for easier text input; some smartphones have a small integrated keyboard.
- The most problematic controller is the **TV set** or the set-top box connecting the TV set to the media network. While it has a very good visual output to the user to show text messages, its input capabilities are even more limited than the mobile phone's. Most remote controls do not have the letters A-Z on the keypad to help the user with text input. Yet, the TV set is the logical choice to control the media network. Services in the media network may also need text-based user input and we can simply require the TV set to have a remote control with the letters on the keypad. This makes the input similar to the mobile phone—but unlike mobile phone users, TV set users may not be familiar with this kind of interface.

The whole process of requiring an account must be explained to the user. Users are familiar with setting up accounts like this: many web-based services on the Internet require the user to choose a user name and a password. If the media network should only be used link-local, an account is not required and it may be possible to connect devices to a local media network without any external server registration. If the user chooses link-local-only operation, it may be possible to provide the credentials later to expand the media network to the outside.

A completely different approach is to choose a pre-defined server and use a random user name and password. In that situation the user interaction is reduced to acknowledging the creation of a personal new media network and everything else is running in the background. The devices

must exchange the credentials without the user ever seeing them; seeing two random strings as user name and password will confuse the user.

The decision between user-provided and random credentials depends on other user interactions and has to be postponed until the device and friend management is refined.

5.4.2 Certificate Management

Section 5.3.3 suggests two methods for certificate management: using certificates signed by a CA and self-signed certificates. The decision should be based on how user-friendly the certificate handling can be made; a user will not manage certificates based on fingerprints or manually copy them between devices. Distributing a new certificate to all existing clients manually has similar problems in terms of scalability to creating a unique shared secret between two devices (see section 5.3.3).

Personal Certification Agency

The available public key infrastructure (PKI) for X.509 is based on root certificates used to sign other certificates; some of these certificates are again allowed to sign others, some are not. The user must create a certificate and send it to a CA for signing. Gutmann checked the usability of this task in 2003: "A common complaint about PKI is that it is simply too hard to use at the end-user level. Somewhat surprisingly, there exists no PKI equivalent of DHCP or BOOTP for automated, transparent PKI setup, leaving the certificate user experience similar to the process of bringing up an X.25 link. [...] A representative non-technical user who tried to obtain an (unverified) certificate from a public CA took well over an hour for the process [...] In contrast [...] connecting a new system to a LAN usually requires no more than a user name and password for access to the appropriate server, with the rest being taken care of transparently via mechanisms such as DHCP" [Gut03]. This situation has not changed since 2003: the process is not as simple as providing a user name and a password.

Instead of using the existing PKI, a user can create a private root CA and only the certificate of the CA must be known by all devices. This takes the current CAs out of the picture. Their involvement is questionable anyway: why should users trust organizations they have no relationship to? In addition, they have to pay money for a signature and cannot sign device certificates themselves; every certificate must be signed by a CA with the rights to sign other certificates.

The concepts of a personal CA and a certificate signed by a well-known root CA can be merged to overcome the signing problem. The architecture of a two-layered PKI model by Hwang et al. [LLH+07] consists of a home gateway with a verifiable certificate (global PKI layer) and local device certificates (localized PKI layer). Two home routers create a trust relationship based on their global certificates to verify local client certificates.

With or without the global PKI layer, one device in the media network must be the CA. It must create the public and private key and create a valid (self-signed) certificate for it. While it is

simple to create an unsigned certificate to be signed or a self-signed certificate,[9] setting up a CA is much more difficult. Some libraries refuse to create a CA without a protecting password; therefore, the user must enter the CA password every time a new device is added. Depending on the TLS library, an X.509 CA creation may require the user to provide some additional data such as contact information. This information is useless for the XPMN operation itself. Even though it is only an implementation issue, the current practice is a drawback for wide deployment.

The CA device must be available when a new device is added. If a new device is added by another one, it must send the new certificate to the CA for signing. The CA is also responsible for revoking certificates; the CA must provide a mechanism for others to retrieve the list of revoked certificates. The client certificate must be revoked if the device is compromised or stolen. This is very time critical: a weekly check on the CA for revoked certificates will keep the media network at risk for such a time span. Therefore, devices must poll the CA very often (pull), or the CA has to send the revocation information to all devices (push). This would require the CA to keep track of all devices it has informed. For XPMN interconnection this will get extremely difficult: the CA must not only inform the clients in its own personal media network, it must also send revocation information to all devices of all friends. The Publish/Subscribe XMPP service could be used to distribute this information. The CA publishes the revoked certificates and the pubsub server sends this information to all clients.

Clients Web-of-Trust

Instead of using the pubsub server to distribute the information about revoked certificates, it can be used to retrieve the list of valid certificates. If a device can query for a list of all certificates in the media network, the X.509 certificate chain is no longer needed. The devices can operate with self-signed certificates.

The dependency to the pubsub server makes it impossible to add a device when the home network is offline. However, both a home network being offline and adding a device does not happen very often. It should not be a problem that the user can only add a new device to the media network if the Internet connection is available. In case the home network router is the device in question, it must first be configured to connect to the Internet before it can be added to the media network. The XPMN link-local-only operation needs a pubsub server inside the home network.

There is one major problem with this solution: we do not trust the server. The whole point of using the client certificates is to assure client-to-client security. If the server manages the set of valid certificates, a rogue server can add certificates and the whole security layer is useless. Similar to the two-layered PKI architecture, the router in the home network as a trusted user device could be used as pubsub server. This requires special XMPP support in the router. Besides the router, any device in the home network never turned off can be used as trusted

[9] Creating a key pair and an unsigned or self-signed certificate on Linux-based systems requires only one call of the openssl binary. This can be done in the background without any user interaction.

pubsub server. Putting some management tasks on a client in the home network will also work link-local and should be considered in the architecture.

If the certificates are handled by the XMPP server, clients need to be able to verify that the certificate was uploaded by a legitimate client. Clients could sign each others' certificates independent of the X.509 certificate chain. The server will store the certificate and all signatures from clients next to it. This creates a *web-of-trust*.

User Certificates outside the X.509 Certificate Chain

The web-of-trust causes problems when a device is removed from the network. All other devices added by the removed device are now without a valid signature. A different approach is to add a user certificate with a similar purpose as a CA. All client certificates will be signed by the user certificate. Unlike the CA, the signature is outside the X.509 certificate chain. Thus it is possible to sign a client certificate with a user certificate that does not have the right on X.509 level to sign other certificates. It is also possible to sign a certificate with more than one certificate or even with something completely different such as OpenPGP keys. The latter may be used to verify the media network of a friend. The user's private key can be stored on more than one client and only these clients can add new devices to the personal media network. A new client or a friend only needs to know the user certificate and needs to have read-access to the pubsub server to verify all clients of that user. The link-local-only operation requires a trusted certificate-exchange client in the home network.

An attacker can add new devices if a device with the user's private key is compromised. The private key may be protected by a password, and the user has to enter the password when adding a new device which does not happen often. Unfortunately, the devices that most likely add a new device are the devices with the highest risk of being compromised: mobile phones, laptops, and a PC in the home network. Therefore, each device should be able to add a new device with the assistance of a third device having the user private key. If the private key is not on the mobile phone and the mobile phone just relays the request to another client, a stolen mobile phone is not a security risk for the user certificate anymore. The most trusted devices are closed boxes; the home network router may be a suitable device for this task. The key could also be stored on the server and encrypted with a password.

5.4.3 Device Management

In the worst case scenario, a new device added has no keyboard-like input and no display to show a message to the user; or the input may be reduced to only a few buttons. It is unacceptable to use this interface to provide the user name, password and server address—it may not be even possible if we choose random user names and passwords. Besides that, the system must sign the client's X.509 certificate with the user's private key. This has to be done in the background without bothering the user.

There are three possibilities how a device may be added: the new device detects another using link-local messaging (e.g. in the home network), it is physically in range of another device (e.g. a mobile phone near the new device), or it can neither detect another device link-local nor is physically near one.

When devices detect each other link-local, a device already in the media network can provide the required credentials to the new device to log in to the server. TLS with X.509 certificates only provides end-to-end security if the clients can verify the certificate of their peers. This is impossible if a new device is added to the media network and self-signed certificates are used. In most cases neither the new device nor the device already in the media network can verify the other's certificate.

This is usually the point where the user has to verify the certificate's fingerprint and accept it on both devices. This is unacceptable for several reasons. First and foremost, this is not user-friendly because certificate fingerprints are long cryptic alphanumerical strings. As a result, few people check certificate fingerprints in protocols such as *Secure Shell* and rely on *leap of faith*. This means that if the certificate does not change in future sessions, a client at least knows that it is talking with the same entity it talked with during the first session. However, that entity might be a man-in-the-middle rather than the assumed communication partner.

A second reason why fingerprints may not work are device limitations. A simple TV tuner device may not have a display to show the user the certificate of its peer. The security layer of the media network should bootstrap a new relationship based on something different. One solution is a one-time password, for instance, a short four-digit PIN similar to Bluetooth pairing. Here, the user has to enter the same PIN on both Bluetooth devices for the initial trust setup. There are three distinct types of devices for such a bootstrap protocol:

1. The device supports direct user input; for instance, a keyboard (laptop), a keypad (mobile phone), or a remote control (TV set). The user can enter a short numerical PIN. We assume here that at least one of the two devices has such a user interface.

2. The new device can display a random password or PIN if it does not provide user input capabilities but does have a small display. The user can then enter the provided password on the other device.

3. The new device might not have enough buttons for user input and have no output capabilities. In that situation the password will be fixed. Similar to Bluetooth pairing with simple devices such as a headset, the password will be written in the manual or printed on the device. For security reasons the device should not use password-based authentication without any user input. Many Bluetooth devices have at least one button to set the device into pairing mode.

Besides being more user-friendly, a password increases the security. Unlike fingerprint verification, the user cannot accept a new device without verification. The user is forced to enter the password on both devices (or on one if one does not provide user input capabilities)—the user cannot trick the security layer. The algorithm must be secure even when using simple four-digit

passwords with only 10.000 possible combinations. Possible attacks are dictionary attacks and brute force as man-in-the-middle. Offline attacks can be ignored since the password is only used for the pairing process; if an attacker retrieves the password after the pairing is complete, it is useless (except for devices with a fixed password).

Alternative technologies such as *Bluetooth*, *Infrared* (IR), or *Near Field Communication* (NFC) can be used if devices are physically near to each other. The closeness of the devices can be used for authentication. The XPMN core architecture should address this kind of pairing process even if currently available devices may not support it.

If the user does not have access to an already connected device, the credentials must be provided manually. If the system relies on text input, random user name and password combinations will not work. This is the most complicated scenario. The new device must have the user name, password, server name, and the user's private key to log in to the media network—a user cannot provide the private key without help from additional devices. However, adding a device this way is unrealistic: a mobile device is in the home network when the user is at home and a mobile phone is close to the user; therefore, physically close to every external device the user is close to. Adding a device does not happen often; the likeliness that the user has to add a remote device without another is very small. To cover this rare scenario, the user can temporarily store the credentials on a USB stick. The XPMN core should provide a pairing method based on USB mass-storage devices transporting the required credentials.

Besides adding a device to the media network, there must be a way to remove a device later on. The user must be able to identify a client based on a name to remove it. The certificate's fingerprint may be unique, but unusable for device removal in terms of usability.

5.4.4 XPMN Interconnection

Adding a friend to the personal media network is similar to adding a new device. The user must somehow verify the friend's user certificate without actually comparing fingerprints. A small password as shared secret would be sufficient or the user certificates may be signed by a trusted CA. A different approach is the usage of OpenPGP keys. A user signs the X.509 user certificate with the private OpenPGP key and the friend can verify it based on the OpenPGP web-of-trust. OpenPGP is not widely used in comparison to the potential user base of the media network, requires manual fingerprint verification, and it is questionable whether a user trusts a friend of a friend of a friend (web-of-trust). Password-based verification is the most likely scenario.

To add a friend, the user must provide the friend's bare JID. Choosing random user names and passwords for an easy bootstrap process is a disadvantage in the case: the users do not know their own user names. To have interaction between two media networks, the users must exchange their JIDs. For this reason the media network should not be created using a random user name and password. The bootstrap process is now similar to setting up an e-mail account: the user must choose an XMPP server (mail provider), a user name, and a password. The e-mail address is comparable to the bare JID.

To include external services such as Flickr or YouTube, the user may either provide the service's password to the media network, or perform the same trust bootstrap mechanism as with friends. If the user authenticates to the website with user name and password, and the site uses HTTPS with a valid certificate, this already existing trust relationship can be reused. After logging in, the site could display the XMPP JID of the service and the bootstrap password.

5.4.5 Usability for Services

UPnP has shown that many services are possible without complex user interaction. In fact, controlling a remote service from one device is similar to a service on the same device. The user may not even need to know what device is being used or that it is a remote device at all. It does not matter which DVB tuner is used as long as the recording is scheduled.

In the M4 project [BKML07a] the TZI developed a demonstrator on a mobile phone that allowed the user to play a video on the phone's screen or on an external TV set. The mobile phones API includes local video playback (JSR-135) and the API for the external playback was designed to match the internal one. An application developer only needs to exchange the statement creating the internal player with the one from the M4 library to support external playback; the commands to control the playback reuse existing classes and interfaces.

5.4.6 Summary

The normal operation of the XPMN core should not require any user interaction except for bootstrapping the security layer. Requirements from the networking layer such as NAT penetration should function without any user interaction—the user should not even need to know about the problem. Only device and friend management require the user's attention and must be specified in a user-friendly way.

1. Creating a media network requires at least a user name, a password, and a server address. The asymmetric key pairs and the X.509 certificates for the user and the clients should be generated automatically without user interaction. A PC is the preferred device for the bootstrapping process because it provides the best suiting user interface for this purpose. A mobile phone may also be a suitable device if it has a keyboard or a touchscreen showing one.

2. When adding a device, we assume that the new device can detect another device of the media network: either link-local or they are physically near each other. A USB mass-storage device with the credentials can serve as a fallback mechanism.

3. Adding a friend is similar to adding a device: their devices can either detect each other or the users have to exchange their bare JIDs. A password can be used to bootstrap the trust relationship.

4. A human-understandable resource name is required to remove a device from the personal media network. A cryptic unique identifier such as a certificate fingerprint is not an option as a user will not be able to know which device to remove.
5. Friend access requires some sort of access control for specific services. This is outside the scope of the core protocol and depends on the actual service. The core architecture must provide the basic schema and the profiles have to define the details.

It is important to use simple passwords or similar technologies to make adding devices and friends as user-friendly as possible. The user should not need to be aware that the security layer is based on cryptic X.509 certificates and should neither see a fingerprint nor need to know what it is. A password is a suitable way for initial authentication, but even simpler ways should be outlined for future devices supporting NFC or similar technologies.

5.5 Standardization Process

Some of the required XMPP extensions are not limited to the XPMN use case. These extensions could be useful for chat clients as well.

- The XMPP file transfer would benefit from direct TCP connection. Today, most of the file transfer is done using In-Band Bytestreams. Due to the fact that XMPP servers limit the bandwidth per client, file transfer in XMPP is very slow. SOCKS5 Bytestreams are often not used because there is no fallback mechanism if they do not work.
- End-to-end security in XMPP can be used to secure chat messages and file transfer. The media network relies on the same basic stanzas like any other XMPP application: message and IQ. Tunneling these stanzas over a TLS connection is a suitable solution for many use cases.

Therefore, the XMPP extensions should be developed inside the XMPP Standards Foundation. The process has to include discussions with other developers outside the media network context. Standardization is a long process; there are many XMPP extensions superseded by a different approach and each extension went through multiple revisions until it reached draft or final status. In April 2009, even popular extensions such as Jingle and Publish/Subscribe were still marked as experimental. The various approaches in the standardization of the XPMN core protocol are described in the next chapter.

Of course not everything can be solved in a generic way. A media server has to include some sort of file transfer; so do many chat clients. But the scenario is completely different: a media server has to provide listings of files, access control, and seeking in the file. In contrast, the current XMPP file transfer is a simple push technique and the initiator sends one specific file to another client. Trying to put these two use cases into one specification would make the specification too complicated for the simple file transfer.

The extensions should also be open for future enhancements. Some services may include some optional vendor specific information. An important rule in XMPP solves this problem: any

XML element can have a child outside the specification from a different namespace. A client has to ignore unknown elements. If the additional information is only optional, interoperability between clients with and without an extension is given. Besides that, a client can query another about its capabilities (XEP-0030). Chat clients already do this to determine whether or not the peer supports a specific extension.

Chapter 6

Core Architecture

With the refined requirements in place, it is time to develop the XPMN core based on XMPP. This chapter describes extensions to fulfill the requirements, the concepts behind them, and the development cycle that was needed to specify the extensions the way they are now. The extensions are split into four sections: secure end-to-end streams, device management, external users and their access restrictions, and a rough architecture for a generic service provider. After that, an optional special trusted management client is introduced to move some tasks away from the XMPP server. The chapter ends by checking the developed solutions against the requirements from section 2.2 and chapter 5.

After the XMPP Summit 2009 in Brussels, the XMPP community decided to form a new *XMPP Working Group* in the IETF. Besides updating the existing XMPP RFCs, the working group added end-to-end security on its charter. There are many security experts in the IETF and developing end-to-end security inside the IETF provides additional feedback from those experts. From this it follows that some extensions developed in this thesis are now published as Internet Drafts and the corresponding XEPs are obsoleted by them.

This chapter gives only a brief overview about the developed XMPP extensions and Internet Drafts. The actual specifications go into more details regarding error conditions and include more and elaborated examples.

6.1 Secure End-to-End Streams

In section 5.3.3 we came to the conclusion to rely on Transport Layer Security (TLS) for a secure end-to-end client communication. TLS provides confidentiality and data integrity, and uses X.509 certificates for authentication; without a way to verify the peers' certificates, TLS is vulnerable to man-in-the-middle attacks. As discussed in section 5.3.1, peer entity authentication is required to ensure confidentiality and data integrity. This section only covers how to open a TLS session between two clients and not how to verify a certificate; this will be covered in the next section about device management.

Basically there are two ways clients can communicate: through XMPP servers or link-local (serverless messaging). For link-local communication a client opens a direct TCP connection to the other. It seems to be easier to add TLS in this scenario first. Thus, we start with link-local communication and after that discuss several methods how to use TLS between two clients if they are connected over a server without an end-to-end TCP connection.

6.1.1 Link-Local Communication

The link-local communication defined in XEP-0174 (Serverless Messaging) [SA08a] uses multicast DNS and DNS-SD (see section 3.2.1) to announce and discover clients in the local network. This discovery mechanism is not designed to be secure. The multicast announcement replaces the presence information distributed over the server; the DNS service type is `_presence`. The DNS TXT record contains an entry with a TCP port for clients to connect to, the JID, and the presence status. A client can never be sure that the information from a peer is accurate.

Since XMPP uses XML streams for client-to-server and for server-to-server communication, it is only logical to use XML streams for client-to-client communication, too. XEP-0174 defines an additional attribute in the opening <stream> element to provide the client's JID to the peer.

During stream setup, after connecting to a server, the server presents the client a list of features. Some of these features may be required, some may be optional, and some are only available if another feature was previously selected or not. The usage of TLS and SASL is negotiated over stream features. For link-local communication the recipient can provide stream features to be activated by the initiator in a similar way as for client/server communication. "XMPP networks use TLS for channel encryption, SASL for authentication [...]; these technologies help to ensure the identity of sending entities and to encrypt XML streams. By contrast, zero-configuration networking uses dynamic discovery and asserted machine names as the basis of sender identity. Therefore, serverless messaging does not result in authenticated identities in the same way that XMPP itself does, nor does it provide for an encrypted channel between entities." (XEP-0174, Security Considerations, section 13.1).

While XEP-0174 only mentions the TLS stream feature as being optional, an XPMN device must require it before exchanging stanzas—ignoring the issue of how to verify a certificate for now. The recipient behaves like an XMPP server and therefore, also takes over the role of the TLS server in the TLS handshake. If the recipient requests a client certificate in the second phase of the TLS handshake, both clients have the certificate of their peer after the TLS session setup is complete. This ensures authentication if both clients have a way to verify their peer's certificates.

6.1.2 TLS over In-Band Bytestreams

For server-based communication the situation is more complicated since the clients do not have a direct TCP connection between each other. Instead, both have a connection to their servers.

TLS does not limit its usage to TCP and only requires some characteristics of a TCP stream: data transparency, ordering, and error-correction. In-Band Bytestreams (XEP-0047) provide the same service: arbitrary data is Base64 encoded and encapsulated in IQ stanzas. Sending IQ stanzas ensures ordering and guarantees data delivering.

```
<iq from='romeo@montague.net/orchard'
    to='juliet@capulet.com/balcony'
    id='gfgdpl87'
    type='set'>
  <xtls xmlns='urn:xmpp:tmp:xtls' sid='MySid'/>
</iq>

<iq from='juliet@capulet.com/balcony'
    to='romeo@montague.net/orchard'
    id='gfgdpl87'
    type='result'/>

<iq from='romeo@montague.net/orchard'
    to='juliet@capulet.com/balcony'
    id='polaqhty'
    type='set'>
  <open xmlns='http://jabber.org/protocol/ibb'
        sid='MySid' block-size='4096'/>
</iq>

<iq from='juliet@capulet.com/balcony'
    to='romeo@montague.net/orchard'
    id='polaqhty'
    type='result'/>
```

Example 6.1: Open XTLS In-Band Bytestream

The basic principle of *TLS over XMPP In-Band Bytestreams* (XTLS) is that the entity in the role of the TLS client opens an In-Band Bytestream to the peer and tunnels the complete TLS stream including the handshake messages over it. Example 6.1 shows the XTLS negotiation with the IBB identifier "MySid". To accept the XTLS session, the peer sends an empty IQ result; to reject the session various kinds of IQ error messages can be used. After agreeing to use TLS, the initiator opens an In-Band Bytestream with the announced session identifier.

From now on, the <data> element in the IBB namespace with the negotiated session identifier can be used for the TLS stream; including the handshake and the encrypted XML stanzas. The chunks provided by the TLS library are sent Base64 encoded in the <data> IQ stanzas to the peer. The peer has to convert the Base64 encoded data back into its binary form and provide it to the TLS library.

The initiator starts the TLS handshake and takes over the role of the TLS client; the recipient is the TLS server requesting a client certificate during the TLS negotiation. The peers exchange XML stanzas over the TLS connection similar to server-based routing after the handshake is complete, but instead of sending a stanza to the server, a client sends it through the TLS layer

over the In-Band Bytestream (see figure 6.1). A client must ignore the 'to' and 'from' attributes in the stanza and only rely on the attributes from the surrounding bytestream.

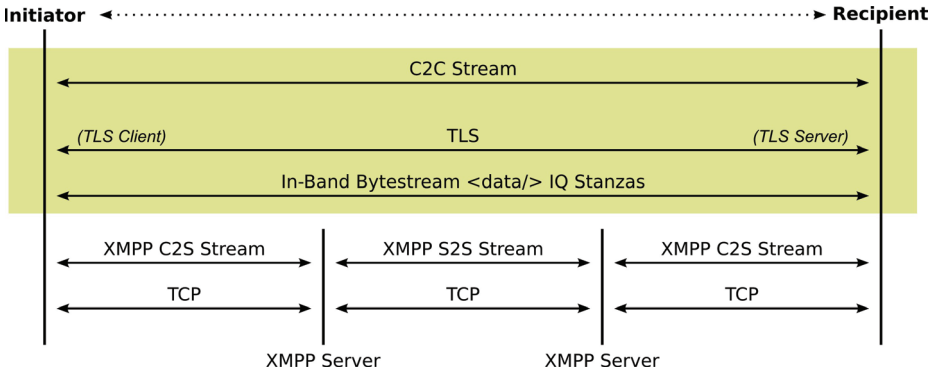

Figure 6.1: TLS over In-Band Bytestreams

If the TLS stream itself becomes invalid, for example, on an authentication error, the In-Band Bytestream must be closed by both clients. If no error occurs, it is recommended to keep the secure stream open until one client leaves the XMPP network. It does not cost many resources for a client to keep the TLS stream open; the TLS library only requires some memory for the session's state. Closing the stream and opening it again later on the other hand requires the two-way XTLS handshake, the two-way handshake to open an In-Band Bytestream, and the four-way TLS handshake including the expensive asymmetric authentication. To reduce the number of round-trips in subsequent connections the initiator could send the IBB open without waiting for the XTLS IQ result and start the TLS handshake before the IBB setup is confirmed by the peer.

```
<iq from='romeo@montague.net/orchard'
    to='juliet@capulet.com/balcony'
    id='gr60k14t'
    type='set'>
  <open xmlns='urn:xmpp:tmp:xtls'/>
</iq>

<iq from='juliet@capulet.com/balcony'
    to='romeo@montague.net/orchard'
    id='gr60k14t'
    type='result'>
  <proceed xmlns='urn:xmpp:tmp:xtls'/>
</iq>
```

Example 6.2: Simplified XTLS Setup

In an updated version, XTLS inherits some logic from In-Band Bytestreams: the client does not open an In-Band Bytestream anymore and sends the TLS data inside <data> elements of the XTLS namespace instead. Example 6.2 shows the updated open IQ stanza. The advantage of the updated protocol flow is to eliminate the two-way handshake opening the stream to save one

round-trip. The main disadvantage is code duplication in the XMPP library: XTLS does not just use IBB as it is anymore, but requires similar functionality in its own namespace instead.

Most server administrators reduce the load on their servers by restricting the bandwidth each client is allowed to use. This bandwidth limit is based on typical client communication behavior. Besides the initial setup, XTLS requires a similar amount of bandwidth (if stream compression is enabled during TLS negotiation).

XTLS was initially published as *XMPP Transport Layer Security* in June 2008. In December 2008, the updated session open procedure was added. The specification was never accepted as XEP due to alternative solutions described next.[1]

6.1.3 Jingle XML Streams

A different approach to open a TLS session between two clients is to use a similar technique as for link-local communication. The link-local extension consists of two parts:

1. **Device discovery**: the TXT records are used to announce the JID of the client and a port for another client to connect to.
2. **Stream negotiation and stanza exchange**: the clients open a stream, activate the START-TLS feature, secure the TCP connection with TLS, and exchange stanzas.

If two clients connected to the server open a TCP connection between each other, they would be able to use the same stream handling as for link-local messaging. Only the device discovery to determine the port differs.

```
<iq from='romeo@montague.net/orchard'
    id='hfo0t4wq'
    to='juliet@capulet.com/balcony'
    type='set'>
  <jingle xmlns='urn:xmpp:tmp:jingle'
          action='session-initiate'
          initiator='romeo@montague.net/orchard'
          sid='851ba2'>
    <content creator='initiator' name='xmlstream'>
      <description xmlns='urn:xmpp:tmp:jingle:apps:xmlstream'/>
      <transport xmlns='urn:xmpp:tmp:jingle:transports:bytestreams'/>
    </content>
  </jingle>
</iq>
```

Example 6.3: Initiate Jingle XML Streams

In this solution Jingle, originally developed for negotiating direct RTP sessions for Voice over IP, is used to open a stream between the clients. The XML stream from link-local communi-

[1] The specification is in the XMPP extension inbox at `http://xmpp.org/extensions/inbox/xtls.html`. Besides opening a TLS session, the XEP proposal also includes a way to negotiate TLS parameters that will be discussed in section 6.2.4.

cation can be considered as Jingle application requiring a TCP stream or something different with similar characteristics; for instance, In-Band Bytestreams. When *Jingle XML Streams* where introduced to the XMPP community in June 2008, the possible transport extensions were SOCKS5 as defined in XEP-0065 and In-Band Bytestreams. Example 6.3 shows a jingle-initiate IQ stanza opening a Jingle XML Stream.

The usage of SOCKS5 in XMPP requires the initiator to provide addresses of *streamhosts* for the recipient to connect to. One streamhost should be the initiator itself for a direct TCP connection. If the initiator is behind a NAT or firewall, SOCKS5 proxy servers can be used to relay the traffic. The transport setup will fail if the initiator has no knowledge about proxy servers it can use and is behind a NAT. In this case, Jingle allows changing the transport to something different such as In-Band Bytestreams; they will always work.

After the transport is open, the TCP-like stream is provided to the *XML Streams Jingle* extension. A developer can treat the provided transport in a similar way as link-local connections: exchange opening <stream> start-tags, negotiate stream features, secure the connection with TLS, and exchange stanzas.

This solution for a server-routed end-to-end TLS session is not as simple as the previous solution. However, it reuses the many existing XMPP extensions to open a secure connection: Jingle, the different bytestream extensions, and stream handling from link-local messaging. Future Jingle transports such as ICE-TCP will automatically be available for end-to-end security once they are specified.

The Jingle-based approach turns XMPP into a powerful secure third generation peer-to-peer network. The XMPP servers act as super nodes and can be used to route messages between clients in In-Band Bytestreams. Clients can also communicate directly to reduce the load on the servers. An In-Band Bytestream is only a fallback solution that is guaranteed to work.

Jingle XML Streams was introduced one week after XTLS was first published by the same group of authors. The XMPP council agreed on publishing Jingle XML Streams in favor of XTLS as XEP-0247. The Jingle-based approach was considered cleaner because it reused concepts from other extensions.

6.1.4 Jingle Security Layer: XTLS

At the end of 2008, some developers contacted the XMPP editor to give the simple TLS over In-Band Bytestreams specification another chance. For them, the usage of Jingle and the XML streams from link-local communication make the extension too complex. Not many XMPP libraries supported Jingle or link-local messaging at that time. Simply using TLS over In-Band Bytestreams seemed to be easier for them. This had to be considered; the success of an end-to-end security protocol depends on the acceptance of the developers. Both solutions have their advantages and disadvantages:

- It is faster to just open an In-Band Bytestream compared to Jingle XML Streams. It only requires a two-way XTLS stanza handshake and the four-way TLS handshake. After

that, the clients have a secure communication channel. Jingle XML Streams on the other hand have a more complicated negotiation: they require a two-way Jingle handshake, a transport layer handshake (at least a two-way handshake to open an In-Band Bytestream), a four-way handshake for stream setup and TLS feature negotiation, TLS itself, and a second stream initialization. This results in at least seven round-trips compared to three round-trips.

- Jingle XML Streams are more flexible. They can bypass the server by using direct TCP connections. The stream feature negotiation also allows other stream features such as SASL authentication for external services. An external service can authenticate itself with a certificate signed by a well-known CA and the client authenticates with a password using SASL.

The XMPP community agreed on using Jingle for end-to-end security but wanted to remove the complex stream setup from Jingle XML Streams. The stream setup is responsible for most of the round-trips and the complexity. It was pointed out, that if Jingle is too complicated for something as simple as opening an In-Band Bytestream between peers, the design of Jingle itself is wrong. In fact, Jingle is very simple. Only the usage for Voice over IP using Jingle is a complex use case due to ICE-UDP and the codec negotiation. File transfer over Jingle, the second available Jingle application at that time, is much simpler.

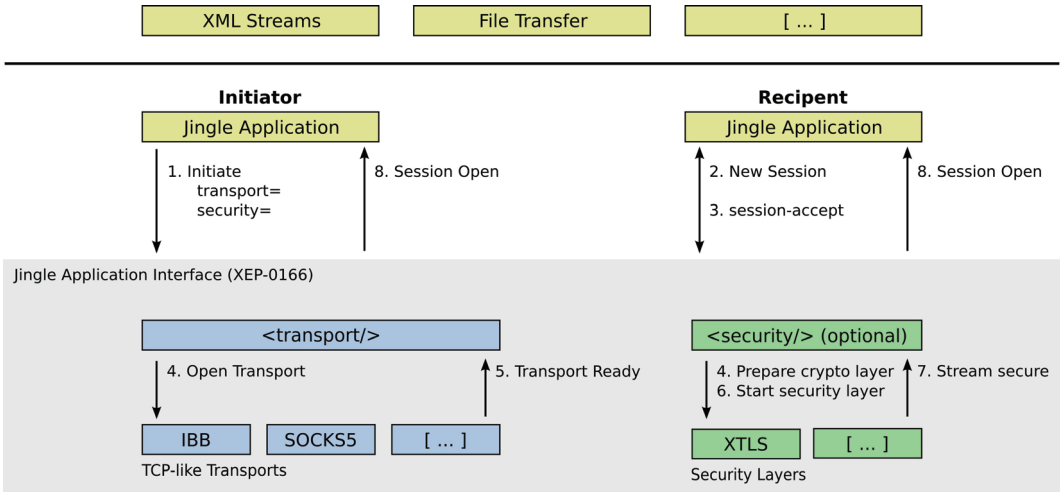

Figure 6.2: Jingle Transport and Security Layer

The logical step is to merge the two previous solutions: use Jingle to set up a stream, and instead of reusing the stream setup from link-local messaging with its many round-trips, use TLS directly on the stream and start exchanging stanzas afterward. This only adds one additional round-trip compared to the first solution. At the XMPP summit in Brussels in February 2009 Peter Saint-Andre pointed out, that RTP over Jingle includes the transport negotiation directly in the <jingle> element while In-Band Bytestreams and SOCKS5 Bytestreams are negotiated outside. Two new specifications *Jingle SOCKS5 Bytestreams Transport Method* (XEP-0260) and *Jingle In-Band Bytestreams Transport* (XEP-0261) adjust these transports methods to be

negotiated in the Jingle session-initiate and the session-accept. For In-Band Bytestreams, the number of round-trips using Jingle is now the same as in the first specification.

Since the Jingle application starts with the TLS negotiation, the usage of TLS could be separated from the stanza exchange. Instead of only transport and description, Jingle should have three layers: transport, *security* and description. The Jingle XML stream application only defines the stanza exchange and Jingle is used to open a TLS-secured stream. Figure 6.2 depicts an updated Jingle overview. Similar to the various transports, a <security> element can optionally be included in any Jingle negotiation, and a new security-info Jingle action is added for sending security-related information. The basic security framework was added to the Jingle specification in version 0.35.

```
<iq from='romeo@montague.lit/orchard'
    id='xn28s7gk'
    to='juliet@capulet.lit/balcony'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:0'>
          action='session-initiate'
          initiator='romeo@montague.lit/orchard'
          sid='a73sjjvkla37jfea'>
    <content creator='initiator' name='xmlstream'>
      <description xmlns='urn:xmpp:jingle:apps:xmlstream:0'/>
      <transport xmlns='urn:xmpp:jingle:transports:ibb:0'
                 block-size='4096'
                 sid='ch3d9s71'/>
      <security xmlns='urn:xmpp:jingle:security:xtls:0'/>
    </content>
  </jingle>
</iq>
```

Example 6.4: Initiate Jingle XTLS

Since the XTLS specification from section 6.1.2 was never accepted as XEP, the name "XTLS" was still free and many people like it. The usage of TLS as additional security layer in Jingle was called Jingle-XTLS and sent as XEP proposal to the XMPP editor in February 2009. Before Jingle-XTLS got accepted as XEP by the XMPP council, it was suggested to publish it as Internet Draft instead. Together with XML streams as Jingle application, the end-to-end security layer is specified as Internet Draft *Extensible Messaging and Presence Protocol (XMPP) End-to-End Encryption Using Transport Layer Security ('XTLS')* [MSA09a]. Example 6.4 depicts a session-initiate for client-to-client XML streams using In-Band Bytestreams and TLS security.[2]

The specification defines a method whereby any XMPP entity that supports the Jingle negotiation framework can use TLS for end-to-end encryption. The basic use case is to tunnel XMPP stanzas between two users for end-to-end secure communication using end-to-end XML streams. However, XTLS can be used by XPMN profiles for secure media transport as well;

[2]The example does not include child elements of the security element required by the Internet Draft. The child elements are required for bootstrapping the end-to-end security which will be discussed in section 6.2.4.

the Jingle-based file transfer already benefits from additional security without even updating its specification.

6.1.5 Application Layer Multicast

Similar to Mbus, XMPP provides application layer multicast. The simplest form is the usage of chat rooms (XEP-0045: *Multi User Chat*, MUC) to coordinate several clients. All clients join a specific chat room and a message send to the room is automatically distributed to all clients. The idea of applications communicating this way is implemented in the *Telepathy* framework used by the *One Laptop Per Child* (OLPC) initiative. Telepathy maps the D-Bus IPC system to XMPP. The D-Bus request/response communication is sent directly between clients using an In-Band Bytestream and events are distributed to all clients by each client connecting an In-Band Bytestream to a chat room. The concept is called *XMPP Tubes*, respectively *XMPP D-Bus Tubes*.[3] The second form of application layer multicast is realized by the Publish/Subscribe extension. A client sends a new item to the server and the server distributes it to all clients interested.

Unfortunately, the TLS-based security solution does not work with application layer multicast. The TLS channel is always between two entities; it is not possible to create a TLS chunk that can be distributed to more than one peer. This is a limitation of TLS and peer-to-peer protocols relying on TLS such as JXTA also have no secure multicast mechanism.

For a secure MUC-based multicast the node providing the chat room must be trusted. If all clients open a secure link to the room, it can decrypt all messages and encrypt them individually for all recipients. In most cases the MUC service is provided by the XMPP server itself; the same server we want to prevent from manipulating the messages with the end-to-end security layer in the first place. The obvious solution is a trusted client operated by the user to provide the MUC; though, this requires a user device to be always available—a requirement the XPMN core architecture should not have.

The challenge of multicast security is not limited to XMPP and is the focus of many research activities. The group members must agree on a symmetric *group key* to avoid the expensive asymmetric encryption. The clients must start a re-keying process every time a new member is added to the group or one is removed. This is required to ensure that the new device cannot decrypt older messages and, more importantly, that a removed device is excluded from the group. Rafaeli and Hutchison did a survey on various group key management solutions [RH03]. The simplest form is having a special client responsible for key distribution. Again, such a special device does not exists in the XPMN architecture. Distributed key management algorithms such as *Group Diffie-Hellman Key Exchange* [STW96] do not require such a special device but add additional complexity to the protocol.

[3] As of December 2008, *Tubes over XMPP* is not officially published as XEP. The current version of the XEP proposal can be found at `http://telepathy.freedesktop.org/xmpp/tubes.html`.

However, even with a group key known by all members, we only get confidentiality. While it may be simple to update the protocols to include data integrity as well, peer entity authentication is not easy to achieve [CGI+99]. A single key shared among all group members cannot be used to authenticate the sender. This may not be a problem in a multicast conversation between clients of the same user, but gets important in the XPMN interconnection use case where clients from different users interact. Signing the messages with the sender's private key provides source authentication with the costs of asymmetric encryption to sign and verify messages.

To summarize, it may be possible to use the application layer multicast provided by chat rooms and pubsub nodes for some tasks, but right now we only have unicast support. The XPMN core architecture provides no secure multicast mechanism.

6.2 Device Management

The TLS-based end-to-end security solution requires all devices to have a public/private key pair and a corresponding X.509 certificate for authentication. In the previous section we assumed that a device has somehow access to other devices' certificates.

This section introduces the usage of the X.509 certificates to log in to the server, the device and service discovery, and methods to distribute the list of devices and their certificates to other devices. The certificate distribution must be developed keeping the usability in mind—the obvious choice, copying certificates between clients, does not seem to be very user-friendly.

6.2.1 Client Certificates for Server Login

An XMPP client typically needs a user name and a password to log in to an XMPP server. Many clients provide a mechanism to store these credentials so they can automatically log in without prompting the user for the password. While this practice is very user-friendly, it can be a security risk: mobile devices such as a mobile phone or a laptop might get stolen, providing the thief with the required password. Mobile phones are particularly problematic since entering the password on the keypad for each login is complicated, and the risk of losing the phone is high compared to other XPMN devices.

A solution to this problem is to enable a client to log in without knowing the password. XMPP allows the use of any SASL mechanism, including SASL EXTERNAL (see XEP-0178: Best Practices for Use of SASL EXTERNAL with Certificates [SAM07]) and the server can request a client certificate during the TLS handshake. Since all clients already have a unique asymmetric key pair with an X.509 certificate, this certificate should be used for SASL EXTERNAL. A client does not need to know the account password anymore and logs in using its certificate instead. It can easily be removed from the media network in case it gets compromised because the certificate is only used by one client.

After the client first logged in using the password provided by the user, it uploads its certificate to the XMPP server. The certificate is uploaded in DER encoding, Base64-encoded for sending over the XML stream. Additionally, the client provides a human-readable name for the certificate, thus making it easier for a user to manage multiple certificates. Example 6.5 shows an certificate upload for a mobile client.

```
<iq from='hamlet@example.com/denmark'
    id='hfg65sw'
    type='set'>
  <upload xmlns='urn:xmpp:saslcert:0'>
    <item>
      <name>Mobile Client</name>
      <certificate>
        MIICCTCCAXKgAwIBAgIJALhU0Id6xxwQMA0GCSqGSIb3DQEBBQUAMA4xDDAKBgNV
        BAMTA2ZvbzAeFw0wNzEyMjgyMDA1MTRaFw0wODEyMjcyMDA1MTRaMA4xDDAKBgNV
        BAMTA2ZvbzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0DPcfeJzKWLGE22p
        RMINLKr+CxqozF14DqkXkLUwGzTqYRi49yK6aebZ9ssFspTTjqa2uNpw1U32748t
        qU6bpACWHbcC+eZ/hm5KymXBhL3Vjfb/dW0xrtxjI9JRFgrgWAyxndlNZUpN2s3D
        hKDfVgpPSx/Zp8d/ubbARxqZZZkCAwEAAaNvMG0wHQYDVR0OBBYEFJWwFqmSRGcx
        YXmQfdF+XBWkeML4MD4GA1UdIwQ3MDWAFJWwFqmSRGcxYXmQfdF+XBWkeML4oRKk
        EDAOMQwwCgYDVQQDEwNmb2+CCQC4VNCHesccEDAMBgNVHRMEBTADAQH/MA0GCSqG
        SIb3DQEBBQUAA4GBAIhlUeGZ0d0msNVxYWAXg2lRsJt9INHJQTCJMmoUeTtaRjyp
        ffJtuopguNNBDn+MjrEp2/+zLNMahDYLXaTVmBf6zvY0hzB9Ih0kNTh23Fb5j+yK
        QChPXQUo0EGCaODWhfhKRNdseUozfNWOz9iTgMGw8eYNLllQRL//iAOfOr/8
      </certificate>
    </item>
  </upload>
</iq>
```

Example 6.5: Server Certificate Upload

If the client attempts to upload and authorize a certificate for subsequent login attempts, it must protect the client-to-server stream using channel encryption via TLS and SASL. Otherwise, the certificate can be replaced during upload in an active attack by a certificate an attacker possesses the private key of.

Once the server has accepted the certificate, a client can use that certificate to authenticate using SASL EXTERNAL on subsequent logins. Therefore, the client does not need to store the password. The updated login process comes down to the following steps:

1. The server requests a client certificate during the TLS handshake. A client not supporting logging in with a certificate will ignore this request. It that case the login procedure will rely on the client providing a user name and a password.
2. The server advertises the SASL mechanisms after the TLS handshake and the stream restart. It advertises the SASL EXTERNAL mechanism if it expects that the client will be able to authenticate as the identity provided in the presented certificate.
3. The client chooses SASL EXTERNAL as preferred SASL mechanism. The certificate either contains the JID in the subjectAltName or the client has to provide it as authorization identity. A certificate may include a full JID in the subjectAltName. In that case,

the client is forced to use the resource name from the certificate. Another client already logged in with that resource name is automatically logged out.

Besides uploading certificates, clients can request the list of all certificates that are authorized to authenticate for its bare JID. The server returns the list of all known certificates, including the provided name and a server-generated unique identifier.

A client may create a new certificate before its current one expires. After the new certificate is uploaded to the server, it might want to remove the old certificate to keep the list of certificates short. Otherwise the list will grow indefinitely, making certificate handling more difficult for the user. Once a certificate has been removed it can no longer be used for SASL EXTERNAL. Besides removing a device from the list, the user can also revoke a certificate for a stolen or compromised device. The mechanism is similar to removing a certificate. The only difference is that if a client is logged in with a compromised certificate using SASL EXTERNAL, the server closes the stream to that client, thus forcing that client to log out.

The client uploading the certificate does not need to be the client that will use it, for instance, a user might use a chat client to upload the certificate for a TV set. In this scenario the TV set never needs to know the password and only needs to transmit its certificate to the chat client.

In February 2009, this process was specified in *Client Certificate Management for SASL EXTERNAL* (XEP-0257) [Mey09a] and approved by the XMPP council. During the preparation for the XMPP BOF at the IETF 74 it was decided to include it in the standardization process at the IETF. The specification was merged with XEP-0178 (Best Practices for Use of SASL EXTERNAL with Certificates) and published as Internet Draft *Management and Use of Client Certificates for the Extensible Messaging and Presence Protocol (XMPP)* [MSA09b].

6.2.2 Device and Service Discovery

The discovery requirements in section 2.2.6 distinguish between the device and the service discovery. The device discovery for XMPP can be realized with the presence and contact list subsystem (roster) of the XMPP instant messaging extension defined in RFC 3921. Each client sends its presence status to the server; this status is set on a per-client basis. The server distributes the presence information to all other clients of the same user and all friends in the roster with subscription status ‘to’ or ‘both’. The friends’ servers relay the information to the actual clients. A client not sending its presence status does not receive presence information from others—it does not exist for the media network. Presence can be used for the device discovery to find other parts of the personal media network and even friends’ devices.

Section 3.2.1 suggests the usage of mDNS and DNS-SD for service discovery in the home network to create a local group of devices which can interact even without an outgoing Internet connection. XEP-0174 defines sharing presence information using Zeroconf technologies. Similar to the presence information relayed by the server, this technique provides only device discovery and no service discovery.

A remote device is queried about its supported services once it is discovered. XEP-0030 (Service Discovery) [HMESA08] defines a request/response mechanism to query a remote node for the supported features (extension namespaces). After the number of XMPP extensions increased, XEP-0030 was written to determine what XEPs a peer supports.[4] Today, XEP-0030 is implemented by most XMPP client and libraries. This mechanism can be adapted to the XPMN architecture where service providers announce support for a feature (profile); controllers are passive and do not announce their support. Example 6.6 shows a controller (mobile phone) querying a service provider (VCR) of the same user about its capabilities.

```
<iq type='get'
    from='hamlet@shakespeare.lit/phone'
    to='hamlet@shakespeare.lit/vcr'
    id='hrt97htd'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>

<iq type='result'
    from='hamlet@shakespeare.lit/vcr'
    to='hamlet@shakespeare.lit/phone'
    id='hrt97htd'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity
        category='automation'
        name='VCR'/>
    <feature var='http://jabber.org/protocol/disco#info'/>
    <feature var='http://jabber.org/protocol/disco#items'/>
    <feature var='urn:xmpp:media:recorder:0'/>
    <feature var='urn:xmpp:media:server:0'/>
  </query>
</iq>
```

Example 6.6: XMPP Service Discovery

The XMPP Registrar currently manages the list of XEP-0030 features from the XMPP URN namespace. "Because the XMPP Standards Foundation publishes a relatively large number of protocol specifications, it is important to keep track of the namespaces defined by those specifications as well as the parameters used in the context of the relevant protocols. [...] It is the role of the XMPP Registrar to make those unique assignments and to maintain registries of the currently assigned values." (XEP-0053: XMPP Registrar Function). For this reason, the XMPP Registrar is a logical choice to administer the list of XPMN profiles.

As predicted in requirement 17 the service discovery stanza exchange results in a lot of traffic. Once an entity publishes its presence information, all other clients send a service discovery IQ stanza and the client performs a service discovery on all the others. Most devices have a constant list of services they provide; yet, it is possible that the service list changes over time. This problem is addressed in XEP-0115 (Entity Capabilities) [HSATK08]. The list of features

[4]Originally XEP-0011 (Jabber Browsing) was specified for this purpose, but had some limitations. XEP-0011 is superseded by XEP-0030 and now obsolete.

from XEP-0030 is reduced to a small capability string using a hash function. This string is distributed to other clients in the presence information, either in the presence stanza or the TXT record for link-local communication. If a client does not recognize the hash string, it requests the features using XEP-0030 and associates the result with the hash. If the features do not change, the client will always use the same capability string; different clients with the same capabilities may even use the same capability string. This reduces the XEP-0030 service discovery queries massively. After a while, all clients can determine the services of possible peers through their presence information—even if some service providers turn some service on or off sometimes.

6.2.3 Certificate Management

In 2007, Paterson wrote a specification for *Public Key Publishing* (XEP-0189) [PSAM09]. Originally, the XEP described an X.509 certificate or OpenPGP key exchange based on the *XML Signature Syntax and Processing* schema. It provided a mechanism to request a public key from another entity, to send a public key to someone, and to manage public keys using pubsub. The extension required the user to trust the pubsub server not to alter the information. In July 2009, the specification was marked experimental and there was no working implementation.

XEP-0189 was updated to the XPMN requirements within the scope of this thesis. The first adjustment was to remove the XML Signature Syntax and Processing schema because it is too complicated and the complexity is not needed for this use case. The schema may be one reason why the extension was never being implemented. Instead, the <keyinfo> element is now based on the output most cryptographic libraries support: the X.509 certificate in DER encoding or the OpenPGP key. The data is Base64 encoded to be included in an XML stream. This can easily be implemented with most TLS libraries without a greater understanding about the internal structure of X.509 certificates.

The second change is to allow one or more signatures for a public key inside the <keyinfo> element. Section 5.4.2 introduced the concept of user and client certificates with signatures outside the X.509 certificate chain. The signature has issuer information and the signature data. The issuer element contains the fingerprint of the key that was used to create the signature. While OpenPGP defines how to sign a string, X.509 does not specify a hash algorithm. For X.509, the signature data contains an additional attribute which hash and sign algorithms were used. To make it easier to use standard cryptographic libraries, the signature is generated based on PKCS#1 section 9.2 (Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1, RFC 3447) [JK03]. In most cases the cryptographic library will automatically take care of this. The data to sign is the X.509 certificate in DER encoding or the OpenPGP binary string of the public key. This equates to the key data without the Base64 encoding. Example 6.7 shows an X.509 certificate with an external signature referring to the signers certificate.

```
<keyinfo xmlns='urn:xmpp:tmp:pubkey'>
  <x509cert>
    MIICXDCCAcWgAwIBAgIJAKBfLqul2lj3MA0GCSqGSIb3DQEBBQUAMCkxJzAlBgNV
    BAMUHmRtZXllckBqYWJiZXIuY29tXDJmdGVzdGNsaWVudDAeFw0wODA5MDYxOTI0
    MjVaFw0wOTA5MDYxOTI0MjVaMCkxJzAlBgNVBAMUHmRtZXllckBqYWJiZXIuY29t
    XDJmdGVzdGNsaWVudDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAwaRLyj7J
    /mmliYhjEwGnRGRs6gmcPaIywEK2QLFz6c3/RmRabYbIOE0iZ22D33TguSNQBWfd
    lweT3bBETUhd3yuCcqWO5Ptiq/6wulMlxVeV5mxwNP/IF94VPWj0jHbRJcU8ZhS4
    UnX6R5q6OSfBGdUU4mYKdiaHpgqTAO9eeqUCAwEAAaOBizCBiDAdBgNVHQ4EFgQU
    b8touIdFuXF5clv2I/S1aOOFdN4wWQYDVR0jBFIwUIAUb8touIdFuXF5clv2I/S1
    aOOFdN6hLaQrMCkxJzAlBgNVBAMUHmRtZXllckBqYWJiZXIuY29tXDJmdGVzdGNs
    aWVudIIJAKBfLqul2lj3MAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQEFBQADgYEA
    pA5tI1J9Qpn3jSoQctFksRLb2H3A48R3rU8/qnarwE/AyOvth3k3ulLEmhJBT+0S
    mVb6WzrZEA/2plu7DhR8ylhuvJv6cAEIN+TPha3yzO2P8uoVZf7hdunOhMLl2Z6w
    xEfiGI5X9OsaMeFOQa+B2C3uUVAMLbVV7Rp/qQkai1Y=
  </x509cert>
  <signature>
    <issuer algo='sha1'>428b1358a286430f628da23fb33ddaf6e474f5c5</issuer>
    <value method='RSA-SHA1'>
      E3q/UkjRR3zcZMcIIoE2sSVKUATl26zyzO1Pmoe96p8apW91c3a0KqkQp1ZMBqXX
      +e2ImqQ79CKv+9qzXitxx+V4EcniKN0ZsSR+9ZbfflxkOvmBa2rpq9hFE1NYyfuT
      fsAZkRhAGlP7P5ELcvhqJ4WL6qBPYQU2NEnbVlcZSbA=
    </value>
  </signature>
</keyinfo>
```

Example 6.7: XMPP Certificate with Signature

XPMN has the following certificate policy:

1. Each user has one user certificate C_U and one or more clients can have access to the corresponding private key K_U to sign device certificates. One client generates the keys and the certificate on behalf of the user to bootstrap the system.

2. Each client has its own certificate C_D. This certificate may be self-signed. All device certificates are signed with the user private key outside the X.509 certificate chain inside the <keyinfo> element.

$$\begin{aligned} signature_{DU} &= Hash(C_U), \{Hash(C_D)\}_{K_U} \\ keyinfo_{DU} &= C_D, signature_{DU} \end{aligned}$$

The signature outside the X.509 certificate chain or the OpenPGP web-of-trust enables the user to sign an X.509 certificate with an OpenPGP key or the other way around. This allows verifying a self-signed X.509 user certificate with the OpenPGP web-of-trust; it allows the verification without setting up a CA.

For X.509 certificates to be used for authentication, all clients must have knowledge of others' client certificates signed by the user certificate. To keep the list of trusted devices somewhere without the devices asking each other, the public key information is published using pubsub. A previously published key can be updated by re-publishing the key using the same identifier.

The identifier is the fingerprint of the X.509 certificate or the OpenPGP key in lower case. Therefore, subscribers or other interested entities are able to request a single key by specifying its fingerprint. Since X.509 has no standard fingerprint mechanisms, the SHA-1 value in hex is used. SHA-1 may not be considered a very secure hash algorithms anymore, but it is only used to generate a unique identifier for the pubsub node and the strength of the algorithm is of no importance.

$$D \rightarrow pubsub \quad : \quad SHA1(C_D), keyinfo_{DU}$$

If a client wants to publish its own certificate or the certificate of a newly added client C_A, it needs access to the user's private key to sign it. As pointed out in section 5.3.4, a mobile phone is a device likely to be used to add a new device and also the device with the highest risk of getting compromised. Thus the mobile phone should not be in possession of the private key. To solve this, the client sends a sign-request to another client S having access to the user's private key. Being able to sign certificates is a XPMN service and announced by S using XEP-0030.

$$D \rightarrow S \quad : \quad C_A$$
$$S \rightarrow D \quad : \quad C_U, \{Hash(C_A)\}_{K_U}$$

The sign request must be done over an end-to-end secured TLS stream with mutual authentication to avoid the certificate to be signed being altered while it is transmitted to the signing party. This allows a mobile phone to add a new device without compromising the user's private key when the phone is stolen or lost.

6.2.4 Adding a Device to the Media Network

When a new device is added to the media network it needs the XPMN bare JID to create its certificate C_A. This certificate must be uploaded to the server to be used for SASL EXTERNAL, signed and uploaded to the pubsub server, and finally the device needs the user certificate to verify other devices in the network.

This is the first part of the security layer that directly involves the user; usability is important here. There has to be a user-friendly way to add a device without forcing the user to verify fingerprints and, if possible, a validation method the user is forced to perform and cannot bypass by acknowledging the new device without checking the end-to-end character of the connection.

This section goes into details how to add a new device using a one-time password, a procedure that is part of the end-to-end XTLS Internet Draft developed in section 6.1.4. Additional methods based on special hardware capabilities are outlined in section 6.2.5.

Password Authentication

The usage of passwords as TLS authentication mechanism is defined in RFC 5054 (Using the Secure Remote Password (SRP) Protocol for TLS Authentication) [TWMP07]. Originally de-

signed to authenticate the TLS client to the TLS server, the protocol can provide mutual authentication without certificates being involved. The TLS server may send its certificate but is not required to. One question that needs to be asked, however, is whether TLS-SRP is secure with very simple passwords such as a four-digit PIN. After all, there are only 10.000 possible password combinations. The protocol must be secure against brute-force and dictionary attacks for such limited number of passwords.

To answer this question we take a deeper look at the TLS-SRP handshake:

1. The first message is the *ClientHello* message from the client. It includes the information that SRP should be used and a user name for the authentication. The password-based authentication in our use case does not require a user name, any string is possible as long as both agree on it. A possible user name is the bare JID of the initiator.

2. The server answers with the *ServerKeyExchange* message. It contains the server's public value for the key exchange (B), the salt (s) the TLS client should use on the password, and prime and generator values (N, g).

3. The TLS client sends the *Finished* message in the third step of the handshake which is encrypted with the premaster secret. This is where the password comes in: based on the information the server sent in the second phase, the user name (I), the password (P), and the client's own public and private values (A, a), the client calculates the premaster secret (RFC 5054, section 2.6).

$$\begin{aligned} x &= SHA1(s \mid SHA1(I \mid \texttt{":"} \mid P)) \\ u &= SHA1(PAD(A) \mid PAD(B)) \\ k &= SHA1(N \mid PAD(g)) \\ premaster &= (B-(k*g^x))^{(a+(u*x))} \bmod N \end{aligned}$$

   The TLS server can verify whether or not the TLS client knew the password by checking the encrypted *Finished* message and calculating the premaster secret itself. This means that the TLS client has no way for brute force or dictionary attacks: it has only one chance of sending the correct *Finished* message based on the password and random information from the server—information it cannot influence or know preliminarily to the handshake.

4. The server also sends a *Finished* message encrypted with the premaster secret in the last phase of the handshake. At this point, the server can retrieve the password using a brute force attack on the previously received message. However, the server already proved that it is in possession of the password without revealing it in the second phase: the server's public value B is calculated based on the private value (b, random), the password, and the prime and generator values.

$$\begin{aligned} v &= g^x \bmod N \\ B &= k*v+g^b \bmod N \\ premaster &= (A*v^u)^b \bmod N \end{aligned}$$

The client can verify that the server already had the password in the second phase of the handshake in this last phase. The algorithm only works if B is calculated the correct way based on the password.

From this it follows that both the TLS client and the TLS server can not use brute force or a dictionary attack to retrieve the password; they can only guess and have a chance of 1:9.999 for four-digit numerical pins. Each client only has one chance to guess the password and the TLS handshake fails if the password is wrong; the peer will notice it right away.[5] Admittedly, if the client in the role of the TLS server does not know the password it can use a brute force attack during the handshake; this can be ignored since it is a one time password to bootstrap the trust relationship.

After the XMPP clients opened a TLS session using SRP they can securely exchange certificates for subsequent communications. The new device A must send its certificate C_A to the device already in the media network to be signed, added to the pubsub node, and sent to the server for certificate-based login. The new device must retrieve the user certificate C_U so it can verify other devices belonging to its user.

$$\begin{aligned} A \rightarrow D &: C_A \\ D \rightarrow server &: A, C_A \\ D \rightarrow pubsub &: SHA1(C_A), keyinfo_{AU} \\ D \rightarrow A &: C_U \end{aligned}$$

Choosing a TLS Method

With X.509 and SRP we now have two different methods for Transport Layer Security; both having advantages and limitations. X.509 can only be used if both devices are already in the media network and SRP can only be used if the user provides a password to the clients. The devices need a way to determine whether they must prompt the user for a password to be used within the SRP handshake, or if both clients can verify each others' certificates. In the worst case, an XPMN device does neither know the certificate of a new device nor is it capable of adding new devices due to the lack of a user interface. The clients should be able to detect this prior to the TLS handshake.

To determine which TLS method to use, a pre-TLS handshake is added to the Jingle negotiation in the <security> elements and in the STARTTLS feature for link-local communication. The initiator provides the fingerprint of the certificate it will use during the TLS negotiation and a list of supported TLS methods; currently X.509 certificate-based authentication and TLS-SRP. This information is exchanged over the insecure connection (server-based routing or link-local) and vulnerable to a denial of service attack. The purpose of the exchange is only to gather information what TLS method should be used in the TLS handshake; the actual authentication

[5] This analysis was confirmed in a private mail conversation with Thomas Wu, the author of SRP and one of the authors of TLS-SRP. He confirmed that TLS-SRP is indeed secure for short XPMN one time passwords.

is done during the TLS handshake. The clients may prompt the user for a password if one client cannot verify the certificate of its peer. Example 6.8 shows example 6.4 from section 6.1.4 with the additional negotiation elements.

```
<iq from='romeo@montague.lit/orchard'
    id='xn28s7gk'
    to='juliet@capulet.lit/balcony'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:0'>
          action='session-initiate'
          initiator='romeo@montague.lit/orchard'
          sid='a73sjjvkla37jfea'>
    <content creator='initiator' name='xmlstream'>
      <description xmlns='urn:xmpp:jingle:apps:xmlstream:0'/>
      <transport xmlns='urn:xmpp:jingle:transports:ibb:0'
                 block-size='4096'
                 sid='ch3d9s71'/>
      <security xmlns='urn:xmpp:jingle:security:xtls:0'>
        <fingerprint algo='sha1'>
          428b1358a286430f628da23fb33ddaf6e474f5c5
        </fingerprint>
        <method name='x509'/>
        <method name='srp'/>
      </security>
    </content>
  </jingle>
</iq>
```

Example 6.8: Initiate Jingle XTLS including TLS Method Negotiation

The recipient answers with its fingerprint and the methods it supports. It omits X.509 as possible method if it cannot verify the initiators fingerprint. After the recipient sent its supported methods, the initiator can determine whether X.509 certificate-based authentication will succeed or if TLS-SRP should be used. It sends a security-info message to the responder to signal its choice. This step is not really necessary because the responder will detect the initiator's choice in the first TLS message, though, it might help an implementation to set up the TLS library properly.

This pre-TLS negotiation was originally developed for Jingle XML Streams (see section 6.1.3) and link-local messaging and published as *XEP-0250: C2C Authentication Using TLS* [Mey08]. During the Jingle XTLS development the additional handshake was included in the Internet Draft; XEP-0250 is still required for link-local negotiation.

Protocol Flow for Adding a Device Link-Local

With the password-based peer authentication in place, a new device only needs a way to discover a device already in the media network to interact with. Since it has neither the bare JID nor the password it cannot log in to the server to discover one. However, the discovery process for

link-local messaging uses mDNS and DNS-SD; requiring no prior knowledge about the user's media network. A new device can discover other devices and the personal media networks they are in. Unless a device is part of a personal media network it is unable to announce itself due to the lack of a bare JID and only scans for other devices to start the pairing process. The complete protocol flow consists of five steps from XPMN discovery to logging in to the server:

1. The new device A detects a media network based on the bare JID of the service announcements of another device D. It generates an X.509 certificate with this JID as subjectAltName and tries to connect to other devices. Not every one of these devices has a user interface and can be used for the pairing process. In that case the pre-TLS negotiation fails and pairing with that particular device has to be aborted.

$$D \rightarrow A \quad : \quad JID$$

2. The user operating at least one other device approves the new device to be added to the media network. The devices open a secure communication based on TLS-SRP since they cannot verify each others certificates. The user has to provide the same PIN P to both devices. This step is the only one requiring user interaction.

$$\begin{aligned} A \rightarrow D \quad &: \quad TLS\text{-}SRP(P) \\ D \rightarrow A \quad &: \quad TLS\text{-}SRP(P) \end{aligned}$$

3. The device D already in the media network requests the device certificate from the newly added device A. It will either sign the certificate if it is in possession of the user's private key or requests a signature from a device S with the key. The certificate is uploaded to the pubsub server for end-to-end client communication, and to the XMPP server to be used with SASL EXTERNAL. The pubsub server sends the new certificate including its signature to all other available clients O. The new device is now integrated in the media network.

$$\begin{aligned} A \rightarrow D \quad &: \quad C_A \\ D \rightarrow server \quad &: \quad A, C_A \\ D \rightarrow S \quad &: \quad C_A \\ S \rightarrow D \quad &: \quad C_U, \{Hash(C_A)\}_{K_U} \\ D \rightarrow pubsub \quad &: \quad SHA1(C_A), keyinfo_{AU} \\ pubsub \rightarrow O \quad &: \quad SHA1(C_A), keyinfo_{AU} \end{aligned}$$

4. The new device receives the user's certificate so it can verify other devices of the user. At this point the new device also knows its certificate is signed and uploaded at both places.

$$D \rightarrow A \quad : \quad C_U$$

5. The new device logs in to the server. The bare JID provides the required information to determine which server and the certificate allows the client to log in using the SASL EX-

TERNAL. It queries the pubsub server for the certificate of others and sends its presence information to the XMPP server to become part of the media network.

$$\begin{aligned} A \rightarrow server &: C_A \\ A \rightarrow pubsub &: SHA1(C_O) \\ pubsub \rightarrow A &: SHA1(C_O), keyinfo_{OU} \end{aligned}$$

Admittedly, this is the simplest way since the new device is able to locate another one without any user interaction. If the devices cannot get in contact with each other they must exchange the credentials differently. The user must enter the bare JID, the password for the XMPP server login, and the SRP password for the pairing as described above. This is far less user-friendly. The next section discusses some alternative ways for the credentials exchange to increase the usability.

6.2.5 Usability Aspects for Device Pairing

The five steps in the link-local protocol flow can be split into creating a trust relationship on the communication channel (step 2), the credentials exchange (steps 1, 3 and 4), and the final login procedure (step 5). Being sure that the communication channel is secure depends on the channel itself; for link-local messaging we need to verify it using SRP since an attacker in the home network may impersonate the peer for both devices. This is the step that requires user interaction.

Suomalainen et al. compared the pairing techniques from *Bluetooth Secure Simple Pairing*, *Wi-Fi Protected Setup* (WPS), *Wireless USB Association Models*, and *HomePlugAV Protection Modes*. "Introducing a new device to a network or to another device is one of the most security critical phases of communication in personal networks. There have been several different proposals to make this process of associating devices both easy-to-use and secure. [...] In reality, it is impractical to mandate a single association model for all kinds of devices because different devices have different hardware capabilities. Also, different users and application contexts have different usability and security requirements" [SVA07]. The various association concepts can be applied to the XPMN device pairing.

The pairing process consists of a three-way handshake: the new device needs the user certificate for peer authentication and the bare JID to generate a certificate, the client certificate must be sent to the device already in the media network to integrate it, and thereafter the new device must be notified that it can now log in. Unlike some other pairing mechanisms we only care about data integrity. Confidentiality is not required since the devices only exchange certificates and the bare JID; these credentials are no secret and an attacker cannot gain access to the media network intercepting them. This simpler security requirement might make it easier to find a suitable pairing solution.

Leap of Faith

One approach to make the pairing process easier is to remove the password. The user has to press a pairing button on both devices in the home network and they locate each other using mDNS and exchange certificates. There is no authentication involved in this process; the user trusts that there is no attacker in the home network. This *leap of faith* provides the required security for subsequent communications, most importantly when used outside the home network later; it will detect attackers later unless the bootstrap process already was compromised. In that case an attacker as man-in-the-middle might have a device in the user's media network without the user knowing.

This can be made more secure by applying ideas from the *Push Button Configuration* used by Wi-Fi Protected Setup. Here, the user presses a button on both devices and afterwards the access point is open for pairing and the new device is probing for available access points. If one device sees multiple possible peers it aborts the operation with an error. A mechanism like this can detect man-in-the-middle attacks in the media network: either the new device sees two possible devices to pair with or two devices try to start the pairing process with the device already in the media network. The user will get the feedback that the pairing failed because there are more than two clients involved. This procedure is more secure than the leap of faith, but not as secure as a password-based pairing.

Out-of-Band Pairing

An already secure communication channel not associated with the media network itself can be used to exchange the credentials. For instance, if both devices have Bluetooth capabilities the handshake can be performed over Bluetooth with a special Bluetooth profile for the three steps of the credentials exchange. the clients have a secure link to exchange the credentials after the Bluetooth pairing process is complete. Even with the Bluetooth handshake completely different than TLS-SRP internally, it looks similar from the user's point of view. Using Bluetooth has the advantage that the Bluetooth device discovery mechanism is used instead of IP multicast. This allows a mobile phone connected over a 3G network to add a device close to it.

Bluetooth 2.1 introduced *Bluetooth Secure Simple Pairing* (SSP) as enhanced pairing mode. One new pairing method is the out-of-band model which is intended to be used with different out-of-band channels that can be used for XPMN as well without using the Bluetooth stack. The most interesting out-of-band channel from the usability point of view is the use of *Near Field Communication* (NFC) technology [Tec07].

The main characteristic of NFC is that the communication distance is limited to about 10cm. A mobile phone with NFC capabilities can start the pairing process when brought close to the device to be added. "There is no denying that a mobile phone is by far the most common ubiquitous computing platform available to the general public. Therefore, it is only reasonable that all the current Near Field Communication applications rely on it. However, it can be argued

that a mobile phone has its limits as a universal interaction device. [...] Besides, we really cannot expect a person to carry his mobile phone around the house at all times." [Rai07]

The user must manually set the phone into pairing mode to avoid adding every device that comes close to the mobile phone; a new device not in any media network is always in pairing mode. Unfortunately, NFC is not necessarily secure against attacks even with its limited range [HB06]. Based on the antenna used by an attacker, both passive and active attacks are possible. While eaves dropping is no security risk since we only require data integrity and no confidentiality, data corruption, data modification, and data insertion could be a serious thread. But even with these three attacks a man-in-the-middle attack is impossible because one of the client will detect the active attack (for details see section 3.5 in [HB06]). Thus, the active attack can only be used to perform a denial of service attack and not compromise the data integrity in the three-way handshake.

NFC defines active and passive devices; active devices generate their own RF field and are able to initiate the communication. The communication itself is always based on a message (M) from an active device and a reply (R) from an active or passive device. For the XPMN pairing handshake the active device is the one already associated; the new device can be passive, thus does not have to generate an RF field constantly while it is waiting for the pairing process. The following pairing handshake example uses a mobile phone to add a new device:

M1: The user initiates the pairing process on the mobile phone and it creates an RF field. It sends the XPMN bare JID to enable the new device to create its X.509 certificate accordingly.

R1: The new device returns its certificate with the bare JID as subjectAltName. The mobile phone sends the certificate to a signing device with the user's private key and waits until the device is added to the personal media network.

M2: The phone sends the user certificate and therewith, signals the new device now being part of the media network.

R2: The new device acknowledges and logs in to the server. Similar to link-local bootstrapping it fetches the roster and the client certificates from the pubsub node.

Besides NFC, other out-of-band channels are also possible for bootstrapping the trust relationship between a new device and the extended personal media network.

Unidirectional Out-of-Band Pairing

The aforementioned credentials exchange consists of three steps: the new device must retrieve the XPMN bare JID to generate a certificate, the new device must send that certificate to another device already part of the media network for signing and uploading, and it must retrieve the user certificate to verify other devices in the network. This means that the communication channel to exchange the credentials must be bidirectional. But it may also be possible that the device already in the media network generates the certificate and the associated private key on behalf of

the new device. After signing and uploading the certificate, it sends the new client its certificate and private key, the user certificate, and the bare JID.

$$D \rightarrow A \quad : \quad C_U, C_A, K_A$$

Reducing the exchange to this unidirectional step makes it possible to use a USB flash drive to add a new device to the media network. For instance, a laptop creates the credentials and puts them on the flash drive. Afterwards, the user carries the flash drive to the new device and plugs it in. The new device retrieves its new credentials and erases them from the flash drive. This solution is a fallback and will always work if the user has physical access to the new device. However, it is not as user-friendly as "touching" the new device with a mobile phone.

Additional Security Concerns

All the introduced methods for the pairing process share one weakness: the access to the device already in the media network. For usability reasons, it is nice for a user to simply bring the mobile phone close to a new device; for security reasons, the pairing process must be initiated manually or everyone that comes close to the user carrying the mobile phone around can add a device to the user's media network.

Based on the user's personal security requirements, the mobile phone should provide a way to secure the pairing process further by asking for a password. Some users will allow the simple pairing process while others may be afraid of the fact that an attacker only needs to have physical access to a device in the media network for a very short time to add a device.

Future devices could generate the password for the TLS-SRP peer authentication based on the user's fingerprint—touching two devices could initiate the pairing process. But currently available cheap fingerprint scanners are easy to fool. Again, it depends on the security level the individual user wants and there is no optimal solution. It is always a trade off between security and usability; there is no "best" solution for all users.

6.2.6 Initial XPMN Bootstrapping

Creating the media network by adding the first device is a special case. There is no other device to provide the required credentials, the account on the server may not even exist yet. The simple setup with a random user name and a random password from section 5.4.1 is not suitable since the user name may be required for XPMN interconnection and the password is the fallback to log in to the account to remove stolen devices. Consequently, the process of creating the media network requires user interaction by choosing a bare JID and a password.

In case the user already has an XMPP account for instant messaging, this account can be used for the media network as well; the IM client is the first device in the personal media network. Several surveys have revealed that IM is widely used, first and foremost by teenagers and young adults. "IM communications are mostly restricted to one's 'real space friends'—people who

first met face-to-face in physical space settings [...] Peer pressure helped to achieve a critical mass of users within a social group, which in turn sustained long-term use" [GP02]. The same applies when instant messaging is introduced into the work environment. De Vos et al. observed an increase from 11% to 48% of workers using instant messaging after a seminar of its usage at a company [dVHdP04]. "A central use of IM was to support quick questions and clarifications about ongoing work tasks. [...] IM was also used frequently for coordination and scheduling." [NWB00] A telephone survey of 1,000 Canadians between the ages of 16 and 54 conducted in April 2002 revealed that 80% of Internet users under 20 and 57% of the 20-34 group use instant messaging [Ran02]. Presumably, these numbers increased over the last years; besides, the first group of people under 20 is now in the second one between 20-34.

Unfortunately, most of these IM accounts are not XMPP accounts. Unlike e-mail, there is no common standard and besides the major players Skype, AOL (AOL Instant Messenger, ICQ), Yahoo (Yahoo Messenger), and Microsoft (Windows Messenger, MSN Messenger), social websites created their own closed infrastructure: MySpace and Facebook implemented their own instant messaging client based on web technology. On the bright side, Google Talk is based on XMPP and every Google Mail user already has a Google Talk account with the same user name. Besides Google, AOL seems to be experimenting with XMPP[6] which would allow all AIM users to bootstrap a media network without registering anywhere and Facebook is preparing the launch of an XMPP connection interface.[7]

If the user has no XMPP account, creating one is similar to setting up an e-mail account. Users are familiar with this procedure and an e-mail account may even be combined with an XMPP account such as Google Mail and Google Talk. With XMPP In-Band Registration it is also possible to create an account on first login. Surprisingly, the administrators of jabber.org got many e-mails asking where to register for an account, because the questioners simply did not expect that they could use an Internet-based service without registering on a website first. As of May 2009, jabber.org has a registration website.

A PC as the first client would be advantageous compare to a TV set with its limited input capabilities. An XMPP-based IM client can be used to bootstrap the media network even without supporting any XPMN profiles. As an alternative solution, a management client to bootstrap the media network and to manage the devices could be installed on a PC or a mobile phone.

6.3 External User

For the XPMN interconnection scenario described in section 2.1.5, a device does not only need to know what other devices exist in its own media network, it also needs to interact with devices

[6]There is no official statement from AOL. Some observations and an unofficial statement can be found at `http://florianjensen.com/2008/01/17/aol-adopting-xmpp-aka-jabber`. As of November 2009, AOL did not announce the service and the XMPP server formerly reachable at `xmpp.oscar.aol.com` is down.

[7]In May 2008, Facebook announced plans to support XMPP on their developer blog. As of November 2009, there is no official announcement; an XMPP is running at `chat.facebook.com`. More details can be found at `http://www.process-one.net/en/blogs/article/facebook_chat_supports_xmpp_with_ejabberd/`.

from friends. XMPP provides a list of friends in the roster. A client gets notified about all clients of all friends based on presence subscriptions. By adding XPMN interconnection we need to manage this list of friends and requirement 27 demands access control to services a friend is allowed to use.

6.3.1 User Management

The device management from the previous section requires all device certificates to be signed with the user's private key when stored on the pubsub server. Having the user certificate, it is possible to verify all clients belonging to that user. Hence we need a list of the friend's user certificates to realize the XPMN interconnection use case.

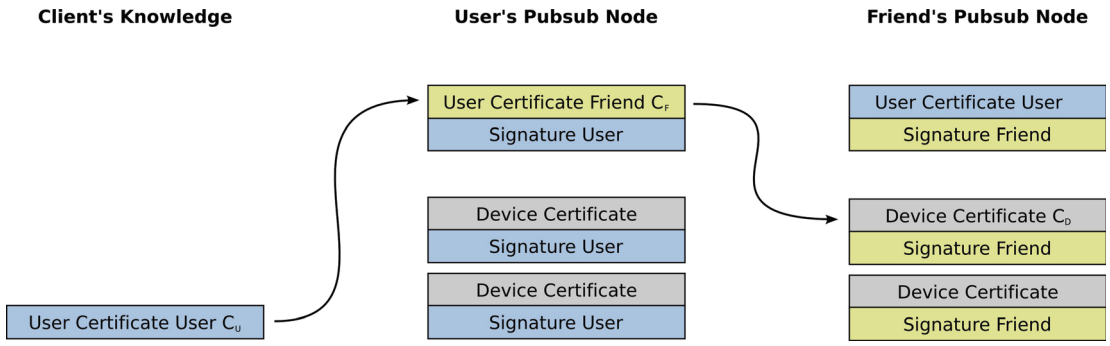

Figure 6.3: Certificate Verification Chain

Managing these certificates is similar to the device management: they have to be signed by the user certificate similar to a device certificate. If a client wants to open a secure end-to-end connection to a client D of another user, it checks its pubsub node for the friend's user certificate C_F and the friend's pubsub node for the client certificate C_D. The friend's user certificate must be signed by its own user private key ($\{Hash(C_F)\}_{K_U}$) and the client's certificate must be signed by the friend's user private key ($\{Hash(C_D)\}_{K_F}$). Figure 6.3 depicts how to verify the certificate of a friend's device.

$$\begin{aligned} keyinfo_{FU} &= C_F, (Hash(C_U), \{Hash(C_F)\}_{K_U}) \\ keyinfo_{DF} &= C_D, (Hash(C_F), \{Hash(C_D)\}_{K_F}) \end{aligned}$$

The process of bootstrapping the trust relationship between two humans is similar to adding a device: the two enter the same password on their devices for peer entity authentication. Afterwards, the clients open a Jingle XTLS session, use TLS-SRP to verify the end-to-end characteristic of the stream, exchange the user certificates, and each client signs the others' user certificate with its own and uploads it to its pubsub server. From now on, the friend's device can authenticate itself, but may have no authorization to access any services of the personal media network.

The password must be exchanged over a different communication channel such as e-mail, phone, or a meeting in person. Exchanging the password to establish a secure end-to-end com-

munication over an insecure chat session using the same XMPP stream does not provide any additional security. Using unencrypted e-mail for the exchange is more secure because there are other servers involved. Similar to adding a device, the credentials could be exchanged using Near Field Communication, an already authenticated Bluetooth link, or without additional security (leap of faith). Alternatively, the user certificate may be signed with an OpenPGP key allowing others to verify the certificate based on the OpenPGP web-of-trust. Friends can bootstrap a secure communication without any user interaction.

The read access to the pubsub node containing the friends' certificates can be limited to "owner" to hide the information about the trust relationships to others. A more open access module such as "roster" creates a small web-of-trust meaning that all friends in the roster can rely on the user having verified someone's certificate. If a user trusts the judgment and verification process of others, a client can accept new external user certificates based on signatures from trusted friends.

6.3.2 Access Control

Needless to say, a friend does not have the same permission in the media network; the access has to be limited. A reasonable default is to deny everything and use *access control rules* to permit friends access to some functions. An *access control list* (ACL) contains several access control rules. What to allow depends on the profile: a media server may provide access control to set read or write access to files or directories, and a TV tuner may provide access control to define a list channels the friend is permitted to tune to.

The same mechanism can be used to implement access restriction for children not of age to watch some movies (see the second use case in section 2.1.2). The child has a personal media network without any TV tuners or media servers and the parents add their child as external user, only allowing it to watch channels they choose to grant it access to. This provides only a very basic protection of minors; metadata embedded in movies or channels should provide further information such as FSK labels in Germany. Such a metadata-based finer grained automatic access control is outside the scope of this thesis.

Each rule includes the list of external users this rule applies to. There are only three categories to simply the configuration; the categories are based on the pubsub access model.

- The access module can be **open** for all users in the XMPP network. A possible use case for the open access is a directory of photos. The user can host photos on the media server at home similar to uploading them to Flickr.
- A more restricted access module is **roster**. The user defines a list of roster groups and every user in one of those groups has access to the service. This requires some trust in the XMPP server because a compromised XMPP server could modify the roster groups. Since the access requires end-to-end authentication, a compromised server can neither access the files directly nor can it add a new contact to the roster list who can. A false

friend in the roster has no corresponding signed user certificate in the XEP-0189 pubsub node.

- The third and most secure access is similar to the pubsub **white-listing**. The user has to define a list of bare JIDs and the rule only applies to those users.

In the XPMN interconnection use case described in section 2.1.5, the friend grants access to control the TV set. This access right may be limited to the evening the two meet and automatically removed later. Therefore, an access control rule may have a limited lifetime. The user certificate authentication is kept in the system for future use. A different kind of time constraint is used for the child in the second use case: its access to some parts of the system is limited to specific hours of a day, but has no general expiration time.

$$ACL \quad = \quad ([open, roster, whitelisting], times*, permission+)*$$

As a side effect of having the access rights defined by the profiles, a client can only configure services it has support for; the client must implement the access control rules of the service. If the user has a preferred configuration client on the PC, but the profile is only supported by the TV set controller, the inferior TV interface must be used.

Data Forms

The user could fill out a simple input mask to configure the access control of a service provider without the controller having support for the actual profile itself; it only needs to support a generic form specification. XEP-0004 (Data Forms) [EHM+07] specify how to send and fill out forms describing a very simple user interface with text fields, lists, and buttons. Supplementary, XEP-0068 (Field Standardization for Data Forms) defines a hidden FROM_TYPE element in the form to declare where the form is specified. If a device supports the profile it can fill out the form without user interaction or show a custom user interface more powerful than the actual form. The combination of data forms and the field standardization is used by several XMPP extensions such as pubsub, chat rooms, and in-band registration.

In the XPMN interconnection use case where the user allows the friend's TV set to access a specific file on the media server, the actual action the user performs is playing the file on a remote device. The mobile phone can automatically permits the friend's media network access to the file because it wants to use the friend's TV set to play it. The user does not need to see the form or interact with the access control system in any way directly.

The profiles should keep their data forms simple. It may be possible to have a very fine granulated access control, but this will make the user interface very complex. A profile such as media playback control may only have a Boolean field allowing to control the device; the TV tuner ACLs can be limited to select channels the external user is allowed to tune to.

A disadvantage of data forms is the limited set of fields. It may be sufficient for most profiles, but profiles such as the media server require complex access control management to allow

access to specific files or directories. Due to the limitations of data forms, each file or directory requires an extra form and will require the user to enter the file or directory name. This is not user-friendly and can be very confusing. Therefore, showing the data forms to the user can only be a fallback if the device has no support for the profile. A completely different user interface is to select a file while browsing the directory tree and permit or deny access to it. Microsoft Windows allows the user to right-click on an entry and modify the access control. A client could map this interface to the data forms if it is aware of the profile.

Internal Web Server

A different approach is to separate the automatic access control setting from the manual one. The profiles define access control elements in their own namespaces and an internal web server is used for manual configuration. While this allows complete configuration of any device without the limitations of data forms, it breaks with the consistent user interface. Each client has its own web server, thus its own style for web pages. Something similar to Web4CE (see section 4.1.4 for details) can be used to provide an interface for TV sets.

A new Jingle application must be defined to use HTTP over XMPP. Jingle is used for opening a transport between the two clients; with XTLS providing additional security if required. On top of it HTTP is used. Due to the characteristics of Jingle and In-Band Bytestreams as fallback, it is always possible to open a transport between two clients and only the bandwidth may be limited by the XMPP server.

HTTP 1.0 opens a new TCP connection for each element a client requests. While it is only a small overhead when HTTP is used over TCP, performing the complete Jingle negotiation will slow things down notably; the open transport should be reused. HTTP 1.1 defines a header field to signal that the connection can be reused for the next request. Here, a client only needs to perform one Jingle negotiation for a web page including all its elements.

One open issue is the HTTP URI: a host name and an optional port as authority part does not work. To define a valid URI, the full JID of the client must be encoded in the authority part. For the usage of HTTP as configuration interface this can be ignored since all references are on the same host and need no authority part.

After publication of a XEP proposal *Jingle HTTP*, some people in the XMPP community raised the legitimate concern that an HTTP server is another weak spot that can be used to compromise a system. For instance, insufficient path checking allows an attacker to access files outside the root directory of the server. An HTTP server can be quite complex and HTTP provides many features not required for configuration.

There is no final decision whether an integrated HTTP server is a good idea or not. For the XPMN architecture an integrated HTTP server is preferred over the limited control data forms provide. The risk of being compromised can be reduced to only allow clients of the same user access to the internal HTTP server and secure the Jingle transport with XTLS. The HTTP server

is protected from devices not belonging to the same user due to the fact that XTLS provides client authentication.

Alternatively, clients could exchange web pages without using HTTP. After all, XMPP already supports XHTML in messages, including external elements such as images. It is possible to define a simple IQ stanza to replace HTTP GET for a simple configuration interface. This solution requires a URI scheme to identify a resource on a remote XMPP client.

Independently of how the generic configuration interface will look like, it is clear that the limited feature set of data forms makes it unnecessary complex for the developers without providing additional benefit for the user; complex data forms are not user-friendly. Thus, the architecture requires profiles to define access control in their XML namespace and does not require a device to provide a generic access control configuration interface.

6.4 Service Provider

With the device management and the security layer in place, the architecture only lacks some guidelines how a controller and a service provider should interact. This is up to the specific profiles, although different profiles share the same basic principles. This section introduces some guidelines about controller and service provider communication. Some profiles will move away from these guidelines, but if possible, they should be followed to have a generic client/server-like architecture. This makes it easier for profile developers since it provides a familiar interface. This increases the usability for application developer which must not be forgotten since the end users are not the only ones with usability requirements for a wide deployment. An XMPP library with support for XPMN profiles can encapsulate a remote service in objects that makes controlling it similar to controlling the same service locally.

6.4.1 Commands

For the service provider to do something useful, a controller has to send a command the service provider should execute. In the UPnP terminology these commands are called actions, Mbus calls them remote procedure calls, and D-Bus calls them methods. They all mean the same: the controller triggers an action on the service provider and afterwards, the service provider returns a result (which can be empty). Some commands trigger no actions and do not modify the state of the device; instead, they are used to retrieve information about the internal status. Examples are receiving a directory listing from the media server, the current schedule on a TV tuner device, and metadata about the current playing item on a media player. These commands are similar to functions with the *const* or *final* keyword in languages such as C, C++ and Java. They should be mapped to IQ stanzas with the type *get*. Commands changing the state of the service provider or modifying internal variables should be expressed in IQ stanzas of the type *set*.

To summarize we have three kinds of commands:

1. query variables using IQ get,
2. set variables to a new value using IQ set, and
3. call an action using IQ set, similar to a function call.

When a variable changes its value, the service provider may send an event to some controllers to notify them about this change. How this can be done and which controllers get notified is discussed next.

6.4.2 Events

An event mechanism is important to avoid controllers having to poll the service providers for changes. Instead, the service providers notify the controllers. All protocols in the home network we looked at in chapter 3 have this kind of asynchronous message support and the lack of a subscribe/notify mechanism working outside the home network is one reason why we did not choose HTTP as basis for the XPMN core.

There are three different types of events a service provider can emit:

1. An event can signal the **status change** of the service provider, for example, the playback status of a media player. All possible controllers of the device may be interested in the new status.
2. A controller may want to get notified when a **variable** of the service provider changes, for example, the list of scheduled recordings on a TV tuner. An action is used to schedule a new recording or the variable itself is modified using an IQ set. The change to the list of recordings is signaled to all controllers monitoring this variable.
3. A **query result** may change over time and the notify mechanism can be used to signal updates. One possible use case is the media server: a controller may query the media server for music performed by a given artist and the resulting list may change if files are added or deleted. Unlike the previous use cases, only the controller who called this initial query receives the event notification about the changes; only it is able to understand its meaning. The changing result from a query is expressed by a *dynamic variable*.

A controller must be able to subscribe to specific events it is interested in. Depending on the event's type, this should be done without causing much traffic on device startup. The procedure looks similar to pubsub with the service providers being their own pubsub server—an end-to-end pubsub and personal eventing mechanism.

End-to-End Publish/Subscribe and Personal Eventing

Publish/Subscribe (Pubsub, XEP-0060, see section 4.4.2) defines a mechanism for a client to publish information to a pubsub node and to subscribe to such a node for notifications on

changes. Pubsub has an auto-subscribe feature based on Service Discovery (XEP-0030) and Entity Capabilities (XEP-0115) used by the Personal Eventing Protocol (PEP, XEP-0163, see section 4.4.3). Both extensions are already required by the XPMN core for device and service discovery. The entity capabilities can be used to signal interest to nodes from a specific XML namespace. This removes the need to subscribe to some variables such as the status changes.

The service providers' eventing mechanism is even simpler than PEP. PEP removes the need to subscribe to a node by using the service discovery and a client only needs to publish new information to the pubsub server. The virtual pubsub server on the XMPP server cannot be used since we require end-to-end secure XML streams. The service provider has to send the information to all (auto) subscribed peers itself. This removes the need to publish information and the publishing client takes the role of the pubsub server. While this makes the protocol simple, it increases the complexity of the client. A client has to include a small subset of the pubsub server specification. Most importantly, the service provider has to keep track of controllers it has to notify and, optionally, send the last pubsub message to a newly connected controller.

```
<message from='hamlet@denmark.lit/player'
         to='hamlet@denmark.lit/controller'
         id='hftr24sq'>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='urn:xmpp:media:player:status'>
      <item id='status'>
        <stopped/>
      </item>
    </items>
  </event>
</message>
```

Example 6.9: A Service Provider sending an Event

The three different types of events require distinct types of subscriptions. The basic status information should be sent to all possible controllers with auto-subscription. Example 6.9 shows the status change example of a media player. A controller can subscribe to these kind of events by adding `urn:xmpp:media:player:status+notify` to its list of features in the service discovery. Variable monitoring depends on the variable itself, for instance, the list of scheduled recordings may use the auto-subscription while the current volume of a media player may require an explicit subscription.

A third mode is a mixture of these two: controllers of the same user automatically subscribe while external users must explicit subscribe to the variable. The service provider must check its access control rules whether it should notify clients of other users. A reasonable default setting is to notify clients of the same user only. Profiles have to define an exception to this rule and the user can adjust these settings when configuring the access control (see section 6.3.2).

Dynamic subscriptions based on queries have a different subscription model. The subscription is part of the command and a dynamically created pubsub node is part of the result. Only the

controller calling the query is subscribed to the event. The dynamic pubsub node is automatically deleted when the subscriber unsubscribes from the node, or the subscriber or the service provider leaves the XMPP network.

Variables / Subscription Nodes

The variable names a controller can subscribe to depend on the profiles. Using pubsub's terminology, each variable is a pubsub node the controller can subscribe to. A profile must define the list of profile specific variables, including what kind of variable it is and how it should be handled:

- A **simple variable** has one value and changing it overwrites this value. Examples for such a simple form are the status notification or the current volume of a media player. This is consistent with the behavior of PEP.
- A **complex variable** is a list of values; a pubsub node with several items. Notifications do not contain a new value and describe changes to the list instead. This variable type takes advantage of the pubsub item identifier and the delete notification. The list of scheduled recordings is an example for such a variable: a notification includes the new or updated item and the delete event is emitted when a scheduled recording is removed.

The usage of the existing pubsub terminology and XML schema makes it more complicated than needed, but allows building upon existing and deployed solutions. The client-to-client publish/subscribe mechanism is much simpler than XEP-0060 or PEP. Actually, it does not publish items because this it is done internally. By that, we removed complexity from managing write access to the pubsub node, moderation, deletion, and configuration—configuration is a long and complex section in the pubsub specification, but here, it depends on the profile and, therefore, does not need to be changed by a controller. This leaves the IQ stanzas *subscribe* and *unsubscribe*, and the message stanzas *event* and *delete*. This is a very small and simple subset of XEP-0060.

6.5 Trusted XPMN Management Client

Some core functionality such as managing the list of friends and devices is provided by the XMPP server; even though the server is not fully trusted. To work around the lack of trust, certificates are signed by the users' private keys to prevent a compromised server from adding clients. The certificate publishing from XEP-0189 described in section 6.2.3 requires some rarely combined features from the pubsub specification. At the end of August 2009, no available XMPP server can be used because they lack support for virtual pubsub servers with permanent item storage. Their virtual pubsub servers only support PEP where a newly published item overwrites the current one; device certificates are a list of items.

Instead of using the XMPP server's pubsub component an XPMN client can be used for this task. This client should be always available and without it, new devices cannot be added and communication with friends' devices may be limited. A possible device for this task is the router in the user's home network. The router is a reasonable choice for devices in the home network: when the router is offline, the XMPP server is unreachable anyway.

We need the following to include the router as trusted XPMN management client—as trusted XPMN certificate distribution client:

- The router must be in possession of the user's private key and acts on signing requests. In fact, it should be the only client with this key to prevent other clients to add certificates while it is offline. After creation, a backup of the private key should be stored on a USB flash drive to avoid losing it when the router has a hardware failure. The private key stored outside the router should be protected by a password for security reasons. This password could be the same as the XMPP server password to make it easier for the user to remember. Only the server knows this password and it is unlikely that an attacker has access the user's home to steal the private key and compromise a server.[8]
- The pubsub server for device and friend management is located on the home router. The certificates are automatically signed on upload. Each client from the media network can publish a new certificate over an XTLS secured link. The router will sign the published certificate and add it to its pubsub node.

A client must be able to determine whether the client and friend certificates are stored on the server's pubsub node or on a trusted management device. However, this question does not come up in a real life scenario. If the router takes care of the certificate management it can erase existing information on the pubsub server and announce its service to the media network. Acting as trusted management client is a profile on top of the core all clients must support in the role of a controller. If a client providing this service is part of the media network, a controller knows it can upload a certificate to it using XTLS.

The pubsub notification can either be sent with or without using a secure XML stream. Without XTLS, it is as secure as if it is sent by the server's pubsub and the server must be trusted not to alter the message. All certificates are still signed with the user certificate making it impossible for the server to add a device to the personal media network.

Alternatively, the device can send the updates over an XTLS secured link. Unfortunately, this solution does not scale if the client has to open a secure connection to all its user's devices and all devices from all friends. There are two possible solutions to work around the problem. The first one is that the trusted management clients form a federated network among themselves and a client only notifies all clients of the same user and all management clients of the friends. These management clients must distribute that information to the corresponding clients of their

[8] Law enforcement agencies may have this capability. To make it harder, the XMPP server should be located in another country. This requires at least two law enforcement agencies to work together to compromise the system and it is still unknown to them if the password even is the same.

users. The second solution is not to notify devices from friends and require them to poll once they want to communicate with a device in another personal media network.

We assume that the router is accessible most of the time and if the router is offline, devices have knowledge of the last valid device list. Admittedly, if we require XTLS and do not notify friends' clients as suggested as second solution, they cannot check the device list if the router is offline. But if the router is offline, most of the user's devices are unreachable anyway for a friend outside the home network.

To simplify this problem, it comes down to two possible operations:

1. Send notifications over a secure link to the server, but publish without a security layer. This provides the same security as using the server's pubsub component and is independent from the server providing a pubsub service.
2. Open an XTLS secured stream to all clients of the same user and all management clients of friends. If a friend has no management client, notify all client of that user over the insecure link that there is a change and they have to open an XTLS session when they are interested in that information—when they actually want to communicate with a new device.

There is no single correct way to handle this and it depends on the security requirements of the users, the use cases, and how often devices change. A user with only a few friends in the roster or only a few actions the friends are allowed to call may not require to notify all friends. A user may only use the management client because of the server's pubsub limitations and may be fine without end-to-end security.

6.6 Summary: The Generic Device Architecture

The developed solution is more powerful than just an extended personal media network. It is a generic platform for devices to securely interact with each other – a generic device architecture based on XMPP. It provides a generic way to discover peers and their services, a secure way to communicate with other clients, and a unique identifier to interact with devices belonging to friends. Figure 6.4 provides a rough overview about how the various pieces described in this chapter interact with each other.

Looking back to the refined requirements from chapter 5 we have to check the three areas we defined to focus on: the networking layer, security, and usability considerations.

Networking Layer

The developed architecture allows two clients to communicate securely even with dynamic IP addresses and private networks behind a NAT. A client either locates its peer using multicast

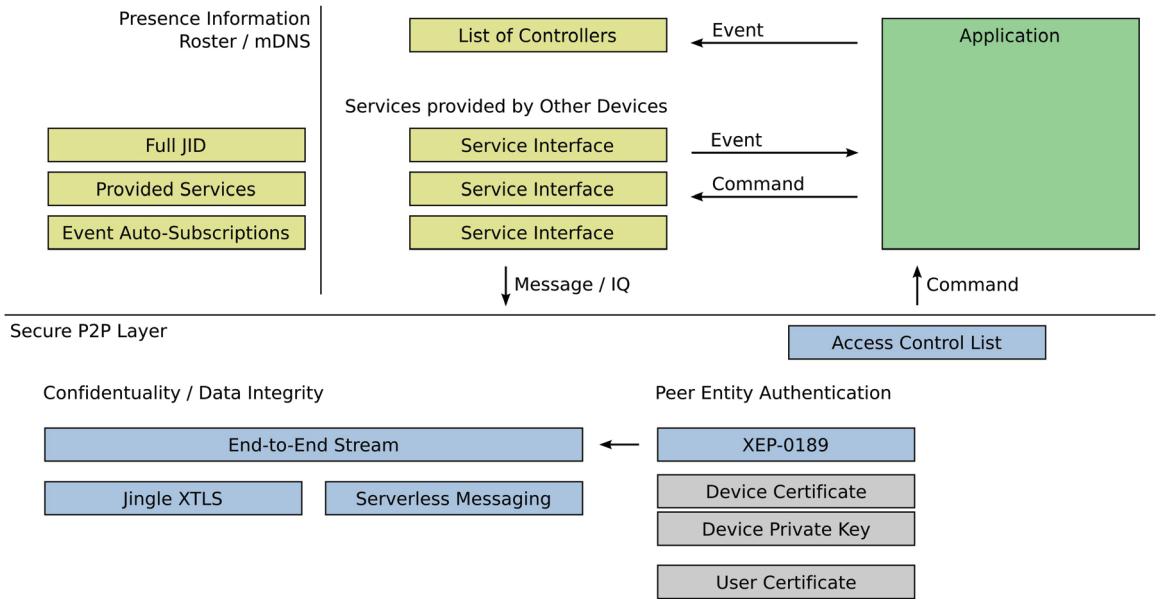

Figure 6.4: Generic Device Architecture

DNS or through a central XMPP server. Since the developed XTLS security layer works on top of In-Band Bytestreams, it does not have additional requirements to the networking layer.

The generic interaction between devices only covers commands and events which is the basic control operation. Profile specific requirements such as media transport is not covered. We will catch up on this in the next chapter defining profiles on top of this core.

Security

TLS provides peer authentication, confidentiality, and data integrity if a client can verify the X.509 certificate of its peer. The developed Jingle XTLS solution does not only guarantee a secure stanza exchange between peers, it can also be used for all kinds of streams set up by Jingle. The XTLS security layer can be reused by media transport profiles.

Using signatures outside the X.509 certificate chain is a simple solution to circumvent the complex and expensive X.509 signing process and allows including other existing technologies such as OpenPGP. Additionally, the user is free to let the user certificate be signed by a trusted CA; it is optional just like OpenPGP. Using pubsub to manage the certificates provides an easy way to revoke certificates.

The security layer is extended by access control lists to limit the access of friends' devices. The actual access control definitions are based on XPMN profiles on top of the core and outside the scope of this chapter.

But the solution is not without flaws: the XTLS negotiation and determining support for end-to-end security is done over an insecure link. An attacker could modify or drop the messages to prevent the clients from opening the secure Jingle session. Only the servers can manipulate the

data if the TCP connections to and between them are secured using TLS. This denial of service attack is a risk we are willing to take. The visible effect for the user is similar to a broken Internet connection. Most importantly, the private data—the media assets we want to protect—is not compromised. The media network itself already depends on a server operating correctly and this is only one denial of service attack a compromised server is able to perform: the XMPP server could stop routing messages, manipulate service discovery stanzas, and prevent rightful clients from logging in. As described in section 5.3, we only require confidentiality and data integrity for the profile specific stanzas and accept the denial of service as a possible attack.

A second threat is that the system becomes insecure once an attacker has access to both a stolen device in the media network and the XMPP server. The optional trusted management client implemented in the home network's router eliminates that problem, but the media network again is compromised if that device gets under the control of an attacker. If the media network is compromised the user has to recreate it, replacing the compromised router or choosing a new XMPP server—with an open, federated system such as XMPP this is an annoyance, but at least possible.

Usability Considerations

The user interaction required in the core architecture is the initial XPMN bootstrapping, adding a device, adding a friend, removing a device or friend if necessary, and access control. The normal operation does not require any user interaction: XTLS works based on certificates, pubsub can be used to even verify certificates from friends' devices, and the service discovery works completely without any user interaction.

Compared to UPnP device security, adding a new device only requires the user to enter the same PIN on both devices. This increases the usability in two ways. First of all, entering a user chosen PIN should be easier than comparing cryptic fingerprints. Secondly, the user cannot bypass the system and an error will not result in the wrong device being added: the correct PIN must be entered on both devices and a simple button click is not sufficient (ease of use: error tolerance, requirement 31, see section 2.2.9).

The architecture is open for future enhancements; here, the use of Bluetooth and NFC is already outlined in section 6.2.5. Furthermore, adding a friend is similar to adding a device and requires the same authentication steps. This makes it easier for the user to learn and understand the process. Needless to say that the pairing process must be explained to the user on the controlling interface. The user must be helped to understand why the security step is necessary to avoid the feeling of annoyance. This is outside the scope of this document and must be solved by interface designers.

The usability of the access control lists with data forms is unsatisfying. The restriction of the allowed types in data forms makes it very complicated for complex profiles such as a media server to define fine granulated access. The HTTP-based approach using Jingle makes more sense, but requires an HTTP server inside each service provider. This is also not satisfying, but it

provides access control for devices not knowing the specific profile. A web-based access control is not part of the core architecture; possible future profiles on top of the core are *Browser-based Configuration* and *Browser-based Service Control*.

Comparing to the Use Cases

In order to decide whether the developed architecture is indeed an extended personal media network as outlined by the use cases in chapter 2 we need to test it against them. This cannot be done by defining the architecture alone; we need profiles on top of it.

The following chapter defines the required profiles to execute the last and most complex use case from section 2.1.5: XPMN interconnection. Profiles are required to test whether the architecture is indeed powerful enough to fulfill the promise to deliver an extended personal media network.

Chapter 7

From Building Blocks to Profiles

The core developed in the previous chapter provides the basic functionality most profiles will require, but does nothing useful on its own without the actual profiles to define services. The requirements derived from the use cases described in section 2.1 are the reason why the core architecture is designed as it is; these use cases outline a wide range of profiles.

This chapter adds the profiles required to realize the XPMN interconnection use case from section 2.1.5. It is the most complex one by including two personal media networks with multiple user certificates and device listings, file transfer from the home network, and two service providers from different users controlled by the same controller. The profiles required to realize this scenario are *MediaServer Profiles* to search and access media files at home and *MediaControl Profiles* to control the media playback on the friend's TV set.

The profiles introduced in this chapter are only exemplary; the final versions need to be tested in several use cases, integrated in multiple home theater systems, and installed on wide range of devices ranging from mobile phones over low powered set-top boxes up to PC-based devices. This thesis only covers a first draft to fulfill the requirements from the use case for evaluation purposes only; one part of the evaluation is based on an exemplary implementation to test a scenario.

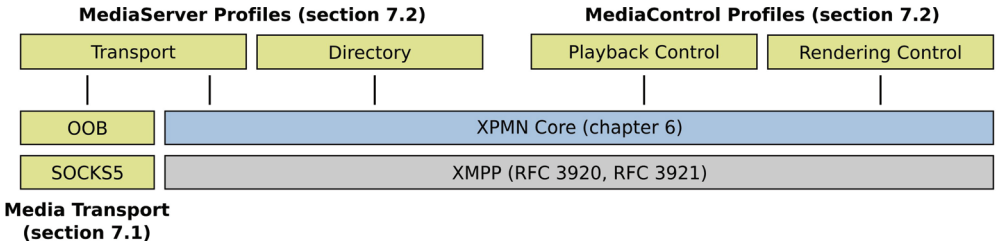

Figure 7.1: XPMN Profiles

Before realizing the MediaServer profiles, we first need to take a look at how to transport media data between devices; keeping private home networks behind a NAT and direct TCP connections in mind. Thereafter, the profiles are delineated and the chapter ends with an outlook on

possible future profiles. Figure 7.1 depicts an overview of the additional XMPP extensions (OOB, SOCKS5) and XPMN profiles described in this chapter.

7.1 Media Transport

The XPMN MediaServer profiles require a media transport between a service provider and a controller—more precisely: between a file server and the playing device. The secure end-to-end XML streams use In-Band Bytestreams as default transport layer. While this is a good solution for the XML control stanzas it is unsuitable for the actual media transport. The favored transport is a direct TCP connection between the two clients without any third party as relay server.

Section 5.2 outlined the need for a direct Jingle transport for media streaming and it was decided not to make it part of the XPMN core. This section catches up with this additional transport requirement and defines a new Jingle transfer, discusses the security of media streaming, and specifies a multiplex Jingle application.

7.1.1 TCP Streams in NAT Environments

As described in sections 4.3.2 and 4.3.3, some SIP clients use *Interactive Connectivity Establishment* (ICE) to open an end-to-end stream for voice and video calls. The ICE-TCP variant can be used to open end-to-end TCP streams (see section 4.3.3 for details about ICE-TCP). XEP-0176 adapts ICE-UDP to Jingle used by XMPP VoIP clients. When the XMPP community discussed ICE-TCP-based transport streams for applications such as *Jingle File Transfer* (XEP-0234) at the XMPP Summit's Jingle Interoperability event in Brussels in February 2009, it was consensus that ICE-TCP is not ready to be integrated:

1. ICE-TCP is very RTP-centric because it requires RFC 4571 as framing protocol. The XMPP community seeks a protocol that can be used for file transfer and end-to-end chat sessions as well. Not even SIP uses ICE-TCP for file transfer: "If only file transfer of SIP actually did 'work', there would be demand for TURN-TCP. But even MSRP, if it ever works, has gone a different way." (Denis-Courmont in a discussion about the status and future of ICE-TCP on the BEHAVE mailing list.[1])

2. The relayed candidates for ICE-TCP use TURN servers to relay the traffic between hosts; the TURN-TCP Internet Draft is in an early stage. In March 2009, Petit-Huguenin suggested to multiplex the TURN-TCP connections over one TCP connection to the TURN server [PH09] because this behavior matches TURN-UDP. If his changes get accepted, they will be a major protocol change. On a side note, in February 2009, TURN-TCP lost two of three authors because they did not have sufficient time to work on the draft.[2] In the

[1] The mail is archived at `http://www.ietf.org/mail-archive/web/behave/current/msg05454.html`.

[2] The announcement of the BEHAVE Working Group chairs that TURN-TCP needs a new primary author can be found at `http://www.ietf.org/mail-archive/web/behave/current/msg05447.html`.

mail announcing the need for a new primary author, the Working Group chairs raise the question whether or not the TURN-TCP development should be continued: “Is anyone planning to implement TURN-TCP client? TURN-TCP server? Or should we abandon TURN-TCP due to lack of need? [..] if you can volunteer to take over editing TURN-TCP, please drop the chairs an email. Or should we abandon TURN-TCP due to lack of editor?”. TURN-TCP got a new primary author, but implementations are still rare and there is no notable deployment of TURN servers supporting TCP.

3. ICE-TCP lacks support for Teredo and NAT assisting protocols such as UPnP IGD and NAT-PMP. An Internet Draft defining ICE-TCP as extensible framework [RL08] expired in April 2009.

While ICE-TCP may be a suitable solution in the future, it makes no sense to adopt ICE-TCP to XMPP in its current state. It was decided to build something similar to ICE-TCP based on deployed solutions with some ideas from ICE-TCP—but simpler. The goal is to create a Jingle transport that may not be as sufficient as ICE-TCP, although powerful enough for the typical use case with home networks behind a NAT.

XMPP file transfer not based on Jingle as defined by XEP-0096 can use several transport mechanisms. One transport suggested in the document is the usage of SOCKS5 Bytestreams as defined by XEP-0065. However, most clients use In-Band Bytestreams for file transfer, mainly because XEP-0096 defines no fallback if SOCKS5 stream negotiation fails. In a Jingle-based file transfer this drawback does not exist anymore: a client can replace a failed SOCKS5 Bytestream with an In-Band Bytestream.

As mentioned in section 6.1.4, XEP-0096 was revised within the scope of this thesis to be used as Jingle transport and published as *Jingle SOCKS5 Bytestreams Transport Method* (XEP-0260) [SAMKH09]. Besides including the SOCKS5 stream offers in the Jingle initiate and allowing the stream to be bidirectional, the specification includes some techniques from ICE to increase the chance of direct streams and the success of SOCKS5 Bytestream negotiation in general. XEP-0260 is influenced by ICE-UDP in Jingle RTP sessions and ICE-TCP:

1. A client can send several SOCKS5 candidates in the <transport> element to its peer similar to ICE-TCP candidates. This includes the host itself by opening a TCP port on all available interfaces the user wants to use (e.g. maybe not an expensive UMTS link). The host candidates are the IPv4 and IPv6 addresses of these interfaces (if available); additionally using an assisting NAT protocol if possible. If the client knows it is behind a NAT and the router announces UPnP IGD or NAT-PMP support, the client can map the open port to the external interface and include the public IP address and port number in its offer. Due to the fact that the peer includes the session identifier in the SOCKS5 connect, it is possible to initially map one port in the NAT and use it for all subsequent Jingle SOCKS5 negotiations.

2. The responder is also allowed to send candidates, increasing the chance of a successful connection. For example, the initiator might be behind a NAT without knowledge of a

proxy, whereas the responder might have a public IP address, might know about a proxy, or might have NAT traversal support such as NAT-PMP in its router.

3. A candidate has a priority to prevent one client from choosing a relayed connection before its peer had the chance to try all possible direct candidates. The priority can also be used to include a direct but expensive link as last resort and allows rating of relays based on their available bandwidth.

In example 7.1, Juliet's client provides candidates with its private IPv4 address, its public IPv6 address, the public address created by mapping the private IPv4 address/port using NAT-PMP, and a relay the client knows about. The direct host candidates equate to the list of active candidates in the ICE negotiation and the proxy is a relayed candidate.

A client should try the offered candidates sorted by their priorities and wait awhile before trying the next one.[3] The client must not wait for a TCP timeout. A successful connect to a candidate includes the SOCKS5 login and connect (`CMD = X'01'`, see XEP-0065 and RFC 1928 for details) to avoid choosing an invalid candidate. After all, a private IP address from one client might address a different host for the peer if the peer is in a different private network.

A client sends a candidate-error message using the Jingle transport-info action if it is unable to connect to any candidate. If no error occurs, both clients send a candidate-used with the identifiers of the first candidates they were able to connect to (candidate identifier, cid). The candidate with the highest priority will be used and the other connection is closed. In case both candidates have the same priority, for instance, when both clients were able to open a direct connection to each other, the candidate used by the initiator is chosen. As an optimization, a client can stop trying connecting to candidates if it gets a candidate-used from its peer which candidate has a higher priority than all its remaining candidates have. In that case the client can send a candidate-error immediately. If both clients send a candidate-error, the SOCKS5 negotiation has failed and the Jingle layer has to replace the transport with something else (see XEP-0260, section 3: *Fallback to IBB* for details). If the remaining transports are unsuitable for the application, for instance, the available bandwidth provided by an In-Band Bytestream is insufficient for media streaming, the initiator terminates the Jingle session.

These enhancements to SOCKS5 Bytestreams add most of the functionality of ICE-TCP to XMPP. It has host candidates (the network interfaces), NAT-PMP or UPnP to open a port similar to server-reflexive candidates, and relayed candidates using XMPP SOCKS5 relays. Moreover, a client can use Teredo to increase the change of a successful direct connection using IPv6. Teredo makes a direct connection possible by tunneling the data over UDP and many home network routers implement a cone NAT, allowing Teredo to send the UDP packets directly. Teredo is widely deployed and successfully used in the Microsoft Peer-to-Peer Network (see section 4.2.2).

ICE-TCP distinguishes between active and passive candidates where an active candidate opens a connection to a passive one. The passive candidates also exist in Jingle SOCKS5 Bytestreams

[3] The XEP suggest to wait 200ms; the number was randomly chosen to provide an actual number for developers.

```
<iq from='juliet@capulet.lit/balcony'
    to='romeo@montague.lit/orchard'
    id='hwd987h'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'>
          action='session-accept'
          initiator='romeo@montague.lit/orchard'
          sid='a73sjjvkla37jfea'>
    <content creator='initiator' name='stub'>
      <description xmlns='urn:xmpp:jingle:apps:stub:0'/>
      <transport xmlns='urn:xmpp:jingle:transports:s5b:1'
                 sid='vj3hs98y' mode='tcp'>
        <candidate cid='ht567dq'
                   host='192.169.1.10'
                   jid='juliet@capulet.lit/balcony'
                   port='6539'
                   priority='8257636'
                   type='direct'/>
        <candidate cid='hr65dqyd'
                   host='134.102.201.180'
                   jid='juliet@capulet.lit/balcony'
                   port='16453'
                   priority='7929855'
                   type='assisted'/>
        <candidate cid='grt654q2'
                   host='2001:638:708:30c9:219:d1ff:fea4:a17d'
                   jid='juliet@capulet.lit/balcony'
                   port='6539'
                   priority='8257606'
                   type='direct'/>
        <candidate cid='gft54deq'
                   host='relay.capulet.lit'
                   port='1265'
                   priority='720889'
                   type='proxy'/>
      </transport>
    </content>
  </jingle>
</iq>
```

Example 7.1: Jingle SOCKS5 Bytestream Transport

but are hidden in the application itself; they are neither exposed to the peer nor used for calculating the priorities.

Besides active and passive candidates, ICE-TCP supports symmetric candidates where the clients try traversing the NAT by opening a TCP connection to each other simultaneously. The solutions for symmetric candidates either depend on a specific NAT behavior, very good timing when opening a second TCP connection, or a modified TCP stack in the operating system. This candidate type has no counterpart in Jingle SOCKS5 Bytestreams. In ICE-TCP version 0.8 the authors added a note that "it has been reported that the simultaneous-open technique has a low success rate (˜40%) with the population of NAT devices in use as of this writing. Therefore,

it is RECOMMENDED that implementations of this specification acquire and use IPv6 host candidates." This fortifies the choice not to include the symmetric candidates in XEP-0260.

Even though XEP-0260 increases the change of the a direct connection, a relay host is required in some cases. The information about available relay hosts should be provided by the XMPP server using *External Service Discovery* (XEP-0215). For ICE-TCP, the server has to be responsible for providing information about available TURN servers. Additionally, an XPMN client may provide a SOCKS5 relay service to others. For instance, a SOCKS5 proxy installed on the home network router could be used by the user to relay traffic between two clients outside the home network (see section 6.5 about trusted XPMN management clients).

A Jingle transport based on ICE-TCP can be developed once ICE-TCP and TURN-TCP become more stable. Right now, the introduced transport based on SOCKS5 with some ideas from ICE provides a solid base which should be suitable for NAT traversal in must scenarios.

7.1.2 End-to-End Security

It should be possible to optionally secure the media data exchange against eavesdropping and modification similar to the XMPP control messages; the media transport needs confidentiality and data integrity as well. If the media data exchange is realized as Jingle application—and it should be since this is the use case Jingle was designed for—we get security for free without any additional specification. The XTLS security layer developed within the scope of this thesis is part of Jingle; thus, can be used by any Jingle application. When a MediaServer profile opens a stream to be used for media transport, it can simply add the XTLS security layer to the Jingle negotiation and the stream will be secured using TLS.

For UDP-based transports there is no corresponding security layer. If necessary, *Datagram Transport Layer Security* (DTLS) can be used by XTLS to secure UDP packets. While this should work in theory, it has never been tested and DTLS lacks SRP support. Although, the latter might not be a problem: it is likely that the certificates are already known due to a prior opened Jingle XML stream. Elsewise, it should be straightforward to specify DTLS-SRP-based on DTLS and TLS-SRP.

For media data sent using RTP, the *Secure Real-time Transport Protocol* (SRTP) variant of RTP can be used. SRTP requires the peers to exchange a key used for the encryption before the communication starts (see section 11.3 in XEP-0167: Jingle RTP Sessions). This key is exchanged inside the Jingle session negotiation. The RTP Jingle session can be negotiated using an end-to-end secured stream to protect the key from being exposed. In doing so, the key can be exchanged in plain text since the negotiated channel itself is encrypted.

7.1.3 Out-of-Band Stream Data

The media network uses XMPP stanzas as control messages to trigger actions on remote devices and to emit events. These stanzas are sent in an In-Band Bytestream secured by TLS while the

actual media data is sent over SOCKS5 Bytestreams. In most cases, the separation of control messages and content is simple: an action requesting a file is a control message while the returning file data itself is content.

For actions such as a MediaServer directory listing the situation is more complicated. Similar to the file access, the directory listing request is a control message and the answer is content. However, the answer is sent as IQ result inside the XML stream and by that, appears as part of the control channel. The key problem with this is that large directory listing may block the XML stream. XMPP has one XML stream for the communication and a client has to wait until one stanza has been sent before it can send the next one. This is no problem for small messages, yet a directory listing for hundreds of vacation images including their metadata can result in a very large XML stanza. If the XMPP server throttles the bandwidth per client and the end-to-end stream is based on In-Band Bytestreams, such an IQ result can block the end-to-end XML stream in the order of minutes—this is not acceptable.

Consequently large XML elements such as directory listings are content and should be sent outside the XML stream—sent out-of-band (OOB). Yet, opening a new Jingle session for every directory listing request produces a very high overhead: the clients have to negotiate the Jingle session, open the transport by trying several SOCKS5 candidates, and perform the TLS handshake for end-to-end security. To avoid this, a session established to exchange content data should be reused for subsequent requests.

```
stream          = *(chunk | last-chunk)
chunk           = chunk-size id CRLF chunk-data CRLF
hexdig-nonzero  = %x31-39 ; "1"-"9"
chunk-size      = hexdig-nonzero *HEXDIG
id              = *(ALPHA | DIGIT)
last-chunk      = 1*("0") id CRLF CRLF
```

Figure 7.2: Out-of-Band Stream Data ABNF

Yet, sending large stanzas using the same Jingle session only shifts the initial problem: the transmission of a large element blocks subsequent elements. To solve this, a Jingle OOB application must multiplex multiple elements in one transport stream instead of transmitting the elements sequentially. The elements are split into smaller chunks and the chunks from multiple elements are multiplexed in the negotiated transport layer. Each chunk has a content identifier to identify which element the chunk belongs to, a length field to provide the number of payload data bytes, and the payload data itself. The syntax is inspired by HTTP 1.1 chunked transfer enlarged by the content identifier HTTP does not need. The last chunk of each element has always a zero length to signal its end. Figure 7.2 shows the *Augmented Backus-Naur Form* of the stream data.

An <oob> element with the content identifier (id) attribute is returned in the XML stream at the position where the data sent over Jingle should be inserted by the XML parser. As a positive side-effect, this also allows a better way to transmit binary data between clients. Instead of

adding the binary data Base64 encoded in the XML stream, it is now possible to send the raw binary data out of band and include the <oob> element where the data belongs in the stream. The OOB data should be treated similar to Base64 encoded binary data in case it is no valid XML document. Example 7.2 shows an IQ result stanza with the complete result sent out of band in the multiplex channel with the content identifier 42. The size and hash arguments in the example are optional.

```
<iq from='hamlet@example.com/bot'
    to='hamlet@example.com/denmark'
    id='hfgte45w'
    type='result'>
  <oob xmlns='urn:xmpp:jingle:apps:oob:0'
       id='42'
       size='6022'
       hash='sha1+552da749930852c69ae5d2141d3766b1'
       type='text/xml'/>
</iq>
```

Example 7.2: Out-of-Band Result

OOB should not be used to replace any arbitrary part of any existing XMPP extension; an extension must explicitly allow parts of the XML or binary data transmitted over the Jingle OOB channel. This makes it easier for developers since they must not expect and synchronize OOB data in all modules and plug-ins—after all, OOB data can force a library to delay the stanza handling until all parts are received.

If the receiving client is not interested in the out of band data (anymore) it can abort the content sending to save bandwidth. This is only useful for larger binary content; the MediaServer profile defined later will make use of this feature for playback control. An XMPP extension requires explicit support for this IQ, because if the out of band delivery is aborted the surrounding stanza cannot be delivered from the XMPP library to the actual application—the XML stanza is invalid until the OOB data is completely received.

In April 2009, a first version of *Out-of-Band Stream Data* was published and approved by the XMPP council as XEP-0265 [Mey09b]. After publication, some members of the XMPP community suggested to use the framing format defined by BEEP (see section 4.2.2 for a small introduction to BEEP). The framing format will slightly increase the required bandwidth; other parts of BEEP should not be included. There is no consensus yet whether or not the framing format should be changed because OOB will violate the BEEP standard by only having frames of the type MSG.

A second alternative is to use the *Stream Control Transmission Protocol* (SCTP) which supports multiple streams inside one connection naively. Unfortunately, SCTP itself is not widely deployed yet. Likely, this will prevent a XEP requiring SCTP from being implemented in a large number of XMPP libraries and applications.

7.2 MediaServer Profiles

The use case to test the core specification in the evaluation includes a media server at home where the user searches for a specific file and plays the video on a remote TV set. This breaks down to two profiles: browsing and searching a media database and accessing files from another client. In a UPnP environment these two profiles are provided by a UPnP AVServer.

This section starts with discussing the *MediaServer Transfer* profile because its design influences the general complexity of the device. Afterwards, the *MediaServer Directory* profile to search for media assets is introduced and at the end, a brief outlook on integrating existing UPnP devices and control points is given.

7.2.1 MediaServer Transfer

UPnP requires all devices to support media access over HTTP. Furthermore, some UPnP devices support streaming audio and video files using RTP; images are always transmitted over HTTP. The RTP streaming is controlled using the *Real Time Streaming Protocol* (RTSP, RFC 2326) [SRL98]. RTSP allows to initiate, start, pause and stop an RTP stream. In section 5.2.1 we came to the conclusion to use a reliable media transport protocol but RTP is usually streamed over UDP. Nevertheless, it is possible to encapsulate RTSP and RTP in one TCP connection allowing their usage on top of a reliable transport. UPnP does not support RTP over TCP; presumably because packet loss is unlikely in the home network, thus not considered necessary.

Jingle RTSP/RTP and Jingle HTTP

Jingle RTP/RTSP is one possible solution to realize the media transfer on the basis of existing protocols. Unfortunately, both RTSP and RTP have substantial disadvantages when playing some video containers and codecs. First and foremost, the server must have support for the video container format the video is stored in. It has to demultiplex the video and audio streams and send them in separate RTP streams to the peer. Furthermore, every audio and video codec requires special support in the device to encapsulate the data into RTP packets and some codecs are not even supported by RTP. In consequence only the player needs to have support for the containers and the codecs—the server needs to support them as well. If a server does not support a specific container, it cannot demultiplex the streams, and if it does not support a codec, it cannot send it over RTP.

Secondly, the playback control of RTSP is limited to simple video and audio files or streams. Enhanced media content such as menus, chapter selection, and language and subtitle selection are unsupported. This makes it impossible to play a DVD or a BluRay disc remotely or switch between languages in a DVB recording. The *MMUSIC Working Group* at the IETF is working on the *Real Time Streaming Protocol 2.0* which is mostly finished, but it has the same limitations regarding the additional media features.

A better way is keeping the complete playback logic on the player and use the server just for the data storage. A controller simply requests the data chunks it requires. HTTP provides methods to download and upload files and HTTP 1.1 adds a range header to request chunks from a larger file to realize seeking in the file (see section 5.2.1); WebDav (RFC 4918) includes an extension to move a file. This is all the functionality required for the media transport.

Therefore, Jingle HTTP is a possible solution for the media transport, but similar to RTP/RTSP it has a serious drawback: it adds much complexity which is not needed for the media transfer. Jingle HTTP for an XMPP-based media server was discussed at the XMPP summit in Brussels in February 2009. Many developers felt that downloading files from another user is not worth including a complete HTTP stack in their application. Most of the functionality of HTTP 1.1 is not needed and WebDav adds even more complexity to the system (see section 6.3.2 for more details about the disadvantages of HTTP over Jingle).

Seeking and Video Index Handling using OOB

Since XMPP is available to exchange control messages, it makes sense to use it to trigger the download or upload of a complete file or parts of it. Besides that, the profile needs to define actions to rename, move and delete media assets. The media data itself can be transmitted using the Out-of-Band Stream Data extension described in section 7.1.3. In fact, the media transmission had some influence in the design of the OOB extension, for instance, the binary data handling in general and the ability to abort a transmission.

An important usability requirement is the ability to seek inside the stream without downloading the whole file first. A resume feature is unusable if the user has to wait several minutes until the video is downloaded to resume a playback close to the end. To allow correct seeking inside the stream some video container formats provide an index which is often stored at the end of the file. The reason for this is simple: when the file is created, the encoder or multiplex application does not know how long the file will be and how much space the index will require. Thus, the index is at the end of the file and its byte position is added at the beginning of the file after the multiplex process is complete. Some MPEG-4 files have the index at the beginning; this requires rewriting the whole file after the encoding process is done.

To seek to specific byte positions, the MediaServer Transfer profile allows to specify a range of bytes when requesting a media file. This technique is borrowed from HTTP with its semantic based on the HTTP 1.1 range header. Together with the ability to abort an OOB data stream—a media transfer in progress—this creates a powerful way to stream media formats normally unsuitable for streaming, does not require any special container or codec support from the server, and allows seeking and thereby playback resumption.

The protocol flow for video playback generally consists of the following three steps:

1. The player requests the complete file from the beginning to the end without providing a file range and the stream is sent out of band. This behavior is similar to current video players playing from web servers using HTTP.

2. On arrival of the first bytes the player determines the file format[4] and, if available, reads the byte position of the index. If an index is stored at the end of the file, the player requests the file a second time from the byte position of the index to its end. The first stream from the beginning will be downloaded parallel to the index.

3. The player is ready for playback after the index and the available metadata is received. When resuming playback, the player stops the first stream and starts a new download based on the byte position provided in the index.

Depending on its storage capabilities, the player may already have the stream data if it downloads the file as fast as possible or the user seeks backwards. In that case, it does not need to abort and set up a new transfer for seeking.

```
<iq from='hamlet@example.com/denmark'
    to='hamlet@example.com/fileserver'
    id='gftyert4'
    type='get'>
  <transfer xmlns='urn:xmpp:media:server:0'
            uid='video102'
            from='64537'/>
</iq>

<iq from='hamlet@example.com/fileserver'
    to='hamlet@example.com/denmark'
    id='gftyert4'
    type='result'>
  <oob xmlns='urn:xmpp:jingle:apps:out-of-band:0'
       id='103'/>
</iq>
```

Example 7.3: Requesting a File

Example 7.3 shows a client requesting a video file (unique identifier video102) starting from byte position 64537 up to the end of the file. The actual video data is transmitted with the OOB multiplex identifier 103 out of band. This special handling only makes sense for video and audio files; most images need to be fully downloaded to be displayed correctly.

URI Schema

The last piece missing is a *Uniform Resource Identifier* (URI) to uniquely identify a file on an XMPP client to pass to the player. The existing XMPP URI schema does not only describe a user (xmpp:user@domain) or a resource (xmpp:user@domain/resource), it may also contain an action. XEP-0147 (XMPP URI Scheme Query Components) defines some basic actions such as *message* to send a message to the user. XEP-0147 section 7 defines the XMPP Registrar as

[4] All container formats have a unique "magic number" at the beginning of the file to detect a file type independently of the file extension which could be wrong.

authority for registering new actions. Various XMPP extensions add actions based on their use case, for instance, Multi User Chat defines an action to join a chat room.

The media transfer can be defined as an action. This makes it possible to use an XMPP URI to describe the location of a media asset. Every media asset has a unique identifier that can be added to the URI parameter in the form *xmpp:user@domain/resource?transfer;uid=identifier*.

7.2.2 MediaServer Directory

The *MediaServer Directory* profile is inspired by the UPnP AVServer; more precisely the *ContentDirectory Service Template* [LR+08]. A UPnP media server allows a UPnP control point to browse the collection, move or delete files, upload a file by importing a URI, and search for content based on metadata attributes. The same functionality should be provided by the XPMN MediaServer Directory profile.

Metadata

A server able to parse metadata information from media files would allow the user to search for content based on attributes such as artist and album for audio files, location and date for image files, and locate all episodes of a TV show. Unlike streaming files with RTP, it is not critical for the media server to support a specific container. The file is still playable if the server cannot parse the metadata; the metadata is reduced to the file name and file size (see section 7.2.1 for a discussion about container and codec support). It is obvious that searching based on metadata attributes does not work for such a file. Yet, adding support for metadata parsing is much simpler than adding full support for the various codecs and containers; for instance, the Freevo metadata parser is encapsulated in a small library and supports over thirty different audio, video and image file formats.

Similar to UPnP, the metadata for each item is returned in search results or when browsing the directory structure. Many media center software solutions show supplementary attributes if they are available; for example, the Freevo music browser shows the title, artist, album and cover art if provided by the metadata parser. This improves the user experience when browsing the music collection. Alternative interfaces allow to browse the music collection only by looking at the cover art (e.g. Apple's cover-flow), browse a list of images based on thumbnails, or select a movie based on poster art.

Surprisingly, there is no simple XML schema for media metadata attributes. The *Dublin Core Metadata Element Set*[5] is a well-known set of metadata elements used in various other specifications such as UPnP. The elements include title, license, publisher, date and other generic elements independent of the media type. Information about artist, album, cover art, thumbnails, genre, GPS coordinates where images were taken, and other attributes a user may want to search the media database for are missing.

[5]The metadata set is defined by *The Dublin Core Metadata Initiative*; `http://dublincore.org/`

On the other side of the spectrum are complex specifications such as MPEG-7 (Multimedia Content Description Interface) and TV-Anytime[6]. The specifications can be used to describe a media object in detail, including shooting locations and content descriptions based on the time code in a video. Generally these information are not available in the media files directly and will make an XML metadata parser very complex. Both MPEG-7 and TV-Anytime are not integrated into consumer electronic devices; UPnP defines its own extended set of metadata elements.

Several Linux Open Source projects agreed on a set of metadata attributes for media databases and players.[7] Those are a mixture of Dublin Core attributes, basic file information (e.g. file name and size), ID3 track information for audio files, and EXIF and IPTC header fields for images. While the list may not be sufficient for all possible use cases, it provides a good starting point. Example 7.4 shows a metadata item element for the first audio track of the audio CD "Dark Passion Play" performed by the band Nightwish. The cover art is not included directly and referenced by a unique content id, accessible using XEP-0231 (Bits of Binary, BOB) [SAi08].

```
<item xmlns='urn:xmpp:media:metadata:0'>
  <title>The Poet And The Pendulum</title>
  <artist>Nightwish</artist>
  <album>Dark Passion Play</album>
  <release-date>2007</release-date>
  <duration>830</duration>
  <cover-album>
    cid:sha1+2478f866aeaa49ea96ef5bbcfecc94b5b57a6cba@bob.xmpp.org
  </cover-salbum>
</item>
```

Example 7.4: Audio Metadata

Browsing and Searching

The user should be able to browse the media database by directories as well as by metadata attributes such as artist or album. A typical user interface for browsing the music collection is to present the user a list of artists and after selecting an artist a list of all albums from that artist. Other media assets such as images may already be sorted in directories based on event (e.g. vacation or birthday) and alternatively, the user may browse images based on their capture date or, if available, by tags. Comparing metadata queries and directory listings, a directory listing is a query based on the metadata attribute *directory* the media file is stored in. Thus, the MediaServer Directory profile only needs to provide a generic search function based on metadata attributes.

[6]The TV-Anytime specifications are developed by the TV-Anytime Forum. The second version of the specifications was released in 2005; http://www.tv-anytime.org/

[7]The current version of the Freedesktop.org shared file metadata specification used by most indexing applications is available at http://wiki.freedesktop.org/wiki/Specifications/shared-filemetadata-spec.

The UPnP ContentDirectory specification uses the terms *item* and *container*; both are *objects*. An item is an actual media file, a track on a CD, or a remotely accessible stream. A container represents a collection of objects, for example, a directory with files (items) and sub-directories (containers) in it. A different kind of container is a collection based on search results. A container itself cannot be played directly and only items inside it may be playable; a container can be considered a playlist.

This definition is both simple and powerful enough to be used for queries on the XPMN MediaServer. The query result is always a container with metadata of the container itself and items or containers as child elements. Example 7.5 shows the query result for a directory containing two audio files both performed by the band Nightwish but from different albums. Since artist is a shared attribute from all sub-objects the container has artist as metadata. A container based on a directory query has a locally unique identifier (id) and a parent attribute (parent directory); other types of queries may not have a parent.

```
<container xmlns='urn:xmpp:media:metadata:0' id='dir20' parent='dir18'>
  <artist>Nightwish</artist>
  <item id='audio235' parent='dir20'>
    <title>The Poet And The Pendulum</title>
    <album>Dark Passion Play</album>
    <uri>xmpp:hamlet@shakespeare.lit/avserver?transfer;uid=audio235</uri>
    ...
  </item>
  <item id='audio236' parent='dir20'>
    ...
  </item>
</container>
```

Example 7.5: Container

If the controller wants to monitor a directory or any other query result, a dynamic variable as described in section 6.4.2 is created and its pubsub node identifier is returned in the IQ query result. Subsequent updates are sent using pubsub messages to signal changed, added or deleted items and sub-containers.

In its current state, the profile does not support adding new items using XMPP; this is not required to test the scenarios from section 2.1. A future version must include functionality such as uploading files and adding a URI to the server for remotely stored content.

7.2.3 UPnP Gateway

Today, most network storage devices have UPnP support by implementing some of the UPnP AVServer templates and there are media players with integrated UPnP AVServer control point available. The XPMN MediaServer should provide a gateway service to UPnP devices and control points. The most commonly used UPnP templates should be mapped to XPMN actions and events to include (legacy) UPnP media servers and control points.

A gateway device acts as UPnP control point or UPnP device to the UPnP network and as service provider or controller to the extended personal media network. It provides the services of one network to the other. The UPnP functionality can be mapped as followed:

- The gateway to include **UPnP control points** sends one combined SSDP advertisement for all XPMN service providers supporting the MediaServer profiles. It acts as controller to these devices and as UPnP device to the UPnP control points.
- To include **UPnP AVServer devices**, the gateway listens to their SSDP announcements and announces itself as XPMN MediaServer. It has to map all queries between an XPMN controller and the UPnP device. The gateway must be specially designed for media servers. Similar to the UPnP remote control described in section 4.1.2, a generic UPnP gateway without special support for the specific templates and profiles is not possible.

Both protocol suites provide an subscribe/notify mechanism, though XPMN provides auto-subscription while UPnP does not. The gateway must take care of subscribing to the relevant events. The service and device capabilities are limited to the features both protocols define.

Furthermore, a gateway needs support for media streams to connect the two protocol suites. UPnP uses HTTP for media transport while the XPMN MediaServer Transfer profile is based on Jingle Out-of-Band Stream Data. While an XPMN media player might support HTTP playback, HTTP does not work between clients in distinct private networks. A gateway must be able to relay the data traffic between the media network and the UPnP device.

7.3 MediaControl Profiles

Another profile required for the evaluation is the remote media player control. In the XPMN interconnection use case from section 2.1.5, the user controls the TV set of the friend with the mobile phone. The mobile phone is the controller of the TV set's remote control service.

UPnP splits the remote control into the *AVTransport* template for simple playback control such as start, stop and seek, and the *MediaRenderer* template to influence the playback itself by manipulating volume, contrast and brightness. Other than UPnP, experience with JSR-135 available on some mobile phones had influence on the profile design. In the M4 project, the TZI successfully realized a remote playback interface based on HTTP and the JSR-135 API (see section 5.4.5). JSR-135 distinguishes between playback control and rendering control, too.

7.3.1 Media Playback Control

The state machines from the UPnP AVTransport profile and JSR-135 are similar. First the user loads an URI which is used to connect to the remote host and determines the media type (MIME type) of the file. After the player successfully opened the URI, it is possible to receive metadata,

start the playback, or seek to a specific position. During playback, actions to pause, seek or stop are available. This behavior is similar to media player libraries.[8]

The XPMN profile has a similar behavior. When opening a URI the remote player returns the metadata as defined in the MediaServer Directory profile (see section 7.2.2). For music, the controller may show artist, album and title on its display while the actual audio is played on the remote player. When playing audio and video files, the metadata should contain the length in seconds to allow the controller to seek in the file and display a progress bar.

The metadata returned on open indicates whether or not the playing item supports seeking. A web cam may only provide a constant stream and it may be impossible to pause or seek. The player may implement an internal buffer to cache the stream to allow pausing (live pause) and seeking operations for such sources as well. Seeking is done based on time values or percentages and not based on byte positions because the controller does not necessarily support the required codecs and may not even have access to the stream to read the index. To seek, the controller can either specify a relative position of seconds back or forward, or an absolute position based on the beginning of the file. The latter can be used to resume playback of a playback stopped before.

The controller can open a playlist or several URIs simultaneously. The open IQ result may include the metadata for all media files or streams in the playlist or only from the first item. In the latter case, the controller depends on events during playback to receive updated metadata on the currently playing item.

For playlist control, it is possible to jump to the next or previous track in the playlist. Future versions of the specification must include advanced playback features such as chapter selection, subtitle and audio language selection, DVD menus, and DVB teletext—the lack of these features was one reason why RTSP was not used for playback control.

The device controlling the playback is automatically subscribed to the events to retrieve updates on the currently playing item, state changes, and the new playback time after seeking. Auto-subscription based on the XMPP service discovery features is available for clients of the same user. External users interested in the playback status must have permission to receive the events and must subscribe manually. The access control is kept simple and only contains the information whether or not an external user is allowed to access the service; possible values are forbidden (default), only receive playback information (event notification), and full playback control.

7.3.2 Media Rendering Control

Besides playback control, a user may want to control the way the video is rendered or the sound settings for audio. This is handled by the *MediaRenderer* template for UPnP devices and

[8] The two most popular media player libraries libxine (`http://www.xine-project.org/`) and gstreamer (`http://www.gstreamer.net/`) both have such a state machine and interface to the developer.

the *getControl* API call in JSR-135. As a first step, the profile introduced here only defines volume and mute as possible controls. UPnP additionally allows to control the brightness, contrast, sharpness, video black level, video gain, and horizontal and vertical keystone—it is very unlikely that a user changes this very often with a remote controlling device. Anyway, some of these controls may be added in a future version of the Media Rendering Control profile. The IQ get command *get-controls* can be used to retrieve the list of control variables a controller can access for future enhancements.

```
<iq from='hamlet@shakespeare.lit/phone'
    to='hamlet@shakespeare.lit/tv'
    id='hr6rewqd'
    type='set'>
  <control xmlns='urn:xmpp:tmp:media:control:0'>
    <variable name='volume'>60</variable>
  </control>
</iq>
```

Example 7.6: Set Volume for Remote Control

The current value of each control can be access with an IQ get stanza and changed with an IQ set stanza; a new value is sent as an event to all subscribed controllers. This is based on the generic variable handling defined in section 6.4.2, even though a variable does not have its own element name. Example 7.6 shows the IQ set command to change the volume.

7.3.3 D-Bus-based Media Players

While there are not many UPnP MediaRenderers available, many Linux media players can be controlled using D-Bus. Admittedly, integrating D-Bus applications is not that important compared to UPnP devices and control points: while devices such as a NAS device require a firmware update to add XPMN support, a D-Bus application is most likely open source installed on a PC and can be extended to be an XPMN device.

D-Bus supports remote procedure calls and subscription-based events. Both can be mapped to the XPMN core. The parameters of D-Bus procedure calls and events are much simpler than structures that can be defined within an XML schema. Therefore, it should be possible to include existing D-Bus applications while it may be more complicated to create a D-Bus specification based on an XPMN profile. Since there is no standard for profiles in D-Bus each application needs its own gateway service. For instance, a media player gateway needs support for all available media player D-Bus APIs.

7.4 Outlook: Possible Future Profiles

The four presented profiles are only a small subset of possible XPMN profiles. Some of them only cover the edge of the possibilities the profile could provide; they define what is necessary to

test the XPMN interconnection use case as part of the evaluation. This section gives an overview about other possible profiles and enhancements of the current ones—and this list certainly is not complete.

The current MediaServer profile set includes a profile to transfer media data between two clients and to browse a file server for content. A second source for media content is broadcast TV with different channels and an electronic program guide. Similar to the directory profile a TV device has containers (channels) with items in it (programs). The main difference is a temporal dependency: it is not possible to watch something airing in the future and it requires caching and storage capabilities to watch something from the past. An IPTV-based solution could automatically store the streams from all channels for a day to allow time shifting and watching older content. With prices for storage solutions dropping rapidly, longer storage of the content in terms of weeks is easily possible for the IPTV offerings of larger ISPs. Unfortunately, the copyright holders for the media content often only allow streaming a movie or a TV show to the customer without a storage solution in the infrastructure of the ISP.

Some TV stations rely on advertisement and on consumers watching these. A time shifting solution has to take this business model into account and has to find a way to compensate the TV stations for the losses or they will never allow it. In Germany, the Bundesgerichtshof judged that online recording services are legal under special circumstances. The users must schedule a recording prior to the original air time and the content must not be distributed freely (Bundesgerichtshof, Aktenzeichen I ZR 216/06, April 2009). Access to older content is prohibited. This verdict only covers Germany and laws in other countries may prevent such a service for their citizens completely.

A future *TV Tuner* profile may take advantage of the MediaServer Directory profile to store a recording. A *TV Scheduling* profile may automatically schedule TV shows with a certain name and move a recording to the device best suitable for the task. For example, the Freevo TV server automatically detects the list of channels accessible from the available TV tuner cards on multiple devices, the list of channels on the same frequency accessible parallel, and the quality of the video streams. It schedules the recordings on the best possible device based on this information.

With multiple viewing devices and network constraints, for instance, bandwidth limitations on the DSL connection, an XPMN transcoding service may be useful. If the user selects a HDTV recording to be played on the mobile phone, a transcoding device in the home network must prepare the video stream to be playable on the mobile phone and to fit into the Internet uplink. A transcoding device could also provide the possibility to stream a video or an audio file using RTP; RTP may not the preferred way but it is one possible solution.

The remote control profile can be enhanced by several other ways to influence the playback. To replace a classic remote control for the TV set with a mobile phone, the mobile phone does not need to support actions to play and pause the media playback. It must send the name of the buttons pressed similar to a remote control or a keyboard. This service is comparable to the Bluetooth *Human Interface Device Profile* (HID).

A completely different approach is for a service provider to describe its user interface in a generic way and reduce the logic on the controller. Tietjen developed a system where services can describe themselves and generate a simple user interface that can be rendered on a mobile phone [Tie08]. The main advantage is that the mobile phone does not need to be updated to support a new or improved service. This is very important since the development of a mobile client is a complex task due to the various operating systems, supported programming language and toolkits, and capabilities of the mobile phone. Today, many mobile phone applications are developed for J2ME with different vendor specific enhancements, Windows Mobile, Symbian S60, the iPhone, Google Android, and Palm's WebOS. Having one generic client is an appealing idea to keep development costs down. Even though her work is independent from this thesis it was designed to be portable to the XPMN architecture.

An alternative solution is to pick up the HTML rendering introduced in section 6.3.2. A device may provide a web interface to configure and control it; requiring only a web browser on the controller. A website could be sent as XHTML inside the XML stream and images could be linked using Bits of Binary (XEP-0231). The OOB extension from section 7.1.3 together with XTLS provides a secure transport.

There are many possible future profiles; the existing ones provide a solid base to test the core architecture itself. This should be done first. We need to evaluate the developed system before different developers can start adding XPMN support to their applications and defining new profiles.

Chapter 8

Evaluation

We defined the XPMN core architecture in chapter 6, followed by some exemplary profiles in chapter 7. The specifications can be used to build XPMN media servers, remote controllable TV sets or set-top boxes, and controllers to be installed on TV sets and mobile phones. The following questions arise when looking at the whole system with the use cases and requirements from chapter 2 and chapter 5 in mind:

1. Is it fully specified? The specifications may be incomplete or different developers interpret them differently. Additionally, there may be error conditions and unusual situations not covered.
2. Is it an extended personal media network? There is no formal definition; the architecture is designed based on the requirements derived from the use cases from section 2.1. It must be possible to realize these use cases with the defined architecture.
3. Is it able to overcome current network restrictions? The specifications must cover two clients in distinct private networks, NAT penetration, and dynamic IP addresses for a secure client-to-client communication.
4. Is it secure (enough)? The developed security layer is a compromise between high security and usability. But despite usability concerns, it must cover the user's security requirements—even if the user cannot name them.
5. Is it usable for non-technical users? The usability of the device pairing and the media network itself should be evaluated from the user's point of view.
6. It is useful? Even if the previously asked questions are all answered in the affirmative, the usefulness of XPMN is unknown. We defined use cases in section 2.1, but it is unclear whether or not there will be a user base interested in this new technology. This raises the question what the targeted user group is. Who is an XPMN user?

This chapter tries to find answers to these questions. An XMPP implementation with XPMN support is developed to test the specifications and realize one use case. Afterwards, we take a look at the three topics network, security and usability to answer the last four questions.

8.1 Implementation of an XPMN Framework

An implementation is helpful to test a specification. Sometimes a seemingly simple protocol flow is very complicated to implement or the specification is missing error conditions an implementation runs into. This is one reason why a XEP needs to have at least two reference implementations before it can advance from Draft status to Final status. A secondary reference implementation is required to test the interoperability between the two implementations—to test whether or not the developers interpret the specification the same way.

The implementation developed within the scope of this thesis was used to test the three TLS variants from section 6.1: TLS over In-Band Bytestreams, Jingle XML Streams, and Jingle XTLS. During tests with a typical home network connected to the Internet with an ADSL modem, it was discovered that the Jingle XML Streams have too many round-trips to be usable. This finding is the reason why the number of round-trips were an important factor when developing Jingle XTLS.

Klaus Hartke was the first one implementing the Jingle SOCKS5 Transport[1] and he discovered some concurrency problems and clients not choosing the best possible connection methods. The findings from his tests resulted in adding priorities to the candidate information. These are only two examples where an implementation proved to be useful to test the specifications.

8.1.1 Existing XMPP Libraries

The easiest way to implement an XPMN library is to take an existing XMPP library and extend it by the necessary XEPs and Internet Drafts. The purpose of the implementation is to add XPMN support to Freevo to test the XPMN interconnection use case. Therefore, the library should either be written in Python or in C. There seems to be no existing C library with Python bindings that can be extended in Python. This reduces the choice to reuse existing implementations to Python libraries.

There are various Python XMPP libraries, but most of them are not in active development anymore. The four remaining ones are Twisted Words, PyXMPP, Xmpppy and SleekXMPP.[2] Twisted Words is bound to the Twisted framework, making it hard to integrate into Freevo which is based on the kaa framework. Using Twisted Words would result in two threads each running their own main loop and neither Twisted nor Freevo is thread safe. Xmpppy was abandoned at the time this thesis was started and forked by several projects with no central code base; the different versions are incompatible to each other. Xmpppy's ability to be extended is limited and it only includes the basic functionality for instant messaging. Therefore, neither Twisted Words not Xmpppy can be used.

[1] His XMPP and SOCKS5 implementation is not publicly available.

[2] Twisted Words is part of the Twisted framework. It is often used to write XMPP server components in Python (`http://www.twistedmatrix.com/`). PyXMPP is available at `http://pyxmpp.jajcus.net/`, Xmpppy is hosted at Sourceforge (`http://xmpppy.sourceforge.net/`), and SleekXMPP at Google Code (`http://code.google.com/p/sleekxmpp/`).

The two remaining XMPP libraries are extensible, in active development, and it is possible to include them into the kaa framework used by Freevo. But both are designed for the normal XMPP use case: instant messaging. An XPMN implementation needs support to access services on devices, to address specific clients, and requires presence support per client and not only per user—the available libraries are not designed to meet these requirements. Both libraries also have no Jingle support and adding end-to-end XML streams between clients would break their architecture.

Therefore and because XPMN does not require support for any XMPP chat feature, it seems to be a better approach to write an XMPP library from scratch. Client-to-client XML streams, the device concept, and extensibility for profiles should be part of the initial design.

8.1.2 Core Architecture

The library should become part of the kaa media framework which is the basis for Freevo.[3] The architecture has a slim core implementing the basic XMPP protocol flow as defined in RFC 3920. The core allows connecting to an XMPP server and includes a stanza handler based on an XML SAX parser. A generic plug-in system allows registering callbacks for certain stanzas based on the stanza type (e.g. message or IQ) and the name and namespace of the first child element. Unlike other Python XMPP libraries the core contains a `RemoteNode` class as abstraction layer for communicating with other nodes; either another XMPP client or a server. There is no association between multiple `RemoteNode` objects even if they belong to the same bare JID; each `RemoteNode` represents a remote XPMN device.

An extension to the core consists of two parts: a `ClientPlugin`, independent of any remote node, and a `RemotePlugin` that will be instantiated for every `RemoteNode` providing that feature. The stanza handler allows callbacks based on the remote node's full JID. This makes it possible to have direct callbacks into plug-ins of the client object or plug-ins of a specific peer. Figure 8.1 depicts a very basic view of the core with the plug-in system.

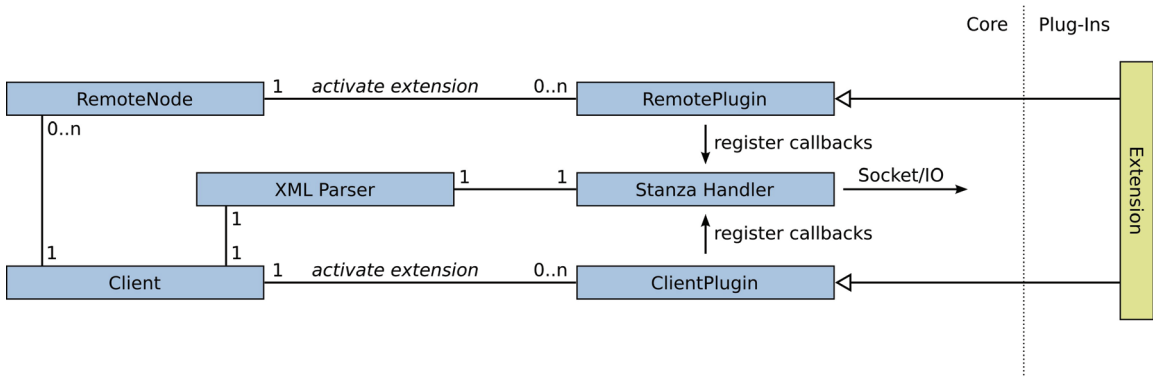

Figure 8.1: Kaa.XMPP Library Core and Plug-in Interface

[3]The source code is available in the kaa repository at `svn://svn.freevo.org/kaa/trunk/xmpp`.

This architecture allows the developer to access each extension individually for both client and remote object and makes it possible to add, enable or disable an extension very easily. The kaa.xmpp core contains neither presence, roster management, device and service discovery, nor Jingle. All these extensions are realized as plug-ins. Example 8.1 contains the XEP-0030 *disco#info* IQ handling used for the service discovery (the actual kaa.xmpp plug-in looks different because it also contains the capability caching; see the "Device and Service Discovery" subsection about what the plug-in is actually doing).

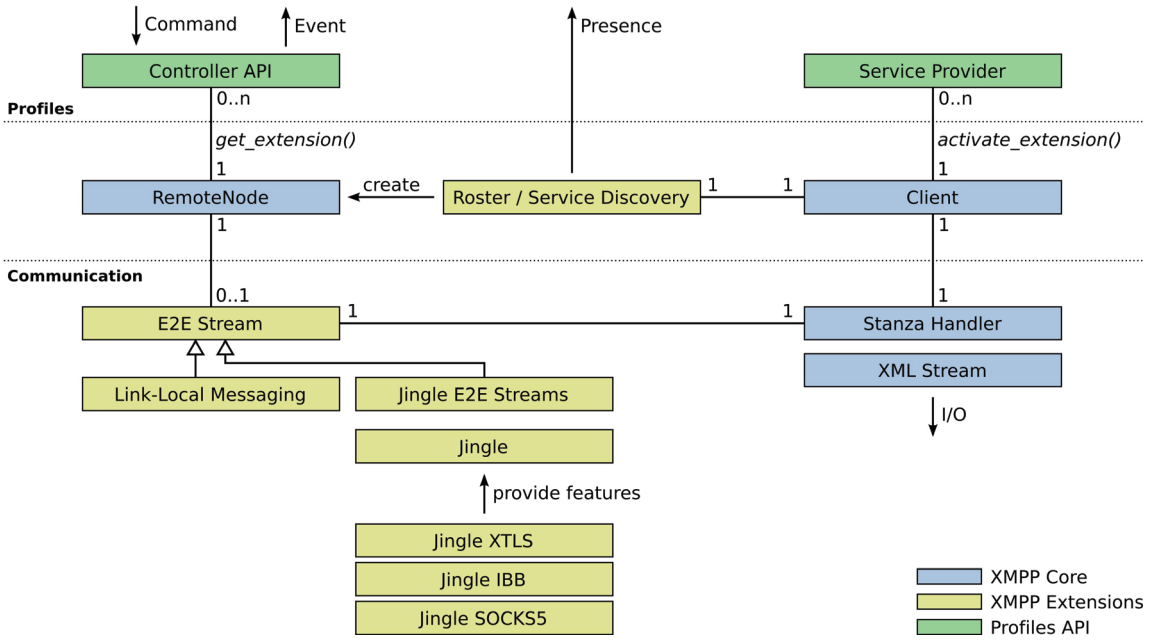

Figure 8.2: kaa.XMPP XPMN Implementation

On top of the XMPP core, kaa.xmpp includes plug-ins for the various XEPs and Internet Drafts required by the XPMN core. Figure 8.2 depicts the XPMN core with the discovery mechanisms and secure end-to-end communication. The kaa.xmpp module contains no plug-ins for Jingle SOCKS5 Streams (XEP-0260), Out-of-Band Stream Data (XEP-0265), and SASL EXTERNAL certificate handling (XEP-0257 / Internet Draft).

Device and Service Discovery

The device and service discovery is realized by two plug-ins; the RFC 3921 roster (roster.py) and XEP-0030 service discovery (disco.py). The service caching defined by XEP-0115 (Entity Capabilities) is merged into the two plug-ins since the information from XEP-0030 is hashed and transmitted to the peer using <presence>.

Example 8.1 shows the XEP-0030 part of the service discovery plug-in. The `RemotePlugin` provides a query function to send the *disco#info* query IQ stanza to the peer. It waits for the result (the `yield` stops the function and returns to the main loop until the result is received)

```
import kaa
import kaa.xmpp

NS_DISCO_INFO = 'http://jabber.org/protocol/disco#info'

class RemoteNode(kaa.xmpp.RemotePlugin):

    @kaa.coroutine()
    def query(self):
        """
        Service Discovery (XEP-0030) disco#info
        """
        r = yield self.remote.iqget('query', xmlns=NS_DISCO_INFO)
        self.cache.features = []
        for feature in r.get_children('feature'):
            self.cache.features.append(feature['var'])
        yield self.cache.features

class Client(kaa.xmpp.ClientPlugin):

    @kaa.xmpp.iq(xmlns=NS_DISCO_INFO)
    def _handle_query(self, jid, stanza, const, stream):
        """
        Service Discovery disco#info callback
        """
        result = kaa.xmpp.Result('query', xmlns=NS_DISCO_INFO)
        result.add_child('identity', category='client', type='bot',
            name=self.client.appname)
        for feature in self.client.features:
            result.add_child('feature', var=feature)
        return result

# register extension
kaa.xmpp.add_extension('disco', [NS_DISCO_INFO], Client, RemoteNode, True)
```

Example 8.1: XEP-0030 disco#info Plug-in

and stores it locally. The `ClientPlugin` class defines the other side of the communication and registers a handler for the query IQ stanza and returns the result to the caller.

The roster plug-in is responsible for the device discovery and indirectly for the service discovery, too. When receiving presence information from a peer, the plug-in looks into the service discovery cache for the XEP-0115 capability hash. If the hash string cannot be translated into a list of services, the module calls the service discovery plug-in's query function and waits for the result. The result is cached for subsequent presence information. If a plug-in exists for a service and it contains a `RemotePlugin` class, an object of this class is instantiated and associated with the `RemoteNode` object. Once a device is discovered and its services are known, the existence of the device is signaled to the application by `Client.roster.signals['presence']`. The service and device discovery plug-ins are automatically instantiated; the library assumes that all peers support XEP-0030 and XEP-0115. This may not be true for instant messaging; yet, support for these extensions is an XPMN requirement.

The serverless-messaging extension (linklocal.py) has its own discovery mechanism based on multicast DNS (requires the Avahi Zeroconf daemon). It only needs to be activated and hooks into the existing device and service discovery mechanism provided by the roster module; for instance, it reuses the roster's signals to provide a single notification interface for the developer.

End-to-End Streams

Even though end-to-end streams are realized as plug-ins, the concept of an XML stream between two clients is part of the initial design. Each `RemoteNode` object has a `stream` attribute and member functions to send message and IQ stanzas to the peer using that `stream`. The object representing the XML stream between client and server is the default value and a plug-in can replace it.

The serverless-messaging plug-in replaces the `stream` of peers discovered using mDNS. The new `stream` object opens an XML stream to the peer once a message or IQ stanza should be sent. It opens a TCP connection to the port provided in the DNS TXT record, performs the XEP-0174 (Serverless Messaging) stream setup with the XEP-0250 (C2C Authentication Using TLS) TLS authentication. The key management is transparent to the developer and only depends on a callback for the TLS-SRP passwords. If no callback is provided, SRP is removed from the possible authentication methods. The callback to get the TLS-SRP password is called only if TLS-SRP is used.

The Jingle XML Streams plug-in (jingle_streams.py) implementing the Jingle XTLS Internet Draft uses a different approach. Unlike link-local communication where the plug-in must open a direct TCP connection for all kinds of communication, an insecure communication is always possible when connected through a server and security is an optional feature. The plug-in exposes a `connect` method to the application to open a secure communication. In example 8.2, the application sends a message stanza to the `peer` without end-to-end security enabled, opens a Jingle XML Stream, and sends a second message; this time secured using XTLS.

```
peer.message('This␣message␣is␣insecure')
e2e = peer.get_extension('jingle-streams')
try:
    if not 'e2e-stream' in peer.stream.properties:
        yield e2e.connect()
except Exception, e:
    raise IOError('unable␣to␣open␣e2e␣stream')
peer.message('This␣message␣is␣secured␣using␣XTLS')
```

Example 8.2: Opening a Jingle XML Stream

The end-to-end secure Jingle XML Streams plug-in requires Jingle (jingle.py), Jingle In-Band Bytestreams (ibb.py), and XTLS (xtls.py). It opened a new stream using Jingle and requests IBB as transport and XTLS as security layer. On success, the `stream` object of the `RemoteNode` is replaced with an end-to-end secured XML stream. An application can query the object's

attributes to determine whether the stream is an end-to-end stream (property `e2e-stream`) and secured using TLS (properties `secure` and `peer-certificate`).

The application developer using kaa.xmpp does not need to deal with certificates; everything is handled inside the library. Furthermore, there is no difference in the API between secure link-local communication and server-routed Jingle XML Stream sessions based on In-Band Bytestreams and XTLS.

8.1.3 XPMN Profiles

A profile is implemented as a `ClientPlugin` and a `RemotePlugin` with dependencies to the end-to-end communication plug-ins. The `ClientPlugin` implements the part of the service provider; a service can be called on the client independent of a specific remote peer object. The controller part is implemented as `RemotePlugin` providing an API to the developer to control the remote service. By requesting a specific plug-in on a `RemoteNode` object, the developer gets an object specially for that profile. For instance, the media renderer `RemotePlugin` (controller) exposes functions to start, pause and stop the remote playback similar to an object for local playback. In example 8.3 the `media-control` plug-in is accessed and the returning object behaves like the local Freevo player with member functions for playback control and signals being emitted when the playback is started or stopped—the signals are automatically mapped to pubsub events (PEP).

```
if 'media-control' in remote_node.features:
   player = remote_node.get_extension('media-control')
   player.signals['started'].connect(start_callback)
   player.signals['stopped'].connect(stop_callback)
   metadata = yield player.open(uri)
   player.play()
```

Example 8.3: Remote Playback Control

An application can easily add a new XPMN profile by registering a new plug-in. Furthermore, it is possible to overload some functions of specific extensions by inheriting from the plug-in classes. This is required to add additional functionality not provided by the XMPP library itself. For instance, the media renderer service provider extension includes the required code to parse the XML elements, but does not have actual playback code. The application has to inherit from the service provider plug-in to create a link between kaa.xmpp and the actual player.

To test the XPMN interaction use case from section 2.1.5, most profiles from chapter 7 are implemented using kaa.xmpp and parts of Freevo. The MediaServer profiles are based on the Freevo media database *Beacon* and the Media Playback Control profile is implemented as Freevo 2.0 plug-in;[4] the Media Rendering Control profile is not implemented because it is not required to test the use case.

[4]Beacon and Freevo 2.0 are not yet released and only available in the Freevo subversion repositories located at `svn://svn.freevo.org/kaa/trunk/beacon` and `svn://svn.freevo.org/freevo/trunk`.

Figure 8.3 depicts the XPMN MediaServer implementation. The application (blue) hooks into Freevo components (green) and kaa.xmpp (yellow). The Freevo media database Beacon contains a powerful metadata parser and a database for the metadata XML elements described in section 7.2.2. The IQ query stanzas defined in the MediaServer Directory profile are directly linked to Beacon.

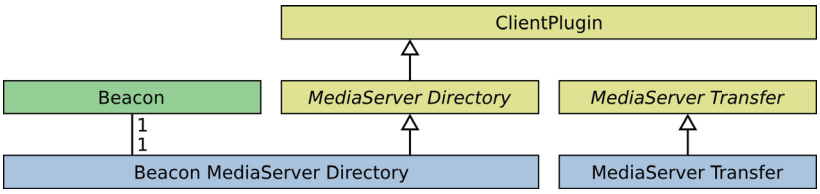

Figure 8.3: XPMN MediaServer Application

The media transfer is not realized using Out-of-Band Bytestreams due to lack of support in existing media players; extending a media player is out of the scope of this thesis and not required to test the XPMN core architecture. Instead, the media file is transmitted using HTTP and requires a port forwarding rule when the file is accessed from outside the home network. This will be changed in a future version when OOB transport is accessible by the used media players. These restrictions must be kept in mind when using the implementation to test the XPMN interconnection use case.

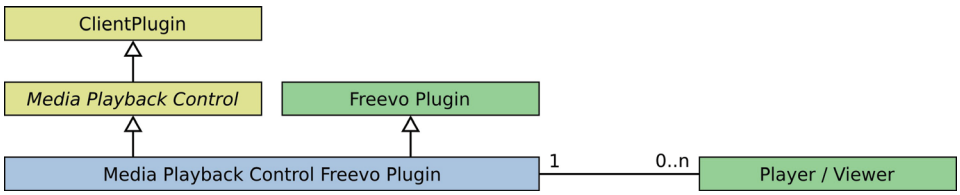

Figure 8.4: XPMN Freevo Plug-in

The Media Playback Control profile is connected to Freevo using the Freevo 2.0 plug-in interface (green). A plug-in is able to start playing a URI and has access to the currently used player; this can be a video or audio player as well as the image viewer object.

8.2 XPMN Interaction Use Case

The implementation is used to test the fourth, most complex use case from section 2.1.5. It covers every aspect of the XPMN core by including two personal media networks with individual user certificates and device listings, access restrictions, and two service providers belonging to multiple users controlled by the same controller.

It contains a mobile phone as controlling device, a TV set in the friend's media network, and a media server at home. All these devices are implemented based on the kaa.xmpp framework and run on Linux machines; kaa.xmpp is not compatible with available mobile phones and as

a consequence the mobile phone is replaced by a PC. The scenario requires a pubsub server with virtual nodes and persistent storage for certificate management. Since no existing pubsub server supports this setup, both media networks contain an XPMN management client described in section 6.5 for signing and key distribution. The XMPP server used is ejabberd 2.0.5.

In the test romeo@freevo.org is visiting juliet@freevo.org and wants to play a video from his home server (romeo@freevo.org/avserver) on her TV set (juliet@freevo.org/tv) using his mobile phone (romeo@freevo.org/phone) as remote control. At the beginning all devices are part of their media network and their certificates are known to the corresponding management devices (romeo@freevo.org/management and juliet@freevo.org/management) and thus to all other devices in the same personal media network. The two users and their personal media networks have no relationship yet. Figure 8.5 depicts the test setup with three hosts with the five devices in two private networks and the freevo.org XMPP server for both media networks. The test runs without any user interaction and simulates the users' behaviors; a user interface is not required to test the specifications.

Romeo's mobile phone connects to the freevo.org XMPP server and logs in using Romeo's user name and password.[5] It receives the presence status from the media server and the management client at home after sending its presence information. The status includes the already known capability strings and the mobile phone recognizes the MediaServer Directory service and the key management service. The mobile phone also discovers Juliet's devices in the LAN using multicast DNS, but ignores them because the TXT records indicate that the devices belong to an unknown media network.

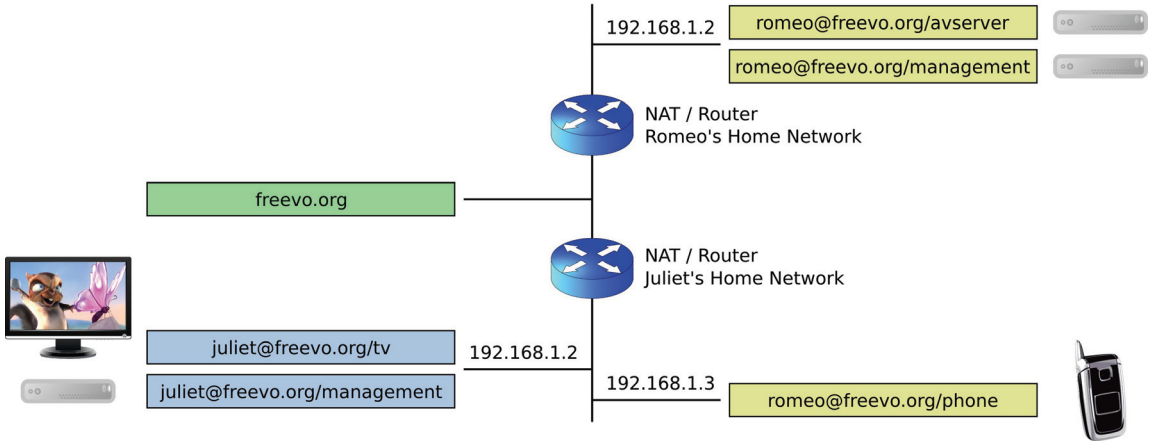

Figure 8.5: XPMN Use Case Test Setup

Romeo chooses to search for a video file. The mobile phone opens a Jingle XML Stream session to the media server and the clients use their certificates for XTLS peer entity authentication; the certificates are already known. Thereafter, the phone sends multiple query requests to search for the video file. Since the client used to test the use case has no user interface it chooses a file or

[5]There is no existing XMPP server with support for uploading certificates to be used with SASL EXTERNAL. With a certificate-based login, the client does not need to know the password. This limitation has no further effect on testing the use case.

sub-directory randomly. The results are very small and therefore sent in-band without opening an Out-of-Band Data Stream. After several queries browsing the media collection's directory structure, Romeo's phone receives the metadata for the desired video file.

Upon asking where to play the file Romeo chooses "playback on a device nearby". His mobile phone is aware of Juliet's devices in the local network and the capability strings provided in the TXT records. The capability string from Juliet's management client is recognized because it is the same management client Romeo uses; the TV set is an unknown device. The phone opens a TCP connection to the TV set and performs a XEP-0030 service discovery. XEP-0030 is available prior to the STARTTLS feature. The phone closes the connection after receiving the service discovery information. The phone now knows that the TV set device in Juliet's home network is capable of media playback and shows it as possible playback device to Romeo. Romeo selects Juliet's TV set as playback device and his phone adds an access control rule on the media server allowing juliet@freevo.org access to the video file.

Romeo's mobile phone opens a TCP connection to the TV set and the TV set provides its certificate and SRP as possible authentication options in the <features>. The mobile phone does not recognize the certificate and checks its management client for Juliet's user certificate; a pubsub error message indicates that there is no trust relationship between Romeo and Juliet. The mobile phone requests the STARTTLS feature and sets SRP as only possible authentication method. The TV set accepts the STARTTLS and confirms the usage of SRP.

The implementation uses a fixed PIN on both devices due to the lack of a user interface, but both clients should open a dialog box with detailed information about the pairing process. The two users must be informed about the identity of the other (bare JID), the devices' names, and that they should enter the same PIN. The PIN is used as password in the TLS-SRP handshake and the TLS-SRP user name is romeo@freevo.org/phone (initiator). After opening a secure stream, the clients exchange their user certificates and send the received certificates to their management clients for signing. The management clients sign the peers' user certificates with their users' private keys and send a pubsub message with the new certificates including the signatures to all clients in their personal media network. Romeo's media server at home is now in possession of Juliet's user certificate, but does not know of the TV set and its certificate.

The mobile phone adds juliet@freevo.org to Romeo's roster and a subscription request is sent to all of Juliet's devices. The TV set in charge of adding the new external user approves the presence subscription. With the subscription set to "both" all of Juliet's clients receive the presence information from all of Romeo's clients and the other way around. If a device does not recognize the capability string in the presence information it sends a service discovery IQ stanza to the corresponding device. In this test, the media server queries Juliet's TV set but already recognizes Juliet's management client's capability string from the management client in its own media network. Juliet's TV set queries Romeo's media server and the mobile phone; the mobile phone already knows the TV set, but not the other way around. The two management clients also perform the XEP-0030 service discovery on the newly discovered devices. Now all devices in both personal media networks have discovered each other and each other's services.

The phone sends a media renderer open IQ with the URI of the video file to initiate the video playback. Next, Juliet should be asked for confirmation since the TV set's access control lists do not contain romeo@freevo.org as allowed controlling user. Due to the lack of a user interface, she automatically selects "Allow control for 24 hours". The URI to play contains the full JID of Romeo's media server and the TV set opens a Jingle XML Stream to it. The Jingle session-initiate contains the TV set's certificate and SRP as possible authentication methods. The media server does not recognize the TV set's certificate but is in possession of Juliet's user certificate. The media server asks Juliet's management client for the TV set's client certificate and verifies the signature with Juliet's user certificate it already has.[6] The media server has no user input capabilities, is not in pairing mode, and therefore does not allow SRP as authentication method. Its session-accept contains X.509 as only possible authentication method and its device certificate's fingerprint. The TV set performs a similar look-up to be able to verify the media server certificate and signals X.509 as authentication method to be used. The Jingle session-initiate and session-accept messages also opened an In-Band Bytestream. The TV set starts the TLS handshake to secure the Jingle transport.

After opening the secure end-to-end stream, the TV set requests the video file from the media server. The next step in the specification is for the media server to open a Jingle OOB session for the video data. In the current version, the implementation sends an HTTP URI instead (see section 8.1.3). The correct behavior would be to negotiate SOCKS5 candidates, open an OOB stream, and send the video file to the TV set. Upon receiving the first bytes of the video file the TV set detects the media type, reads the metadata, and sends the metadata as result to the initial open IQ stanza back to the mobile phone. At the end, the mobile phone sends a play IQ stanza and the TV set starts the video playback. Romeo and Juliet can now watch Romeo's video on her TV set and both can control the video playback.

On the next day, the pairing process and certificate exchange can be skipped. The access control rules only allowed Romeo control over the TV set for one evening and Juliet access to one video file—these user interactions will still be required when they meet again in Juliet's chamber to watch one of Romeo's videos.

8.3 Review of the Networking Requirements

As discussed in section 5.1, XMPP fulfills most of the requirements on the networking layer. They focus on the ability for a controller to communicate with all available service providers and for the service providers to notify interested controllers on changes. The task to send a message to every device independent of the network topology is covered by the XMPP core without modifications. Due to the fact that every client opens a connection to the XMPP server

[6] Certificate distribution is not secured by XTLS, making the management client as secure as a pubsub server with the same task. The management client only replaces the pubsub server due to missing support for virtual pubsub nodes with persistent storage on the used XMPP server.

to route messages, network address translation and dynamic IP addresses do not cause any problems besides a short disconnection.

Serverless messaging in the home network allows a direct connection between clients even if the XMPP server is unavailable, for example, if the DSL connection is down. Both server-based and serverless messaging support presence exchange for device discovery and include the capability hash for a faster service discovery.

However, the security layer uses a TLS stream between two clients. This creates the new requirement to open a stream with TCP characteristics between two clients with error correction, data ordering, and data transparency. The developed solution is able to penetrate any NAT by using In-Band Bytestreams relying on the already covered server-routed stanza delivery. Jingle may try other transport methods first, but always has In-Band Bytestreams as fallback. Every client can open an end-to-end secure communication link to any other client by tunneling the TLS stream over XMPP (requirement 12).

XEP-0198 adds disruption tolerance and reliable delivery even if a client or the whole home network becomes temporally unreachable due to a DSL reconnect resulting in a new IP address (requirements 14 and 15). The developed Python implementation does not include XEP-0198, but XEP-0198 was considered in the initial design and it can be easily added to the XMPP core.[7] The specification was completely rewritten in 2009 and as of September 2009, no available XMPP server supports it. The clients could support XEP-0198 on Jingle XML Streams to not rely on a server supporting this feature. This usage of XEP-0198 is not yet specified.

In-Band Bytestreams are unsuitable for media transfer. The XMPP server will likely throttle the media stream to be unusable for streaming audio; not to mention video. The developed SOCKS5 Jingle transport solves this problem by covering most network setups:

1. It works if either the initiator or the responder has a public IP address and no firewall is preventing the peer from connecting,
2. it takes advantage of NAT-PMP (see section 3.2.1) and UPnP Internet Gateway Devices (see section 3.1.1) as NAT assisting solutions if one of them is available in local network,
3. supports IPv6 addresses with or without Teredo tunnel (see section 4.1.2),
4. and has a relay-based solution as fallback.

The first three cover typical home networks and the fourth guarantees a client-to-client connection if a proxy is known to at least one client. While performing tests with multiple network setups it was always possible to send XMPP stanzas and media files between clients. Admittedly, the media transfer proved to be difficult if both clients were behind a NAT and the proxy was the only way to open a connection. It may be difficult to find a proxy and it is unclear what bandwidth it provides. For the tests performed here the address of a SOCKS5 proxy was provided manually;[8] the average user does not know addresses of proxies. XEP-0215 (External

[7] XEP-0198 must be added to the kaa.xmpp core and cannot realized as plug-in. It interacts with the XML parser directly and is independent of the namespace of the sent stanza.

[8] Some XMPP servers have an integrated SOCKS5 proxy. A list of available servers including their features (e.g. SOCKS5 support) can be found at `http://www.jabberes.org/servers/`.

Service Discovery) could be used to provide this information by the server, but XEP-0215 is not widely deployed yet and depends on the XMPP server administrator to provide SOCKS5 proxies for the users. The XSF is aware of the problem which also occurred for discovering TURN servers needed for Jingle ICE-UDP RTP sessions.

Today, the biggest issue with SOCKS5 or TURN servers is the deployment: there is no business model for running a relay service. While running an XMPP server also lacks a business model, its operating costs are low; unlike SOCKS5 proxy servers which require a large bandwidth and create huge traffic costs (e.g. HD video streaming). The people with interest in using a relay server are the ones with a reason for deploying them. Obviously, not every XPMN user is able to do this; it is not trivial to rent a server with a public IP address and install a SOCKS5 proxy.

Hartke and Bormann reduce this task in *STUN/TURN using PHP in Despair* (STuPiD) [HB09]. STuPiD requires renting a (virtual) server for web hosting with the ability to install PHP scripts, a MySQL database, and unlimited traffic. This is cheap and easy to maintain. "By designing the STuPiD web service in such a way that it can be implemented by a simple PHP script [...], it is easy to deploy by those who need the STuPiD services. The combination of the low-throughput out-of-band channel for synchronization and the STuPiD web service for bulk data relaying is somewhat silly but gets the job done." The STuPiD protocol uses HTTP to send (POST) data to the server and to retrieve it with another client (GET). Internally, the data is temporarily stored in a database to relay it between the two HTTP connections. The STuPiD Internet Draft includes an exemplary implementation on server side with less than 50 lines of PHP code which was tested with a free-of-charge web hoster. While this is no solution for the average user, it is one suitable Jingle transport for some. If the Jingle SOCKS5 Bytestreams Transport Method fails, STuPiD is a possible alternative before falling back to In-Band Bytestreams.

An XMPP-aware router in the home network as depicted in section 6.5 is another interesting solution to this problem. The router always has a public (maybe dynamic) IP address and could announce being a SOCKS5 proxy for the user's devices and even to friends. Similar to Skype, devices with enough bandwidth could relay data for others.

Teredo is another way to open a direct connection between clients, though, in some tests performed within the scope of this thesis Teredo failed to set up a connection between two hosts. It seemed to be that the traffic was not routed to one host and dropped by the last Teredo relay. The usage of Teredo and IPv6 in general has increased over the last year [Lab09]; starting on the day uTorrent 1.8, the first version with IPv6 support, was released. This shows that Teredo is up to the task to provide IPv6 connectivity for peer-to-peer applications such as BitTorrent. If it works for BitTorrent it also works for XPMN device communication.

To summarize, the XPMN core architecture requirements are fully met. Clients can discover each other, discover their services, and exchange stanzas securely through NAT boundaries. Only profiles such as the *MediaServer Transfer* require the usage of direct TCP connections which will not work in all possible network topologies and may fail due to installed software firewalls. Nevertheless, it will work in most cases.

8.4 Review of the Security Requirements

The specified end-to-end security layer provides the missing requirements outlined in section 5.1 based on the security analysis from section 5.3. Two devices can use Jingle XTLS based on In-Band Bytestreams to exchange stanzas; providing authentication, confidentiality, and data integrity (requirement 12). The separation of client and user keys and certificates together with the XEP-0189-based signing process allows to determine whether or not a client belongs to a specific media network (requirements 18 and 26). Pubsub is a simple mechanism to distribute changes of the certificate list to the clients (requirement 19).

Using the certificates in SASL EXTERNAL for logging in to the server provides an alternative login mechanism without knowing the password; thereby, a simple method to remove a stolen device later (requirement 20). The user can always use the password as fallback to log in if an attacker was able to remove all rightful controllers (requirement 21).

The two autonomous parts where the certificates and the private keys are used make the system either secure against a rogue server or compromised clients. Difficulties arise, however, when the same attacker is in possession of a client and has control over the server. In that case it is possible to prevent the user from deleting the credentials of that particular client. This scenario is very unlikely and a result of the compromise we had to make to reduce the complexity of the security layer to keep it user-friendly. As a workaround, a user-controlled device could be used as certificate pubsub server, separating the server login from the certificate handling. Admittedly, the problem arises again if that device gets compromised.

Friends and family members can be included as external users. Each person has their own personal media network with its own devices and the various media networks can interact. To not rely on the XMPP server for authentication, the friends' user certificates are signed with the user's private key similar to device certificates. Access control lists on the devices themselves can be used to grant external users access to specific functions or media assets (requirement 27).

8.4.1 Deployment Issues of TLS-SRP

The usage of TLS-SRP might be a problem for the deployment of the Jingle XTLS security layer. There are only a few TLS libraries supporting SRP, forcing developers to either skip the easy bootstrap mechanism or to rewrite larger parts of their code and switch to another TLS library. For instance, Klaus Hartke wrote an XMPP implementation in C# and the TLS library shipped with Microsoft Windows Vista does not support SRP. Therefore, his implementation currently requires the user to manually distribute the X.509 certificates between the devices. This is a suitable solution to get the desired security, but bad in terms of usability.

Additionally, the bindings to use TLS libraries written in C or C++ in other programming languages require special SRP handling, too. The two mostly used open source TLS libraries OpenSSL and GnuTLS both have TLS-SRP support, but their Python bindings only support

X.509 certificate-based authentication. The XPMN implementation used for the evaluation uses TLSlite[9] which seems to be the only Python TLS module with SRP support.

The dependency to TLS-SRP also limits the XTLS Internet Draft to become an informal RFC because TLS-SRP is informal and not a standards-track RFC. The main reason for this is a possible patent on techniques used in SRP. Details are unknown; the intellectual property rights notification only states that SRP might violate patents from Lucent.[10]

A future version of the XTLS Internet Draft may use a different authentication method if the certificates are unknown. It may be possible to use SASL inside Jingle for authentication and *channel bindings* to bind the authentication to the TLS session. The *Salted Challenge Response* (SCRAM) SASL mechanism [MSMNW09] uses parts of the TLS handshake and a password as authentication method. Yet, a serious weakness of SCRAM is that it only authenticates the TLS client and an XMPP client in the role of the TLS server can perform a brute-force attack to get the password for a simple four-digits PIN. A secondary issue is that the TLS Finished messages are preferred for the SCRAM channel binding because they are unique for each TLS session. A TLS library must have support to access the unencrypted Finished messages—a feature the Microsoft Windows TLS library also does not have.

A channel binding SASL method based on SRP and the X.509 certificates used in the TLS negotiation might be a suitable solution. It provides the security of SRP without needing support in the TLS library. Unfortunately, such a SASL method does not exist. However, there is an expired Internet Draft defining an SRP-based SASL method [BN03] which is implemented in Cyrus-SASL, a well-known open source SASL library. Alexey Melnikov, one of the authors of SCRAM and active in the SASL working group as well as the XSF, wants to revive the draft and add channel binding techniques to it. This would shift the problem from the TLS library to the SASL library; whether SRP has a better chance of deployment as a SASL mechanism is unclear at this point.

Alternatively, SRP is not needed if the user trusts the local network and uses the leap of faith approach to add a new device in the LAN. New pairing technologies based on NFC (see section 6.2.5) may be deployed faster than TLS-SRP and can replace the current authentication bootstrapping mechanism.

To summarize, it is unclear if SRP will be a problem for the deployment. Maybe the developers switch to alternative TLS implementations if the default library does not support SRP, maybe NFC will be the primarily used XPMN authentication technique, maybe something completely different based on SASL will be used, or maybe TLS-SRP will be widely deployed in the future. Ideas exist to integrate TLS-SRP into web browsers and use the browser's user interface to log in to websites.[11] It is likely that this will increase TLS-SRP support in TLS libraries and language bindings.

[9] A simple Python TLS library; `http://trevp.net/tlslite/`.

[10] See `http://www.ietf.org/ietf-ftp/IPR/LUCENT-SRP` for the original declaration.

[11] The thread "Client Certificate UI for Chrome?" on a cryptography mailing list discusses the integration of TLS-SRP into Chrome; `http://www.mail-archive.com/cryptography@metzdowd.com/msg10739.html`.

8.4.2 Provided vs. Required Security Level

A legitimate question to be asked is whether or not the system is secure. There is no satisfying answer; it depends on personal security requirements. XPMN security is a risk management process where each user has a distinct expectation and is exposed to different risks. For instance, the leap of faith pairing method without a password is less likely to be attacked if the home network is completely wired without Wi-Fi. Wi-Fi can be made more secure by enabling WPA or WPA2—admittedly, this may become less true in the future.[12] A house on the countryside compared to one located on a crowded street makes the wireless link also more secure by reducing the number of possible attackers in range.

As discussed in section 6.2.5, usability affects the security, too. It is more user-friendly to just touch a new device with the mobile phone without any password. But in doing so, an attacker can add a new device by just becoming close, for instance, in the subway. Furthermore, the mobile phone, the obvious choice to be used to add new devices using NFC, is the device with the highest risk of getting stolen.

For this reason, XPMN gives the user a choice to select the appropriate security level. Unfortunately, "many users do not understand—or may not be aware of—the threats associated with a technology" [KPW06]. For instance, the fact that everyone in radio range can capture traffic on an unencrypted Wi-Fi link and by that, observe what websites a user is visiting, requires a deeper understanding of the technology—an understanding that cannot be taken for granted. Besides, many people do not expect that they could be the target of an attack.

The device used must explain the process and the risks of the various security settings, helping the user to make a reasonable security decision (requirement 32: inducing awareness). This problem cannot be solved by the XPMN core architecture and is part of the user interface design process. The XPMN architecture can only provide multiple pairing mechanisms to satisfy the individual security requirements of the users.

- A secure operation requires TLS-SRP for device pairing and friend authentication. The PIN must be exchanged over a secondary communication channel.
- In a less secure environment the user adds a new device or a friend without password relying on leap of faith. An attacker impersonating a device in subsequent communications will be detected, but the initial pairing process is vulnerable. The user interface must explain the risk to the user. It is impossible to automatically detect how secure the used network is (see section 5.3.1).
- A mobile device may protect the device pairing with a password. The password is not part of the XPMN architecture and is only used to either start the pairing process in the user interface or to encrypt the user's private key.

The XTLS security layer supports all of these security settings and leaves the choice to the users and the interface designers. Even if XTLS is considered to be not user-friendly enough, it is

[12] Recently Ohigashi and Morii found a way to crack WPA encryption under 60 seconds [OM09]. WPA2 is still considered secure.

easy to exchange the XTLS security layer in Jingle with a different mechanisms and keeping the rest of the core architecture as it is. This makes the security solution extensible and replaceable for future development.

8.5 Review of the Usability Requirements

An evaluation of the usability requirements is more complicated compared to network and security because usability is not a feature that either works or does not work. A protocol by itself is never usable; the question is, whether it is possible to build a usable system based on the XPMN specifications.

This section describes research activities on usability studies for pairing methods, raises general question about usability studies, and outlines an XPMN usability study. Afterwards, we take a look at the developer's point of view because their acceptance of the architecture is important for a wide deployment, too.

8.5.1 Related Work

Some usability conclusions can be derived from similar research activities. The TLS-SRP pairing is similar to Bluetooth device pairing; NFC pairing methods depend on the usability of NFC in general. Besides the pairing process, the overall XPMN usability and the usefulness of such a system is of importance. They depend on the targeted user group.

TLS-SRP Pairing

An important observation when performing usability studies is a change in the participants' behaviors when they are told that security is being evaluated [KPW06]. To make a usability evaluation of pairing methods even more complicated, security is not the task the users want to perform. "Security is usually a secondary goal. People do not generally sit down at their computers wanting to manage their security; rather, they want to send email, browse web pages, or download software, and they want security in place to protect them while they do those things" [WT99]. If the participants know that the pairing is an important security task, their alertness to security changes. This means that the acceptance of the security depends on how the task is explained to the user.

Two usability evaluations of comparing fingerprints show that it is not only the task that matters, but that the usability depends on the user interface as well. While the task "Check if both devices display the same value" with two buttons *Yes* and *No* produced a 20% error rate, the text "Compare the PIN numbers on both devices, are they DIFFERENT?" with *SAME* and *DIFFERENT* as options reduced the error rate down to 2.5% [UKA08, accentuation adopted from source]. Therefore, one single evaluation—no matter how large the number of participants

is—cannot come to a distinct conclusion on whether or not a mechanism is secure and user-friendly. The usability highly depends on the interface design. If a usability study reveals problems in the pairing process, it does not necessarily mean that the pairing process is not usable; it may only be a problem with the user interface.

Surprisingly, in one evaluation comparing pairing methods including string comparison (e.g. fingerprints or Short Authentication Strings) and entering a PIN on both devices as proposed here, the participants described string comparison as easiest and most fun to use [VTK07]. It was the preferred choice in that particular evaluation. The same evaluation showed a 6% error rate where the same string comparison provided a man-in-the-middle access to the system. From this it follows that in this case the preferred pairing method is not wise to use. If the user actively participates in the pairing process and does not want to rely on leap of faith, the system must prevent an attacker from adding a device into the network. It is only an annoyance if an error results in the user not being able to add a device or a friend, but it is an unacceptable security breach if a user error makes it possible for an attacker to get a device in the media network.

Using Near Field Communication

The key usability issue with Near Field Communication (NFC) seems to be the lack of a mental model for this technology [BGHM06]. While RFID, the predecessor to NFC, has gained some popularity in enterprise contexts, the average user knows neither the technology nor the possible use cases due to the lack of consumer products using it. This makes NFC less user-friendly at first glance. “Until the technology becomes ubiquitous in homes, however, the general public need first to familiarise itself with it. That might happen with the help of the applications designed for mobile phones, such as electronic payment, ticketing and information-on-the-move” [Rai07]. A usability study for NFC-based XPMN device pairing performed in Japan instead of Europe or the U.S. will likely come to a different outcome, because in Japan RFID is already deployed in consumer electronic products.

The short range of NFC, an important security feature, can cause a usability problem when large devices such as a TV set should be added to the media network using a mobile phone. The user has to know where the NFC tag is located on the device to successfully initiate the pairing process. As of today, there is no well-known logo that could be used. Certainly, it is most likely that such a logo will exist once NFC-based interaction with phones is introduced to the consumer; for instance, micro-payment also requires the user to know where the NFC tag is hidden. A secondary problem is that a user may not be able to read the phone’s display during the pairing process due to the location of the tag [KO07]. This requires the usage of sounds or tactile feedback such as vibrations to notify the user when the pairing process is finished or aborted due to an error.

These are only some possible usability problems of an NFC-based system. They depend on how NFC is used in other scenarios; the usability depends on how familiar the consumers will get with this technology.

XPMN Target User Group

Many security evaluations done by universities use students as participants; often they are from the field of computer science. This falsifies the results since the average user has far less computer skills. Other studies about a new technology only target younger people. The question arises how meaningful these usability studies are and what the target user group for the media network is.

Prensky divides consumers of today's electronic devices into two categories: digital natives and digital immigrants [Pre01]. "Today's students represent the first generations to grow up with this new technology. Computer games, email, the Internet, cell phones and instant messaging are integral parts of their lives. [...] It is now clear that as a result of this ubiquitous environment and the sheer volume of their interaction with it, today's students think and process information fundamentally differently from their predecessors." From this it follows that younger people will interact differently with a new technology such as the media network. And even in the group of digital natives, the technology usage differs based on age, gender, and social background [Had07].

As pointed out in section 6.2.6, most of the digital natives already have an instant messaging account. Teens often share images and music stored on their mobile phones using Bluetooth and in consequence must perform the Bluetooth device pairing and are familiar with this process. It is likely that these digital natives are also interested in accessing their data everywhere at any time. The same may be true for some digital immigrants; however, some people are not interested in this technology at all. There are many people with no interest in mobile mp3 players, digital cameras on mobile phones, and sharing photos with services such as provided by Flickr. It is questionable whether these people will use the media network outlined here at all.

Greenberg and Buxton take Ubiquitous Computing technologies (Ubicomp) for the home as an example of usability studies and the target groups that will actually use a technology. "The culture is not yet in place to exploit such technology: it is not a case of one person using Ubicomp, but of a critical mass of home inhabitants, relations, and friends adopting and using that technology. Only then does it become valuable. Again, cultural and technical readiness is needed before a system can be deemed 'successful'. How does one evaluate that except after the fact? Do usability evaluations of toy deployments really test much of interest?" [GB08]. This raises the question about the usefulness of an XPMN usability study.

Usefulness

In *The Trouble with Computers* [Lan96] Landauer adds another important aspect for an evaluation: the usefulness. Most evaluations focus on how easy it is to use a technology and not if it is useful at all. The usefulness is hard to foresee: "[...] Individuals tend to purchase wireless phones for emergency and coordination reasons, and do not consider sociability to be important. However, within weeks of purchasing the phone these same owners used it for social

calls" [EG01]. This indicates that even the consumers may find it hard to foresee how they will use an extended personal media network. Additionally, a service useful for one user may be uninteresting or too complicated for another.

If we cannot evaluate the usefulness of a technology, how can we evaluate the usability? "Another way to say this is that usability evaluation, as practiced today, is appropriate for settings with well-known tasks and outcomes. Unfortunately, they fail to consider how novel engineering innovations and systems will evolve and be adopted by a culture over time" [GB08]. Therefore, the usefulness for the media network cannot be foretold. The use cases in section 2.1 describe how users may use the media network, but it is unclear whether a wide range of users actually wants to use the technology that way. Maybe they do not need a media network or maybe the media network will be used for something completely different. Without the use case and the services that will actual be used, the results of a usability study are questionable. The extended personal media network needs to be deployed first to determine its usefulness and its use cases.

Commercial products such as Orb (see section 4.1.1) show that some people have an interest in accessing their media everywhere at any time and the popularity of Flickr shows that there is a target user group interested in sharing images. Yet, these two tasks are only a small subset of the possibilities the XPMN architecture provides and the usefulness of the essential part—the end-to-end security layer—cannot be answered by them.

8.5.2 Requirements for a Usability Study

The requirements for a usability study can be derived based on the findings from the previous section. Since the usefulness of the media network is impossible to determine without actual wide deployment, the evaluation should focus on the pairing handshake. The following items must be kept in mind when performing an XPMN usability study:

1. An XPMN use case must be defined which includes the XPMN device pairing. The evaluation should focus on the pairing method. A possible scenario would be the task of playing a video on a TV set with the mobile phone as user interface. The scenario must include an initial device pairing; the actual task is only for the participants and secondary for the evaluation. As a side-effect, the evaluation may show the usefulness of the given task.
2. Fully functional devices are needed to be sure that the environment is accurate. This includes a real set-top box with an XPMN service provider and controller connected to a TV set and a mobile phone with an XPMN stack. The exemplary implementation from section 8.1 in Python can be used on a set-top box, but will not work on currently available mobile phones. A second implementation based on a phone's operating system is required.
3. To test possible future pairing methods, the set-top box or the TV set as well as the mobile phone need NFC support. While it is relatively easy to hide a passive NFC receiver in an

existent set-top box, we need a mobile phone with an integrated NFC sender to provide a realistic feeling to the user. An NFC logo must be designed to indicate where the tag is hidden.

4. The evaluation must be performed with a large group of participants from various possible target user groups; not only digital natives or students at a university. An evaluation must be done with participants of various social backgrounds, gender, and age. These attributes must be included in the evaluation, if possible by a factorial analysis. One possible result of the evaluation is to get an idea why people are interested in such a technology.

5. The user interface must be changed in case participants have problems with the device pairing. Maybe the pairing method can be made user-friendly and only the interface to the user is not sufficient for the task.

6. The device pairing should be attacked during some evaluations to monitor the participants behavior if the device pairing fails.

To conduct this kind of usability evaluation, several months of work are needed to create the scenario, implement functionality on the devices, and to perform the study with several participants of various social backgrounds, gender, and age—and we need to define the maximum age of a possible target group. And still, the result will only cover the pairing process and not the XPMN architecture in general.

Therefore, the findings of existing pairing evaluations described in section 8.5.1 will have to suffice as a first indication about the usability of the XPMN security layer. An extensive usability study has to be performed once the XPMN core with working profiles is deployed.

8.5.3 Performance vs. Usability

The system's overall performance has an impact on the usability, too. Currently available UPnP-enabled TV sets and set-top boxes are capable of browsing the media collection of a UPnP AVServer in the local network. The same task adapted to the XPMN architecture has a different user experience when the user is browsing the media collection remotely using a 3G connection. While the access to the media server in local network is fast, the 3G connection is much slower and selecting a folder may take a couple of seconds. This limitation is based on the underlying network topology and technology, and is independent of the XPMN core specification and the used profiles.

Whether or not a lack of performance disrupts the overall usability depends on the user's expectation. A similar delay is notable for other tasks on the mobile phone such as browsing the web or accessing email. If the mental model of the user includes that the mobile phone in general is slower when not connected over Wi-Fi, the user experience may not be affected much. If the user expects the system to behave the same no matter where the phone is located and how it is connected, a slow link will confuse the user and this will have negative effects on the usability.

A second performance problem arises when opening the TLS secured XML stream. Due to the two-way Jingle handshake and the four-way TLS handshake it can take up to two seconds to open a secure channel to a device in the home network.[13] This six-way handshake is close to the minimum the XPMN architecture can provide based on TLS—we cannot reduce the number of round-trips used for the TLS handshake. Keeping the Jingle session open reduces the delay to the initial connect.

There is nothing the XPMN core architecture can do to prevent or reduce the delays. The application developers could cache some values (e.g. metadata for known items) and the interface designers should make the user aware of the delay. Instead of blocking the user interface when browsing a remote media collection, the interface must give the user some feedback that the request is being processed; if possible a progress bar should be displayed to provide an estimation how long the user has to wait.

The somewhat higher delay for 3G connections may discourage the user from performing some actions remotely or cause the user to abandon some services due to usability problems. This may affect the overall usefulness of a service. Real-life deployment is the best way to find performance-related usability problems. An evaluation with deployed devices and early adopters is needed and may provide solutions for additional problems.

8.5.4 The Developer's Point of View

The previous sections are user-centric—nevertheless, the XPMN application developers are also important; without acceptance from developers XPMN has less chance of deployment. Hartke executed a usability study on his XMPP library [Har08] and discovered that similar to users using an application, developers using an API have a mental model that may not be correct. A developer may want to use an API differently as expected by the API designer. A goal is to make it simple for developers to add XTLS and XPMN support to existing XMPP libraries, or when designing a new one to make it simple to add a new service to a generic XPMN device interface.

The exemplary implementation developed within the scope of this thesis provides a basic understanding of how complex the security layer and the XPMN core architecture is. By comparing kaa.xmpp to other XMPP libraries two important differences are noticeable:

1. Most libraries have one object used for communication with other XMPP nodes by sending stanzas to the server. The security layer requires direct XML streams between clients. Libraries with serverless messaging support already have client-to-client XML streams and it may be easier to include Jingle XML Streams and XTLS in these.

2. In the XPMN architecture, clients belonging to the same user are separate devices providing multiple services. In the XMPP chat use case, the central element is the user (bare

[13] The test setup resulting in a two second XTLS negotiation included two home networks connected to the Internet by DSL without fast-path.

JID) and the actual client is not that important. For example, some libraries only provide presence information for each friend in the roster. Individual client presence information and information from clients of the same user are not accessible.

Adding XTLS and XPMN support to existing libraries may break their architecture and it may not fit into the design. It is easy to keep these two points in mind when designing a new XMPP library. The developed XMPP library is very small and lightweight: around 2000 lines of code of the XMPP core and another 2000 lines for Jingle, XTLS, pubsub, and serverless messaging. This simple library provides the XPMN core to develop profiles.

8.6 Summary

To summarize, the XPMN core architecture is a suitable solution to the outlines use cases, but only a compromise between networking and security requirements on one side and usability on the other. Importantly, the developed XMPP extensions fulfill the requirements refined in chapter 5, and the XPMN core architecture together with Jingle SOCKS5 Streams and Out-of-Band Stream Data meet all XPMN requirements from section 2.2. The XPMN architecture is a seamless solution for a peer-to-peer network where devices can securely interact with each other.

Testing the scenario revealed some minor issues that should be addressed in future versions of the XMPP extensions. For instance, there is no IQ stanza for a controller to ask a service provider for permission to get access to specific functions (ACL) and no way to query what access rights it has. Currently, the controller just tries to access a resource and this request may be relayed to the user if the service provider has a user interface. In the tested scenario Juliet's TV set prompts Juliet to allow or deny Romeo's device playback control. From this it follows that the access control subsystem fulfills the requirements, but may need some enhancements to be more usable from the developer's point of view.

The usefulness is the main unanswered question; it is unclear whether or not there is a user base that will want to use an extended personal media network. On the other hand, some parts of the overall architecture may be useful in other use cases: XTLS secured Jingle XML Streams allow end-to-end secure chatting and to exchange encryption keys for SRTP. Jingle SOCKS5 Streams and XTLS can be used in other Jingle applications such as file transfer. Secure communication and a working file transfer solution is on the XSF agenda for 2009. If the XSF members think it is a useful topic, maybe other users find it useful, too.

Chapter 9

Conclusions

> *"When once you have tasted flight, you will forever walk the earth with your eyes turned skyward, for there you have been, and there you will always long to return."*
> — Leonardo da Vinci

In this last chapter we reflect on the achieved engineering results and take a look at the future regarding the ongoing standardization process and the deployment of the XPMN core architecture and its protocols. Afterwards, we sum up possible XMPP extensions mentioned throughout this thesis that were never specified, highlight some open issues, and end this thesis with some concluding remarks.

9.1 Engineering Results

In this thesis, we have discussed a number of protocols and deployed products for device discovery and service discovery as well as the communication between two devices. The focus was on networking constraints and reasonable security; always keeping the usability in mind. While neither the protocols nor the research activities extending them fulfill the requirements to realize the extended personal media network outlined in chapter 1 and chapter 2, they provided a solid base for further research activities that led to the development of the XPMN core architecture and exemplary profiles. The standardization process was done openly in the XSF and the IETF for additional feedback from the XMPP community as well as security experts in the IETF.

This section summarizes the three most important engineering results: an end-to-end security layer for XMPP based upon Jingle, Out-of-Band Data Streams together with the SOCKS5 Bytestreams Transport Method, and a generic device architecture turning XMPP into a third generation peer-to-peer network. In addition to the summary provided here, Appendix A gives a brief overview of the protocols specified and published within the scope of this thesis, including their current status and the list of authors.

9.1.1 End-to-end Security Layer for XMPP

The resulting XMPP end-to-end security is based on TLS, a well-known and widely deployed protocol. While the research pointed to TLS from the beginning, its usage for client-to-client communication evolved over several iterations: from a simple "TLS over In-Band Bytestreams" over complex Jingle XML Streams based on link-local communication, to a new security layer for Jingle. Jingle itself was updated by its authors to allow such an additional layer without restricting it to the actual use of TLS. Unlike the previous attempts, a Jingle security layer can be replaced easily if it is not sufficient for some future use cases without affecting the actual Jingle applications. Moreover, its usage is not bound to stanza exchange and can be used for file transfer as well—it can be used by all Jingle applications that require a transport with TCP characteristics. XTLS based on DTLS for UDP-based transports is a possible future enhancement.

The solution was well received by the XMPP community and members of the IETF. After the first attempts based on TLS and In-Band Bytestreams were published, end-to-end security made it on the XSF agenda for 2009 and was added as working group item for the newly formed XMPP Working Group in the IETF. Developing the end-to-end security for XMPP inside the IETF is an important achievement of this thesis.

The certificate handling is based on XEP-0189 which already existed before the work on the XPMN architecture started. Based on the existing XEP from Paterson, a simpler XML representation and an optional signing mechanism was specified. The separation of user and client keys and certificates creates a two-layered PKI with client certificates signed by the user keys outside the X.509 certificate chain. Optionally, existing X.509 PKI structures can be included and the OpenPGP web-of-trust can serve for validation, too. This keeps the signing of user certificates very flexible while the client certificates are always bound to a simple signature mechanism.

And finally, a client's private key and certificate can also serve to authenticate it against the XMPP server. This allows clients to connect to the server without them knowing the password. It is possible to individually remove them once they should not be part of the XMPP network anymore.

9.1.2 Jingle SOCKS5 Transport and Out-of-Band Data Streams

Independent of end-to-end security, the newly developed Jingle SOCKS5 transport method can be used in Jingle sessions to exchange data ordered and reliable without routing it in XML stanzas over the XMPP servers as defined by In-Band Bytestreams. The protocol is more than just an adjustment of XEP-0065 to Jingle: it handles bidirectional communication and different candidates from both the initiator and the responder. The protocol inherits some ideas from ICE and ICE-TCP while it reuses the existing infrastructure of deployed XMPP SOCKS5 relay servers. Outside the XPMN scenario, this specification is a possible transport for Jingle file transfer. Similar to end-to-end security, a working Jingle file transfer solution is on the XSF agenda for 2009—the new Jingle SOCKS5 transport is an important step for realizing this goal.

On top of the SOCKS5 transport method, Out-of-Band Data Streams can be used to eliminate subsequent SOCKS5 candidate negotiations for short-living sessions. XML elements can be multiplexed in one Jingle transport to allow large stanzas without blocking the (ordered) XML stream.

To reduce its complexity, both protocols are not part of the XPMN core architecture. Nevertheless, they are important for the MediaServer Transfer profile. Other XMPP extensions such as *IO Data* (XEP-0244) and *Bits of Binary* (BOB, XEP-0231) can benefit from them as well.

9.1.3 XPMN Device Architecture

The XPMN device architecture built upon the Jingle XTLS end-to-end security layer, the Jingle SOCKS5 transport method, and Out-of-Band Data Streams turns XMPP into a powerful and secure third generation peer-to-peer network with the XMPP servers as super nodes. The clients may route their data over the server, but can open a direct connection if necessary; furthermore, clients can provide a SOCKS5 relay service to others. Device and service discovery is provided by XMPP and existing XEPs, clients can exchange data or control messages, and the communication is secured with TLS.

The extended personal media network outlined by the use cases in section 2.1 is only one possible application for this peer-to-peer network. It is a generic solution that allows applications on the Internet to securely send small messages and larger data chunks to others. This tells XPMN apart from peer-to-peer networks used for file sharing such as BitTorrent. Unlike BitTorrent, the protocol suite does not allow anonymous access; the authentication and device discovery based on presence subscription is an integral part of the design.

9.2 Standardization Process

Even with the specifications described here published, the XPMN standardization process is still ongoing; the XEPs are experimental and the Internet Drafts are no working group documents yet. Besides the core, extensions for media transfer and profiles were introduced and their development process is ongoing, too. This thesis marks only the beginning—the basic scenarios to start the development. This section briefly summarizes the next standardization steps with focus on deployment, possible enhancements of existing XMPP extensions and ideas for new ones, and states some open issues that must be kept in mind for future specification and application development.

9.2.1 The Next Steps for Deployment

The ongoing standardization process for deployment can be split into three different tasks. Each one of these tasks depends on the success of its predecessor.

1. First and foremost, the XMPP end-to-end security layer based on the XTLS Internet Draft and XEP-0189 Public Key Publishing needs to be deployed. Application developers may find further issues not uncovered by implementing the kaa.xmpp XPMN framework. End-to-end security interoperability tests of XTLS implementations is an openly discussed work item for the next XMPP Summit in Brussels 2010.

   At the IETF 75 in Stockholm in July 2009 on the first meeting of the newly formed XMPP Working Group, a security requirements document [SA09b] became an official working group draft. With the requirements document now accepted as work item, a next standardization step is for XTLS to become a working group draft as well.

2. The second step is to make more people in the XSF interested in working on bot-to-bot communication and not only focus on instant messaging; XMPP is more than a chat protocol. While there is no need for a XEP in the Standards Track describing the XPMN architecture because it is built completely out of other extensions without its own protocol flow or XML schema, an Informational XEP may be needed to outline the architecture and show how the extensions interact with each other.

3. And finally, the outlined profiles need to be improved, implemented in various media center solutions and devices, and published as official XMPP extensions. New profiles must be developed; section 7.4 lists many possible profiles that could be used on top of the XPMN core. Since the XSF is an open organization, developers from media-related open source projects, commercial products, and research projects are invited to build applications on top of the XPMN core. Members of the EU project NoTube [A+09] are thinking about building (parts of) their TV-centric semantic web infrastructure on top of XMPP. They showed interest in XMPP and asked about the personal media network architecture and how the research of this thesis can be reused in the NoTube project.[1]

At some point, the XPMN architecture needs to be deployed to gather further information about usability, the target consumer group, and the usefulness of the system. The development of profiles on top of the XPMN core is an ongoing task that hopefully never gets finished—there are many use cases which realizations could benefit from this architecture.

9.2.2 Possible XEP Development

Throughout this thesis, some new extensions as well as some possible enhancements to existing ones are mentioned but not specified. These specifications and enhancements are not needed for the XPMN core architecture, but may help to improve the user experience or the deployment. After all, the core cannot be deployed without profiles; it does nothing useful on its own.

[1] A brief summary of ideas from Dan Brickley working on the NoTube project can be found on the XMPP social mailing list archived at `http://www.mail-archive.com/social@xmpp.org/msg00474.html`.

Out-of-Band Data Streams (OOB)

As mentioned in section 7.1.3, XEP-0265 could be modified to use the BEEP framing format. This would remove the need to specify a new framing format and rely on another existing specification. But OOB based on BEEP will violate the BEEP RFC because there will be only messages of the type MSG; BEEP requires an answer to that message type and OOB does not include the concept of answers. Furthermore, most features of BEEP are not needed and the byte counter, a feature useful for the media transfer, is too small and will wrap for large video files. But using BEEP in a modified version does not provide a substantial benefit over the HTTP-based framing format anymore.

Another enhancement is a Jingle transport based on the OOB Jingle application. By that, other Jingle applications can use OOB as transport to multiplex the data over one real transport. For example, if an Out-of-Band Data Stream based on the Jingle SOCKS5 transport is already open, another Jingle application just uses that already negotiated SOCKS5 session by multiplexing its data over the Out-of-Band Data Stream. As a result, XML Streams, large out-of-band stanzas, and media files could be transmitted over the same SOCKS5 connection.

The reliability of Jingle sessions and the out-of-band data is another open issue. While it is possible for UDP-based Jingle sessions to replace the transport when networking conditions change, an IP address change of the home network will disrupt the byte stream and it is impossible to resume it. Since OOB is already counting bytes, it will be relatively easy to include an acknowledgment mechanism into the specification to resume a session later. Alternatively, this reliability layer could be added to Jingle directly; similar to XTLS being a new security layer in Jingle.

Furthermore, a reliability layer allows to start with an In-Band Bytestream as transport and change the transport to SOCKS5 once a higher bandwidth is required or while the SOCKS5 candidate negotiation is in progress (transport-replace). For instance, a device opens an Out-of-Band Data Stream to a peer for secure client-to-client communication (XML Streams over OOB over XTLS over a reliability layer over IBB). Once the clients want to exchange larger stanzas or media data, the transport could be replaced by Jingle without affecting the higher layers.

HTML over XMPP

Section 6.3.2 depicts the possibility to use an internal web server for access control configuration. This internal web interface could also be used to control a device without having a controller capable of the required profile.

Instead of a full web server for such a simple task, HTML over XMPP with simple IQ stanzas for GET, PUT and POST may be a better solution. The user interface can be built with HTML, CSS and JavaScript—technologies interface designers are familiar with—and the only difference is that the data is transmitted over XMPP instead of HTTP. Files such as images benefit from Out-of-Band Data Streams for transport and XTLS can provide optional security similar to HTTPS.

Multicast Security

Another interesting topic is multicast security which is not covered by Jingle XTLS. As described in section 6.1.5, the existing application layer multicast protocols for XMPP, Multi User Chats (MUC) and Publish/Subscribe, cannot benefit from a Jingle security layer. (X)TLS only provides a secure stream between two entities.

A secure MUC is an ideal communication mechanism to synchronize several devices without opening an XTLS channel to all of them. While it is relativity easy to create such a chat room when the room itself is trusted, key distribution if the room is untrusted is much more complicated. More research on secure multicast protocols is required.

Client-to-Client Usage of XEP-0198

The last previously mentioned but not specified enhancement is the disruption tolerance for end-to-end XML streams. The latest version of XEP-0198 adds reliability and stream resumption to XMPP. The specification is designed for the classic client-to-server communication and does not work together with the XTLS Internet Draft for a direct communication between two clients. If the clients use In-Band Bytestreams as transport they get reliability indirectly since the underlying IQ stanzas are covered by XEP-0198.

A future version of XEP-0198 may include a section how the protocol flow has to be adjusted for client-to-client communication. Although, if we add reliability to Jingle or XEP-0265 as proposed above, reliability could be a Jingle feature and Jingle XML Streams do not need to define reliability on their own.

9.2.3 Open Issues

Besides enhancing existing and developing new XMPP extension protocols, there are some open issues that must be kept in mind for the deployment and the future profile development. These open issues are *Digital Rights Management* (DRM) for some video contents, a usability study, and software restrictions for a wide deployment.

Digital Rights Management

In the beginning of chapter 2 we limit the view to that of the user and explicitly ignore the content providers' and the ISPs' requirements in the use cases. The basic idea is that the content industry will respect their consumers' demands. While this may be true in the future, the currently available set-top boxes do not work well together with the XPMN architecture due to DRM restrictions.

DRM mechanisms are not part of the XPMN design. For DRM to work, the content providers (TV networks, studios) must trust the devices involved—must trust that the device respects the

DRM restrictions. This is a common problem with DRM: the consumer is not trusted; the devices do not trust their owners. It may be possible to include a second trust relationship between devices based upon the information whether the peer was produced by a trustworthy vendor. Yet, DRM is out of the scope of this thesis; and it is likely that the movie industry will follow the music industry and abandon DRM.

Usability Study

We did not conduct a usability study for reasons described in section 8.5. Most importantly, because it is a new technology and its usefulness is hard to foretell without real products implementing it—without deployment. Nevertheless, the usability study outlined in section 8.5.2 should be performed even without actual deployment to get a first indication about usability and feedback on the overall usefulness.

Of course, its results must be evaluated based on the fact that the users are not familiar with the XPMN use case, have no mental model for NFC device pairing, and that the target group for this technology is still unclear. The study is best to be repeated after XPMN devices are deployed.

Deployment of TLS-SRP

The dependency on TLS-SRP may prevent a wide deployment. As described in section 8.4.1 TLS-SRP is not always available. While it is possible for a C# developer to use a third party TLS library with TLS-SRP support, developers on mobile phone platforms may not have this option.

Deployment of XTLS-based chat clients is the only way to determine whether or not the dependency on TLS-SRP is a problem. If chat client developers refuse to integrate this technology, it has failed. Possible enhancements are making TLS-SRP optional or relying on SASL as authentication protocol.

UPnP Integration

Today, most network storage devices support the UPnP AVServer template to access files. Due to the wide deployment of these devices they should be integrated into the media network. Some devices additionally support SMB/CIFS or NFS to access the data, but UPnP is a common feature. It is not wise to tell people they have to replace these devices if they want to access their media data from outside the home network.

A firmware update is the best way to include these devices into the extended private media network. In most cases, a vendor builds the firmware for new devices upon the existing one. Therefore, it is likely that an XPMN implementation for a new device can be ported to already

deployed devices. Unfortunately, many vendors have no interest in enhancing old devices and prefer that costumers buy a new one. Over and above, installing a new firmware is not an easy task and the average user may be overburdened by it. An alternative solution is a UPnP-XPMN gateway to integrate these devices.

9.3 Concluding Remarks

To sum up this chapter and this thesis in general, the XPMN architecture developed provides a solid base for a secure peer-to-peer network based on XMPP: devices can discover each other, discover each other's capabilities, and can communicate securely even through NAT barriers. The main focus of this thesis has been on creating a user-friendly end-to-end security layer for XMPP. The Jingle XTLS security layer has been proposed as a solution for the trade-off between strong security and system usability. It supports several device pairing mechanisms because there is no single correct way for all users; the required level of security highly depends on the user's individual security requirements.

With the XPMN architecture, we have presented an answer to the initial question for this thesis: "How can users interact with all their media through different interface devices in a unified, secure, and user-friendly way?" We have presented

- a **unification** layer based on XMPP where all devices can communicate with each other the same way whether they are in the home network or on the Internet,
- a **security** layer based on Jingle and TLS where two clients can open an end-to-end secure communication channel defined by the XTLS Internet Draft and the key management from XEP-0189, and
- a **usable** authentication and authorization layer that reduces the user interaction to the device pairing process and a simplified access control.

The specifications were discussed openly and published in the context of two standardization organizations; starting the standardization process. This process is still ongoing, but this thesis marks the beginning of a secure inter-device communication architecture based on XMPP.

The results discussed in this thesis provide a generic architecture for various use cases and scenarios. One such application area certainly is home entertainment—an extended personal media network.

Bibliography

[A+09] Lora Aroyo et al. NoTube—Making TV a Medium for Personalized Interaction. Project summary, 2009. http://www.notube.tv/images/stories/notubesummary.pdf.

[Are08] Dirk Arendt. *Universeller Zugriff auf verteilte persönliche Medienserver*. Diploma thesis, Universität Bremen, January 2008.

[ATE07] B. Aboba, D. Thaler, and L. Esibov. Link-local Multicast Name Resolution (LLMNR). RFC 4795 (Informational), IETF, January 2007.

[Aur05] Simon Aurell. Remote controlling devices using instant messaging: building an intelligent gateway in Erlang/OTP. In *ERLANG '05: Proceedings of the 2005 ACM SIGPLAN workshop on Erlang*, pages 46–51, New York, NY, USA, 2005. ACM.

[BEM04] Karlo Berket, Abdelilah Essiari, and Artur Muratas. PKI-based security for peer-to-peer information sharing. In *P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, pages 45–52, Washington, DC, USA, 2004. IEEE Computer Society.

[BGHM06] Sara Belt, Dan Greenblatt, Jonna Häkkilä, and Kaj Mäkelä. User Perceptions on Mobile Interaction with Visual and RFID Tags. In *Mobile Interaction With the Real World (MIRW 2006) workshop*. MobileHCI, Helsinki University of Technology, 2006.

[BKML07a] Carsten Bormann, Dirk Kutscher, Dirk Meyer, and Kevin Loos. M4: Make My Media Mobile—Project Report. Internal document, Universität Bremen TZI, June 2007.

[BKML07b] Carsten Bormann, Dirk Kutscher, Dirk Meyer, and Kevin Loos. ScaleNet Demonstrator—Project Report. Internal document, Universität Bremen TZI, September 2007.

[BKML08] Carsten Bormann, Dirk Kutscher, Dirk Meyer, and Kevin Loos. ScaleNet Demonstrator—Second Project Report. Internal document, Universität Bremen TZI, July 2008.

[BN03] K. Burdis and R. Naffah. Secure Remote Password Authentication Mechanism. Internet Draft (Expired), IETF, May 2003. draft-burdis-cat-srp-sasl-08.

[BS06] S. A. Baset and H. G. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–11, April 2006.

[BS07] P. Belimpasakis and V. Stirbu. Remote Access to Universal Plug and Play (UPnP) Devices Utilizing the Atom Publishing Protocol. In *Networking and Services, 2007. ICNS. Third International Conference on*, pages 59–59, Athens, June 2007.

[CA98] J. Cohen and S. Aggarwal. General Event Notification Architecture Base. Internet Draft (Expired), IETF, July 1998. draft-cohen-gena-p-base-01.

[CAG05] S. Cheshire, B. Aboba, and E. Guttman. Dynamic Configuration of IPv4 Link-Local Addresses. RFC 3927 (Proposed Standard), IETF, May 2005.

[CGI+99] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions, 1999.

[Che] Stuart Cheshire. How does Zeroconf compare with Viiv/DLNA/DHWg/UPnP? `http://www.zeroconf.org/ZeroconfAndUPnP.html`.

[Cho03] Konstantinos Chorianopoulos. The digital set-top box as a virtual channel provider. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 666–667, New York, NY, USA, 2003. ACM.

[CK08] Stuart Cheshire and Marc Krochmal. DNS-Based Service Discovery. Internet Draft (Work in Progress), IETF, September 2008. draft-cheshire-dnsext-dns-sd-05.

[CK09] Stuart Cheshire and Marc Krochmal. Multicast DNS. Internet Draft (Work in Progress), IETF, September 2009. draft-cheshire-dnsext-multicastdns-08.

[CKS08] Stuart Cheshire, Marc Krochmal, and Kiren Sekar. NAT Port Mapping Protocol (NAT-PMP). Internet Draft (Work in Progress), IETF, April 2008. draft-cheshire-nat-pmp-03.

[CMVE06] Antonio Cuevas, Jose I. Moreno, Pablo Vidales, and Hans Einsiedler. The IMS Service Platform: A Solution for Next Generation Network Operators to Be More Than Bit Pipes. *Communications Magazine, IEEE*, 44(8):75–81, August 2006.

[Cor06] The UPnP Implementers Corporation. UPnP technology—The Simple, Seamless Home Network. Whitepaper, December 2006.

[CY08] Jongmyung Choi and Chae-Woo Yoo. Connect with Things through Instant Messaging. In *The Internet of Things*, volume 4952 of *Lecture Notes in Computer Science*, pages 276–288. Springer Berlin / Heidelberg, 2008.

[DC08] R. Denis-Courmont. UDP-Encapsulated Transport Protocols. Internet Draft (Work in Progress), IETF, July 2008. draft-denis-udp-transport-00.

[D'H06] Michel D'Hooge. Report on a secure home network architecture and related Protection Profiles specification. Deliverable DA.3.7, Information Society, February 2006. `http://www.ist-ipmedianet.org/DA.3.7_UPnP_Security_V1_0.pdf`.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), IETF, August 2008.

[DS07] Walter Dees and Paul Shrubsole. Web4CE: accessing web-based applications on consumer devices. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1303–1304, New York, NY, USA, 2007. ACM.

[dVHdP04] Henny de Vos, Henri Hofte, and Henk de Poot. IM [@work]: Adoption of Instant Messaging in a Knowledge Worker Organisation. *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 10–19, January 2004.

[Ebe08] Jan Christoph Ebersbach. *Entwicklung eines Webbrowser-basierten Wiedergabe- und Steuerungssystems für Video- und Hypermedia-Anwendungen*. Diploma thesis, Universität Bremen, March 2008.

[EF94] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631 (Informational), IETF, May 1994. Obsoleted by RFC 3022.

[EG01] W. Keith Edwards and Rebecca E. Grinter. At Home with Ubiquitous Computing: Seven Challenges. In *Ubicomp 2001: Ubiquitous Computing*, volume 2201 of *Lecture Notes in Computer Science*, pages 256–272. Springer Berlin / Heidelberg, 2001.

[EHM+07] Ryan Eatmon, Joe Hildebrand, Jeremie Miller, Thomas Muldowney, and Peter Saint-Andre. Data Forms. XEP-0004 (Final), XSF, August 2007.

[Ell02] Carl M. Ellison. Home Network Security. *Intel Technology Journal*, 6(4):37–48, November 2002.

[Ell03a] C. Ellison. UPnP Security Ceremonies V1.0. Technical report, UPnP Forum, October 2003.

[Ell03b] Carl Ellison. DeviceSecurity:1. UPnP Device Template, UPnP Forum, November 2003.

[Ell03c] Carl Ellison. Security Console:1 Service Template. UPnP Device Template, UPnP Forum, November 2003.

[Epp05] Jeffrey L. Eppinger. TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem. Technical report, January 2005.

[ERS02] D. Eastlake 3rd, J. Reagle, and D. Solo. (Extensible Markup Language) XML-Signature Syntax and Processing. RFC 3275 (Draft Standard), IETF, March 2002.

[Gan08] A. K. R. Ganga. Distributed Algorithmic Mechanism Design in P2P File-Sharing Networks. November 2008.

[GB08] Saul Greenberg and Bill Buxton. Usability Evaluation Considered Harmful (Some of the Time). In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 111–120, New York, NY, USA, 2008. ACM.

[GCA+99] Yaron Y. Goland, Ting Cai, Shivaun Albright, et al. Simple Service Discovery Protocol/1.0. Internet Draft (Expired), IETF, October 1999. draft-cai-ssdp-v1-03.

[GHM+07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, et al. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C recommendation, April 2007. `http://www.w3.org/TR/soap12-part1/`.

[GMICL06] M. Garcia-Martin, M. Isomaki, G. Camarillo, and S. Loreto. A Mechanism to Enable File Transfer with the Session Initiation Protocol (SIP). Internet Draft (Work in Progress), IETF, February 2006. draft-garcia-sipping-file-transfer-mech-00.

[Gol99] Yaron Y. Goland. Multicast and Unicast UDP HTTP Messages. Internet Draft (Expired), IETF, October 1999. draft-goland-http-udp-01.

[GP02] Rebecca E. Grinter and Leysia Palen. Instant Messaging in Teen Life. In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 21–30, New York, NY, USA, 2002. ACM.

[Gut01] Erik Guttman. Autoconfiguration for IP networking: enabling local communication. *IEEE Internet Computing*, 5(3):81–86, May/June 2001.

[Gut03] Peter Gutmann. Plug-and-play PKI: a PKI your mother can use. In *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, pages 4–4, Berkeley, CA, USA, 2003. USENIX Association.

[GVE00] A. Gulbrandsen, P. Vixie, and L. Esibov. A DNS RR for specifying the location of services (DNS SRV). RFC 2782 (Proposed Standard), IETF, February 2000.

[Had07] Leslie Haddon. Looking for Diversity: Children and Mobile Phones. In *Mobile Media: International Conference on Social and Cultural Aspects of Mobile Phones, Convergent Media, and Wireless Technologies*. The University of Sydney, 2007.

[HAP+09] A. Houri, E. Aoki, S. Parameswar, T. Rang, V. Singh, and H. Schulzrinne. Presence Interdomain Scaling Analysis for SIP/SIMPLE. Internet Draft (Work in

Progress), IETF, August 2009. draft-ietf-simple-interdomain-scaling-analysis-08.

[Haq07] Abul Ahsan Md. Mahmudul Haque. UPnP networking: Architecture and security issues. In *Security of the End Hosts on the Internet: Seminar on Network Security Autumn 2007*, 2007.

[Har08] Klaus Hartke. API-Usability am Beispiel einer XMPP-Bibliothek. Independent Study at the Universität Bremen, September 2008.

[HB06] Ernst Haselsteiner and Klemens Breitfub. Security in Near Field Communication: Strengths and Weaknesses. In *Workshop on RFID security*, pages 1–13, July 2006.

[HB09] Klaus Hartke and Carsten Borman. STUN/TURN using PHP in Despair. Internet Draft (Work in Progress), IETF, July 2009. draft-hartke-xmpp-stupid-00.

[Hem06] Armijn Hemel. Universal Plug and Play: Dead simple or simply deadly? In *5th System Administration and Network Engineering Conference*, 2006.

[HG07] A. Haber and M. Gerdes. Remote Service Usage Through SIP with Multimedia Access as a Use Case. *Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium on*, pages 1–5, September 2007.

[HGR+07] Andreas Häber, Martin Gerdes, Frank Reichert, Andreas Fasbender, and Ram Kumar. Remote Service Usage Through Sip with Multimedia Access as a Use Case. In *Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007. IEEE 18th International Symposium on*, pages 1–5, September 2007.

[HMESA08] Joe Hildebrand, Peter Millard, Ryan Eatmon, and Peter Saint-Andre. Service Discovery. XEP-0030 (Final), XSF, June 2008.

[HPEM08] Evan Henshaw-Plath and Kellan Elliott-McCrea. Beyond REST? Building data services with XMPP PubSub. In *OSCON 2008*, July 2008. `http://en.oreilly.com/oscon2008/public/schedule/detail/4359`.

[HSATK08] Joe Hildebrand, Peter Saint-Andre, Remko Tronçon, and Jacek Konieczny. Entity Capabilities. XEP-0115 (Draft), XSF, February 2008.

[Hui06] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), IETF, February 2006.

[HYAK+04] Erkki Harjula, Mika Ylianttila, Jussi Ala-Kurikka, Jukka Riekki, and Jaakko Sauvola. Plug-and-Play Application Platform: Towards Mobile Peer-to-Peer. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 63–69, New York, NY, USA, 2004. ACM.

[IW01] Prakash Iyer and Ulhas Warrier. Internet Gateway Device:1. UPnP Device Template, UPnP Forum, November 2001.

[JK03] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447 (Informational), IETF, February 2003.

[Jon07] Paul E. Jones. A Concept for the Advanced Multimedia System (AMS). July 2007.

[JSL+04] R. Jimeno, Z. Salvador, A. Lafuente, M. Larrea, and A. Uribarren. An architecture for the personalized control of domotic resources. In *EUSAI '04: Proceedings of the 2nd European Union symposium on Ambient intelligence*, pages 51–54, New York, NY, USA, 2004. ACM.

[Kas07] Marco Kasten. *Entwicklung einer konfigurierbaren Web-basierten Mediacenter-Plattform für heterogene Endgeräte*. Diploma thesis, Universität Bremen, October 2007.

[KHSAF09] Justin Karneges, Joe Hildebrand, Peter Saint-Andre, and Fabio Forno. Stream Management. XEP-0198 (Draft), XSF, June 2009.

[KO03] Dirk Kutscher and Jörg Ott. Dynamic Device Access for Mobile Users. In *In Proceedings of the 8th Conference on Personal Wireless Communications*, 2003.

[KO07] Vassilis Kostakos and Eamonn O'Neill. NFC on Mobile Phones: Issues, Lessons and Future Research. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 367–370, Washington, DC, USA, 2007. IEEE Computer Society.

[KPW06] Cynthia Kuo, Adrian Perrig, and Jesse Walker. Designing an Evaluation Method for Security User Interfaces: Lessons from Studying Secure Wireless Network Configuration. *interactions*, 13(3):28–31, 2006.

[KSA09] Justin Karneges and Peter Saint-Andre. In-Band Bytestreams. XEP-0047 (Draft), XSF, March 2009.

[Kut01] Dirk Kutscher. The Message Bus: Guidelines for Application Profile Writers. Internet Draft (Expired), IETF, February 2001. draft-ietf-mmusic-mbus-guidelines-00.

[Kut03] Dirk Kutscher. *Local Coordination for Interpersonal Communication Systems*. Doctoral thesis, Universität Bremen, December 2003.

[Lab09] Craig Labovitz. Who Put the IPv6 in my Internet? The Arbor Networks Security Blog, September 2009. `http://asert.arbornetworks.com/2009/09/who-put-the-ipv6-in-my-internet/`.

[Lam06] Butler Lampson. Practical Principles for Computer Security. August 2006.

[Lan96] Thomas K. Landauer. *Trouble with Computers: Usefulness, Usability, and Productivity*. MIT Press, Cambridge, MA, USA, 1996.

[Laz06] J. Lazzaro. Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport. RFC 4571 (Proposed Standard), IETF, July 2006.

[LBSA+09] Scott Ludwig, Joe Beda, Peter Saint-Andre, Robert McQueen, Sean Egan, and Joe Hildebrand. Jingle. XEP-0166 (Draft), XSF, June 2009.

[LLH+07] Deok-Gyu Lee, Yun-kyung Lee, Jong-wook Han, Jong Hyuk Park, and Im-Yeong Lee. Intelligent Home Network Authentication: Home Device Authentication Using Device Certification. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4804/2007 of *Lecture Notes in Computer Science*, pages 1688–1700. Springer Berlin / Heidelberg, 2007.

[Lov05] Robert Love. Get on the D-BUS. *Linux Journal*, 2005(130):3, 2005.

[LR+08] Gary Langille, John Ritchie, et al. ContentDirectory:3 Service Template. UPnP Device Template, UPnP Forum, September 2008.

[LSAE+09] Scott Ludwig, Peter Saint-Andre, Sean Egan, Robert McQueen, and Diana Cionoiu. Jingle RTP Sessions. XEP-0167 (Draft), XSF, June 2009.

[LSKD07] Jae Woo Lee, H. Schulzrinne, W. Kellerer, and Z. Despotovic. z2z: Discovering Zeroconf Services Beyond Local Link. In *Globecom Workshops, 2007 IEEE*, pages 1–7, Washington, DC,, November 2007.

[Mey08] Dirk Meyer. C2C Authentication Using TLS. XEP-0250 (Deferred), XSF, September 2008.

[Mey09a] Dirk Meyer. Client Certificate Management for SASL EXTERNAL. XEP-0257 (Experimental), XSF, February 2009.

[Mey09b] Dirk Meyer. Out-of-Band Stream Data. XEP-0265 (Experimental), XSF, April 2009.

[Mic06a] Microsoft Cooperation. Introduction to Windows Peer-to-Peer Networking. Microsoft TechNet Library, September 2006. `http://technet.microsoft.com/en-us/library/bb457079.aspx`.

[Mic06b] Microsoft Cooperation. Peer Name Resolution Protocol. Microsoft TechNet Library, September 2006. `http://technet.microsoft.com/en-us/library/bb726971.aspx`.

[Mil01] Jeremie Miller. Jabber—Conversational Technologies. In *Peer-to-Peer: Harnessing the Power of a Disruptive Technology*, pages 77–88. Sebastopol, March 2001.

[Moc87] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), IETF, November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343.

[MSA09a] Dirk Meyer and Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP) End-to-End Encryption Using Transport Layer Security (”XTLS”). Internet Draft (Work in Progress), IETF, June 2009. draft-meyer-xmpp-e2e-encryption-02.

[MSA09b] Dirk Meyer and Peter Saint-Andre. Management and Use of Client Certificates for the Extensible Messaging Presence Protocol (XMPP). Internet Draft (Work in Progress), IETF, March 2009. draft-meyer-xmpp-sasl-cert-management-01.

[MSAM09] Peter Millard, Peter Saint-Andre, and Ralph Meijer. Publish-Subscribe. XEP-0060 (Draft), XSF, October 2009.

[MSMNW09] A. Menon-Sen, A. Melnikov, C. Newman, and N. Williams. Salted Challenge Response (SCRAM) SASL Mechanism. Internet Draft (Work in Progress), IETF, May 2009. draft-newman-auth-scram-13.

[New04] Jan Newmarch. A Critique of Web Services. In *IADIS E-Commerce 2004*, 2004.

[New05] J. Newmarch. A RESTful approach: clean UPnP without SOAP. In *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE*, pages 134–138, January 2005.

[NWB00] Bonnie A. Nardi, Steve Whittaker, and Erin Bradner. Interaction and Outeraction: Instant Messaging in Action. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 79–88, New York, NY, USA, 2000. ACM.

[OK00] Jörg Ott and Dirk Kutscher. The Message Bus: A Platform for Component-based Conferencing Applications. In *In Proceedings of CBG2000: The CSCW2000 workshop on Component-based Groupware*, 2000.

[OK04] Jörg Ott and Dirk Kutscher. Persistent Connection Management Protocol. Drive-thru specification, July 2004. nodraft-tzi-pcmp-01.txt.

[OKM01] Jörg Ott, Dirk Kutscher, and Dirk Meyer. An Mbus Profile for Call Control. Internet Draft (Expired), IETF, February 2001. draft-ietf-mmusic-mbus-call-control-00.

[OM09] Toshihiro Ohigashi and Masakatu Morii. A Practical Message Falsification Attack on WPA. August 2009.

[OPK02] J. Ott, C. Perkins, and D. Kutscher. A Message Bus for Local Coordination. RFC 3259 (Informational), IETF, April 2002.

[OSC+09] Jörg Ott, Nils Seifert, Caleb Carroll, Nigel Wallbridge, Olaf Bergmann, and Dirk Kutscher. The CHIANTI Architecture for Robust Mobile Internet Access. In *Proceedings of the 10th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2009.

[Pat07] Ian Paterson. Stanza Encryption. XEP-0200 (Deferred), XSF, May 2007.

[Per06] Colin Perkins. *RTP: Audio and Video for the Internet*, page 8. Addison-Wesley, 2006.

[PH09] M. Petit-Huguenins. Alternative Proposal for Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations. Internet Draft (Work in Progress), IETF, March 2009. draft-petithuguenin-turn-tcp-variant-01.

[PL+06a] Alan Presser, Gary Langille, et al. AVTransport:2 service template. UPnP Device Template, UPnP Forum, May 2006.

[PL+06b] Alan Presser, Gary Langille, et al. MediaRenderer:2 service template. UPnP Device Template, UPnP Forum, May 2006.

[PL+06c] Alan Presser, Gary Langille, et al. MediaServer:2 service template. UPnP Device Template, UPnP Forum, May 2006.

[PL+06d] Alan Presser, Gary Langille, et al. ScheduledRecording:1 service template. UPnP Device Template, UPnP Forum, May 2006.

[PP08] Adrian Pasto and Petko D. Petkov. Hacking the Interwebs, 2008. `http://www.gnucitizen.org/blog/hacking-the-interwebs`.

[PR09] S. Perreault and J. Rosenberg. TCP Candidates with Interactive Connectivity Establishment (ICE). Internet Draft (Work in Progress), IETF, October 2009. draft-ietf-mmusic-ice-tcp-08.

[Pre01] Marc Prensky. Digital Natives, Digital Immigrants Part 1. *On The Horizon - The Strategic Planning Resource for Education Professionals*, pages 1–6, 2001.

[PSAM09] Ian Paterson, Peter Saint-Andre, and Dirk Meyer. Public Key Publishing. XEP-0189 (Experimental), XSF, March 2009.

[Rai07] Veli-Jussi Raitila. Tag, you're it - NFC in a home environment. In *Seminar on Internetworking*. Helsinki University of Technology, 2007.

[Ran02] Neil Randall. Lingo Online: A Report on the Language of the Keyboard Generation. 2002. In cooperation with MSN Canada.

[RET02] Yasser Rasheed, Jim Edwards, and Charlie Tai. Home Interoperability Framework for the Digital Home. *Intel Technology Journal*, 6(4):5–16, November 2002.

[RH03] Sandro Rafaeli and David Hutchison. A Survey of Key Management for Secure Group Communication. *ACM Computing Surveys*, 35(3):309–329, 2003.

[RK02] John Ritchie and Thomas Kuehnel. AVArchitecture:1. UPnP Device Template, UPnP Forum, June 2002.

[RK03] E. Rescorla and B. Korver. Guidelines for Writing RFC Text on Security Considerations. RFC 3552 (Best Current Practice), IETF, July 2003.

[RL08] A. B. Roach and B. B. Lowekamp. A Proposal to Define Interactive Connectivity Establishment for the Transport Control Protocol (ICE-TCP) as an Extensible Framework. Internet Draft (Work in Progress), IETF, October 2008. draft-lowekamp-mmusic-ice-tcp-framework-00.

[RM01] Arjun Roychowdhury and Stan Moyer. Instant Messaging and Presence for SIP Enabled Networked Appliances. In *IP Telephony Workshop*, Columbia University, New York, April 2001.

[RMM09] J. Rosenberg, R. Mahy, and P. Matthews. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). Internet Draft (Work in Progress), IETF, July 2009. draft-ietf-behave-turn-16.

[RMMW08] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), IETF, October 2008.

[Ros01a] M. Rose. On the Design of Application Protocols. RFC 3117 (Informational), IETF, November 2001.

[Ros01b] M. Rose. The Blocks Extensible Exchange Protocol Core. RFC 3080 (Proposed Standard), IETF, March 2001.

[Ros06] Jonathan Rosenberg. Interactive Connectivity Establishment. In *IETF Journal, Volume 2 Issue 3*, pages 14–19. IETF, November 2006.

[Ros07a] M. Rose. Reclassification of the APEX RFCs to Historic. Internet Draft (Work in Progress), IETF, May 2007. draft-mrose-apex-historic-02.

[Ros07b] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. Internet Draft (Work in Progress), IETF, October 2007. draft-ietf-mmusic-ice-19.

[RR02] Yasser Rasheed and John Ritchie. High-Quality Media Distribution in the Digital Home. *Intel Technology Journal*, 6(4):17–29, November 2002.

[RS05] Andrew Roczniak and Abdulmotaleb El Saddik. JADE: jabber-based authoring in distributed environments. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 279–282, New York, NY, USA, 2005. ACM.

[RS06] J. Rosenberg and H. Schulzrinne. Guidelines for Authors of Extensions to the Session Initiation Protocol (SIP). RFC 4485 (Informational), IETF, May 2006.

[RSC+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), IETF, June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630.

[Rém09] Mickaël Rémond. XMPP in real life: attacks, bad behaviour and how to cope with them. In *FOSDEM 2009*, February 2009.

[SA04a] P. Saint-Andre. End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP). RFC 3923 (Proposed Standard), IETF, October 2004.

[SA04b] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 3920 (Proposed Standard), IETF, October 2004.

[SA04c] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 3921 (Proposed Standard), IETF, October 2004.

[SA07] Peter Saint-Andre. Jingle: Jabber Does Multimedia. *IEEE MultiMedia*, 14(1):90–94, 2007.

[SA08a] Peter Saint-Andre. Serverless Messaging. XEP-0174 (Final), XSF, November 2008.

[SA08b] Peter Saint-Andre. XMPP Extension Protocols. XEP-0001 (Active), XSF, January 2008.

[SA09a] Peter Saint-Andre. Jingle File Transfer. XEP-0234 (Experimental), XSF, February 2009.

[SA09b] Peter Saint-Andre. Requirements for End-to-End Encryption in the Extensible Messaging and Presence Protocol (XMPP). Internet Draft (Work in Progress), IETF, August 2009. draft-ietf-xmpp-e2e-requirements-00.

[SAi08] Peter Saint-Andre and Pavel imerda. Bits of Binary. XEP-0231 (Draft), XSF, September 2008.

[SAM07] Peter Saint-Andre and Peter Millard. Best Practices for Use of SASL EXTERNAL with Certificates. XEP-0178 (Active), XSF, February 2007.

[SAMKH09] Peter Saint-Andre, Dirk Meyer, Justin Karneges, and Klaus Hartke. Jingle SOCKS5 Bytestreams Transport Method. XEP-0260 (Experimental), XSF, July 2009.

[SAS09] Peter Saint-Andre and Kevin Smith. Personal Eventing Protocol. XEP-0163 (Draft), XSF, October 2009.

[SC05] Daniel Steinberg and Stuart Cheshire. *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., 2005.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), IETF, July 2003. Updated by RFC 5506.

[Sco02] Scott Lawrence. Universal plug and play device architecture. UPnP Device Template, UPnP Forum, December 2002.

[Sel06] Kristian Selén. UPnP security in Internet gateway devices. In *Seminar on Internetworking*, 2006.

[SMSA07] Dave Smith, Matthew Miller, and Peter Saint-Andre. SOCKS5 Bytestreams. XEP-0065 (Draft), XSF, May 2007.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326 (Proposed Standard), IETF, April 1998.

[STW96] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security*, pages 31–37, New York, NY, USA, 1996. ACM.

[SV03] H. A. Schotanus and C. A. A. Verkoelen. Extended home environment from a security perspective. *X-home deliverable*, 2003.

[SVA07] Jani Suomalainen, Jukka Valkonen, and N. Asokan. Security Associations in Personal Networks: A Comparative Analysis. In *Security and Privacy in Adhoc and Sensor Networks*, volume 4572 of *Lecture Notes in Computer Science*, pages 43–57. Springer, 2007.

[TAA+03] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean christophe Hugly, Eric Pouyoul, and Bill Yeager. Project JXTA 2.0 Super-Peer Virtual Network. Technical report, 2003.

[Tec94] Technical Committee ISO/TC159. ISO DIS 9241 Part 11: Guidance on Usability. ISO/IEC, International Organization for Standardization, 1994.

[Tec07] Technical Committee JTC1/SC6. ISO/IEC 18092:2004 – Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1). ISO/IEC, International Organization for Standardization, 2007.

[Tie08] Melanie Tietjen. *Persönliche mobile Geräte als plattform- und netzunabhängige Fernbedienung für verteilte Anwendungen*. Diploma thesis, Universität Bremen, December 2008.

[TWMP07] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the Secure Remote Password (SRP) Protocol for TLS Authentication. RFC 5054 (Informational), IETF, November 2007.

[UKA08] Ersin Uzun, Kristiina Karvonen, and N. Asokan. Usability Analysis of Secure Pairing Methods. In *Financial Cryptography and Data Security*, volume 4886 of *Lecture Notes in Computer Science*, pages 307–324. Springer Berlin / Heidelberg, 2008.

[VTK07] Jukka Valkonen, Aleksi Toivonen, and Kristiina Karvonen. Usability Testing for Secure Device Pairing in Home Networks. In Anne Bajart, Henrik Muller,

and Thomas Strang, editors, *UbiComp 2007 Workshop Proceedings, September 2007, Innsbruck, Austria*, 2007.

[Wit05] Georg Wittenburg. Desktop IPC. Technical report, February 2005.

[WPL08] James Walkerdine, Peter Phillips, and Simon Lock. Architecting secure mobile P2P systems. In *SAM '08: Proceedings of the 1st international workshop on Software architectures and mobility*, pages 9–14, New York, NY, USA, 2008. ACM.

[WT99] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: a usability evaluation of PGP 5.0. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[Wöl05] Thomas Wölfl. Public-Key-Infrastructure Based on a Peer-to-Peer Network. In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 7*, page 200.1, Washington, DC, USA, 2005. IEEE Computer Society.

[YMH06] Kiyohito Yoshihara, Toru Maruta, and Hiroki Horiuchi. A zeroconf approach to secure and easy-to-use remote access to networked appliances. In Young-Tak Kim and Makoto Takano, editors, *APNOMS*, volume 4238 of *Lecture Notes in Computer Science*, pages 352–361. Springer, 2006.

All URIs in this document were last checked on November 12, 2009.

According to their status section Internet Drafts are cited as "Work in Progress" and a newer version might be available. Nevertheless, Internet Drafts expired over more than five years ago are marked as "Expired" instead of "Work in Progress".

Appendix A

Internet Drafts and XMPP Extension Protocols

This appendix lists all XMPP extension that were officially published by the XMPP editor as well as the Internet Drafts submitted to the IETF. Other specifications such as HTTP over XMPP were only discussed within the XMPP community but never submitted as ProtoXEP. The same is true for the profiles described in chapter 7; they are for evaluation purposes only.

A.1 XMPP End-to-End Security

The important part of the XPMN core architecture is the security layer based on TLS and SASL EXTERNAL. The following specifications were sent to the XMPP editor for publishing or submitted to the IETF as Internet Draft.

ProtoXEP: XMPP Transport Layer Security

Authors: Dirk Meyer, Peter Saint-Andre, and Justin Karneges
Version: 0.0.5; superseded by draft-meyer-xmpp-e2e-encryption
Last Updated: 2008-12-11

Abstract: This document specifies the XMPP Transport Layer Security (XTLS) extension. XTLS, which provides communications privacy for the Extensible Messaging and Presence Protocol (XMPP), enables XMPP applications to communicate in a way that is designed to prevent eavesdropping, tampering, and forgery of XMPP stanzas. XTLS is based on Transport Layer Security (TLS) and provides equivalent security guarantees. The protocol sends standard TLS handshake and application data encoded as Base64, similar to the XMPP In-Band Bytestreams (IBB) extension but qualified by a dedicated namespace.

XEP-0247: Jingle XML Streams

Authors: Peter Saint-Andre, Justin Karneges, and Dirk Meyer
Version: 0.2; superseded by draft-meyer-xmpp-e2e-encryption
Last Updated: 2009-02-20

Abstract: This specification defines a Jingle application type for establishing direct or mediated XML streams between two entities over any streaming transport. This technology thus enables two entities to establish a trusted connection for end-to-end encryption or for bypassing server limits on large volumes of XMPP traffic.

XEP-0250: C2C Authentication Using TLS

Authors: Dirk Meyer
Version: 0.2; to some extent superseded by draft-meyer-xmpp-e2e-encryption
Last Updated: 2008-09-08

Abstract: This document describes how to negotiate which TLS extension to use for TLS-based end-to-end XML streams between two clients. It covers X.509 certificates with and without a CA, the use of OpenPGP, Shared Remote Passwords (SRP) and how to bootstrap a trust relationship.

Extensible Messaging and Presence Protocol (XMPP) End-to-End Encryption Using Transport Layer Security ("XTLS")

Authors: Dirk Meyer and Peter Saint-Andre
Name: draft-meyer-xmpp-e2e-encryption-02
Submitted: June 29, 2009

Abstract: This document specifies "XTLS", a protocol for end-to-end encryption of Extensible Messaging and Presence Protocol (XMPP) traffic. XTLS is an application-level usage of Transport Layer Security (TLS) that is set up using the XMPP Jingle extension for session negotiation and transported using any streaming transport as the data delivery mechanism. Thus XTLS treats the end-to-end exchange of XML stanzas as a virtual transport and uses TLS to secure that transport, enabling XMPP entities to communicate in a way that is designed to ensure the confidentiality and integrity XML stanzas. The protocol can be used for secure end-to-end messaging as well as other XMPP applications, such as file transfer.

XEP-0189: Public Key Publishing

Authors: Ian Paterson, Peter Saint-Andre, and Dirk Meyer
Version: 0.9
Last Updated: 2009-03-08

Abstract: This specification defines a method by which an entity can publish its public keys over XMPP. It supports key distribution using pubsub as well as a direct key exchange.

XEP-0257: Client Certificate Management for SASL EXTERNAL

Authors: Dirk Meyer
Version: 0.2; superseded by draft-meyer-xmpp-sasl-cert-management
Last Updated: 2009-02-12

Abstract: This specification defines a method to manage client certificates that can be used with SASL EXTERNAL to allow clients to log in to the server without knowing the account password.

Management and Use of Client Certificates for the Extensible Messaging and Presence Protocol (XMPP)

Authors: Dirk Meyer and Peter Saint-Andre
Name: draft-meyer-xmpp-sasl-cert-management-01
Submitted: March 8, 2009

Abstract: This document defines methods for managing and using client certificates in the Extensible Messaging and Presence Protocol (XMPP). These methods, which make use of the EXTERNAL mechanism of the Simple Authentication and Security Layer (SASL) protocol, enable an XMPP client to log in to an XMPP server without providing a password.

A.2 Jingle Transports and Applications

Abstract: Additional to the XPMN core, a Jingle SOCKS5 transport and a Jingle application for transmitting out-of-band data directly between peers was specified (see section 7.1.1 and section 7.1.3 for details). Both specifications were submitted to the XMPP editor and approved by the XMPP Council.

XEP-0260: Jingle SOCKS5 Bytestreams Transport Method

Authors: Peter Saint-Andre, Dirk Meyer, Justin Karneges, and Klaus Hartke
Version: 0.3
Last Updated: 2009-07-14

Abstract: This specification defines a Jingle transport method that results in sending data via the SOCKS5 Bytestreams (S5B) protocol defined in XEP-0065. Essentially this transport method reuses XEP-0065 semantics for sending the data and defines native Jingle methods for starting and ending an S5B session.

XEP-0265: Out-of-Band Stream Data

Authors: Dirk Meyer
Version: 0.1
Last Updated: 2009-04-02

Abstract: This specification defines how to send parts of an XML stream over a direct connection between peers. This allows to send large stanzas or binary data without blocking the XML stream for other stanzas.