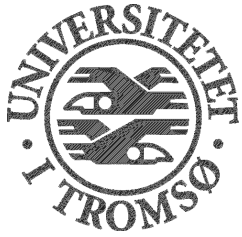


INF-3996

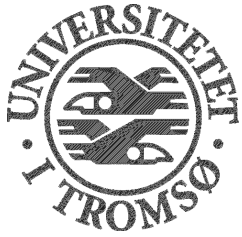
Master's thesis in Telemedicine & E-health

THE RELIABILITY OF XMPP FOR FILE TRANSFER

Kristian Andreassen

06 – 2008

Faculty of science –
Department of computer science
University of Tromsø



INF-3996

Master's thesis in Telemedicine & E-health

THE RELIABILITY OF XMPP FOR FILE TRANSFER

Kristian Andreassen

06 – 2008

Faculty of science –
Department of computer science
University of Tromsø

Preface

This thesis is the final part of my master's degree in Telemedicine and E-health. After several years as a student it feels very good to be almost finished and ready to focus on establishing a working career. This thesis was a result of discussions with my tutor about potential problems and areas that could be looked into by a master's student. The technology in question has been a new territory to me, but it has been very interesting to work with something that hasn't got that many years of evolution. It has also been an interesting perspective working on the subject of this thesis since my position as an EDI (Electronic Data Interchange) consultant at Helse Nord IKT gives me an insight into what solutions is necessary to provide the best possible service. I have also seen other areas where the technology I have studied could be utilized, and this will be discussed further in the final chapter of this thesis.

I believe the discussed protocol has many undiscovered potentials and its development will continue in the years to come.

Acknowledgement

There are several people that deserve to be thanked for their support during my work on this thesis.

I would especially like to thank my supervisor Johan Gustav Bellika, for his guidance during my work. His effort has been crucial to my progress during this semester, and it has been an educational session under his wings. Special thanks also go to Anders Baardsgaard at NHN, for his help in establishing a test environment to perform the experiments I needed.

Several people at Helse Nord IKT deserve to be thanked for the help I have been given during my experiments.

- Terje Bless for giving me the overview of administrative and organizational tasks needed to be addressed when implementing and deploying new services in the environment at hand.
- Torstein Meyer for helping with network engineering and
- Steve-Arne Bentzen for his help in setting up test components at Helgelandssykehuset.
- Lars-Andreas Wikbo and Vegard Jørgensen for their input to discussions and for sharing their knowledge of existing solutions and services related to the field of file transfer.

Helse Nord IKT as my employer also has to be mentioned and thanked for giving me the possibility of 50% leave of absence during these 5 months to complete my work on this thesis. My co-workers at the EDI department also deserve to be thanked for their extra effort during the period I have been absent.

My good friend Cato Sivertsen also needs to be mentioned and thanked for his incredible endurance in helping me with the 96 hours of durability testing that was a marathon in file transfer.

I would also like to thank some of the people from the XMPP / Jabber community for their advice and feedback, especially Artur Hefczyc at tigase.org and John Erics from Saddlers. It has been of great value to my work. A special thank goes to Torsten H (Torque) for his large amount of critic feedback and for always asking *why.*

Finally I would like to thank my friends and my family backing me up and supporting me during the hardest periods of writers block.

Summary

The use of advanced technology in medical systems and applications increase every year, and the data produced by these solutions is increasing in both size and number. This data can be any kind of medical information stored in a variety of formats from small XML based discharge letters to large medical images produced by radiology systems. The possibility of transfer of this data between the health care providers is important in a time where most information is available electronically. File transfer is one of the basic services that need to be provided by a network dedicated to telemedicine and e-health use. Store and forward solutions has so far been based on protocols as SMTP/POP, FTP and HTTP. However, the growth in number of institutions connected to the Norwegian Health Network and the amount of network traffic between the health institutions within the network has currently raised the need for better, more reliable and scalable solutions for file transfer.

This thesis investigates the use of XMPP with extensions for file transfer, to establish a reliable service for transfer of large files within a health dedicated network. We have established a test environment within a health network and have defined a set of measurements that will be answered in this thesis. We have also set up our test environment between two independent health institutions to get measurements from a production environment. Our assessment of the results is compared to both the existing solution and other potential solutions based on different technology.

Based on our discoveries we have also designed a telemedicine solution for file transfer based on XMPP and the benefits that can be drawn from the XMPP technology. The design consists of a description of the functionality that can be implemented into such a solution based on what elements that can be utilized from the XMPP base and extended functionality.

Contents

Figure list

Table list

1 Introduction

This chapter will give an introduction to the rest of this thesis. It consists of background and problem definition alongside a short introduction to the technological aspects discussed in this thesis. Short overviews of the main results are followed by a chapter overview.

1.1 Background

The Norwegian health-network (NHN) is the main supplier of infrastructure and communication services between the primary- and the specialist healthcare providers in Norway. The most important form of electronic collaboration is communication of medical information to and from the hospitals. Examples of data in this communication are discharge letters, lab results, referrals and radiology results. This communication is today mostly performed by using the well known email protocols POP and SMTP. New and potentially better technology and designs have been developed, and one of them is the Extensible Messaging and Presence Protocol (XMPP).

Through years of experience with POP/SMTP as basis for communication, system administration personnel have experienced problems regarding large attachments when communicating using POP/SMTP[1]. As systems and solutions in healthcare evolve, the production of high resolution images and advanced medical documents increase in use. Transmitting these documents and data files to the communication parties will of course be an important part of its use. Usually one could limit the problems with this transmission, by increasing the capacity of the communication channel. However, new technologies and new protocols have been developed, and might be able to deliver better and extended opportunities to the users.

The XMPP protocol is an open XML technology for real-time communication[2]. It includes a wide range of applications such as Instant Messaging (IM) [3], presence, media negotiation and generalized XML routing. Some of these features might be beneficial in the development of efficient and secure communication within the health network. The protocol also defines new and interesting options as the aspect of communicating “presence”. There could be undiscovered possibilities regarding the use of XMPP, and this could be looked into as one of the possible next steps in our use of communication solutions. By taking technology to the next step, one could also benefit from economical advantages when it comes to communication. Further on our question became whether implementation of the XMPP protocol could be a more cost-efficient solution.

1.2 Problem definition

The use of e-mail protocols (POP/SMTP) for transferring large file attachments might in many cases produce problems, since the communication setup uses a server to store the messages (e-mails) until the receiving end collects them. If the transmissions includes large file attachments, these files will also be stored on the server, awaiting client pickup. In principle this doesn't produce a problem, until a given amount of data is stored on the server. If the recipient connects and download these messages continuously it shouldn't produce any problems, hence the messages is deleted after they have been downloaded. The problem arises if the recipient hasn't connected in a while, and the server's message pool has reached sizes that far exceed a "normal" amount. The client could then, depending on bandwidth and size of attachments, be occupied downloading these attachments for a long time. There is no prioritizing of the messages, as they are downloaded sequentially until the server queue is empty. Given that there are no interruptions due to network faults or software malfunction, this solution works and will succeed with its intended task.

When this scenario is put into the healthcare scene, there are other issues that make the use of POP/SMTP as the communication channel for large files, inadequate. Communication between hospitals and primary healthcare is complex, and consists of many different types of messages. The need for communicating messages with high priority, like emergency referrals immediately eliminates the possibility of having an e-mail client occupy the transmission channel with a download of a large attachment. One could discuss the option of establishing a communication setup where transfer of large file attachments could be run in a separately e-mail solution. However this would result in higher need for resources, both economic and in the form of operating personnel. Since the need for transferring large attachments has emerged, a solution for providing a reliable service to perform these operations is necessary. Protocols designed for transferring large files exist, where FTP [4] is the best known and most frequently used protocol for this task. To use FTP for such a solution will demand a design around a client –server setup [4] through the entire communication chain. Setting up a large amount of FTP servers to provide for this communication solution could show to be a cost increasing task, along with huge increase in demand of operating resources.

When asking for improvement in cost-efficiency, this has to be seen from the view of the organization that operates solutions within the NHN. Could the organization achieve improvements in operating costs measured in time, resources and finances? All of these issues should be taken into consideration when looking into new and unknown technological solutions. It is also important to show what level of influence the changes will have on the different sections of the organization. One might achieve improvements in financial measurements, but then loose essential resources in maintaining a solution that is technologically inferior to the existing one. New technologic solutions might not necessarily be an improvement in all areas of its intended use. The main focus should be on its reliability in performing its main tasks, in this case, providing reliable, secure and effective

exchange of large files between communications points within a health dedicated network. To achieve a certainty that these demands are met, one has to perform tests and measurements of the intended solution to make sure it meets the demands set by both users and the system administrators.

The main problem definition of this thesis is to measure the *"Reliability of XMPP as a solution to transfer large files within a health dedicated network".*

The XMPP technology is based on streaming XML [5] and a large amount of extensions have been developed to the base protocol [6]. Some of these extensions have been aimed at extending the XMPP technology to enable file transfer. Two of these extensions are XEP – 0047 In-band bytestream [7] and XEP – 0065 Socks 5 Bytestreams [8].

Can any of these two extensions be used together with a XMPP based client/server setup to provide a reliable file-transfer solution? If the technology can provide benefits that exceed the drawbacks, and if file transfer performance is satisfactory, could there be developed a file transfer solution based on XMPP? Based on knowledge gathered on the XMPP technology, the operating environment and what can be found about possible existing solutions, we will propose a functionality design of a telemedicine solution for file transfer. Due to security and maintainability-issues, health network has an infrastructure that limits the possibilities of implementing new solutions/technologies. One has to comply with the rules and regulations of the health network and maintainability routines. The implementation of an XMPP based solution within the Health Network (HN) structure will have to comply with these rules and regulations, and will be discussed as a part of the conclusion.

To perform the measurements intended, we need to set up a test environment that replicates the actual operating environment. The tests will be performed on a communication setup running within the HN. The two extensions mentioned earlier have different designs in both intended use and communication setup. Before starting a test setup we investigated prior use of these extensions, and what possibilities and limitations exist for the two designs. A successful setup of the test environment consists of the communication solutions represented by these extensions. The sheer principle of the communication solution has also been tested to see if it works within the framework of an existing network. Through the tests we focused on some main measures of success/failure

- Mean time between failure
- Scalability
- Resilience to errors
- Efficiency

Maintainability seen from system administration point of view is also a very important and highly debatable issue that we address.

1.3 Method

The method used in this thesis is based on the quantitative approach. Data was collected from the different tests and was analyzed against the preset measurements given in the requirements specification. To get the necessary data for analysis and measurement comparison a series of file transfer experiments was performed within an operating environment as close to the real setup as possible. A placement of test computers was arranged with Helse Nord IKT and the Norwegian Health Network in Tromsø.

The software and applications used to collect the test data was already existing XMPP clients and servers available to the general public. Since none of the existing clients or servers has been developed with file transfer as their service focus, one could argue that a development of such software should have been done to make the tests as correct as possible. To some extent this is a correct notion, but the implementations of the two extensions tested in this thesis have, according to the developers, been done exactly according to the specifications developed by the XMPP Standards Foundation (XSF).

Another argument for implementation of dedicated client and server functions to perform these experiments, is that it could enable tweaking and adjustment of settings that might affect the performance of the technology. By developing dedicated software, the implementation could be focused on providing the best possible utilization for file transfer. The use of software developed by others also makes the fail rate somewhat obscure. One cannot be sure that the implementation of the client and server software has flaws in its code. This could affect the overall performance of the experiments, but also be a direct reason for several of the errors experienced during the experimental file transfers. Implementation of the proposed XEP´s is also an element of uncertainty. Even though all of the published XEP´s are released as official part of the XMPP technology, they might have several areas where optimization is possible.

The data collection performed during the experiments has been a manual process. The use of existing Instant Messaging (IM) software makes this the easiest option for establishing a file transfer session between the clients. Most XMPP clients have a XML console that gives the user complete overview of all XML segments handled by the client. This console is the feature used for timing the transfers during all experiments. The timing of the transfers has been done on the receiving end, by using the XML console. The console gives us the timestamp of every XML element during the transfer session. We have registered the first and last element in the transfer session providing us with the best possible timing of the transfers. By using the console at the receivers end we got the exact time for each session. The console also keeps a log of recent the events making it possible to monitor the transfer in detail.

One could argue that the timing shouldn't have been performed on the same computer and in the same client that performs the actual file transfer, due to possible influences from the coherent communication partners. This would be a valid argument if the time periods in questions where much smaller. Our transfers have a minimum time consumption of about 1 second for the smallest file, and this is not a time period that in any way would be influenced unnoticeable by any exterior elements.

The drawback of performing the file transfers manually is the level of stress possible to expose the software for. During scalability tests it is desirable to keep continuously transfers running through the server to keep the load as heavy as possible. By performing manual transfer initiations we are not able to do this on the smallest file formats. Manually we were able to start a transfer every 5 seconds at the best. For the larger files this isn't an issue due to the total transfer time for each file.

It has been difficult to find relevant published material on the subject that has been focus in this thesis. XMPP is not a technology that has been focused on providing file transfer, hence the lack of scientific material on related and similar work. This has resulted in the need for using personal communication as part of the references for background material.

1.4 Scope and limitations

When working on experimental issues related to new technology and new solutions it is important to stay within the boundaries of a pre-set framework. One might quickly move outside the scope of what the intended task demands and requires. In my experiment the focus will be to check the reliability of the XMPP file transfer solution, set in the above mentioned environment. Any other issue that has risen during this process that is not directly involved in my problem definition has not been taken into any further consideration. They are mentioned and explained but not further investigated. Some of the obstacles I have met during my tests and experiments might not be present in another operating environment. My view has been narrowed towards the reliability of the XMPP extensions for file transfer within a health network.

1.5 Main results

The experiments performed in this thesis work show that the XMPP protocol is a feasible technology for the intended solution outlined in Chapter 1.2. There are advantages and drawbacks to the two extensions tested, and these have been explained further in the discussion in Chapter 6. Implementing a solution based on XMPP that is dedicated for file transfer, could provide services that would solve several of the problems related to transfer of large files in the given environments. The experiments performed in this thesis show that XMPP is a feasible solution for transferring large files within the framework intended. The protocol also delivers several features that could be utilized for other solutions within the field of telemedicine and e-health.

1.6 Outline

The structure of this thesis is as follows:

Chapter 2 will present the theory surrounding the XMPP technology and its relevant extensions. The existing solutions for file transfer will be presented and discussed in minor detail.
Chapter 3 describes the requirement specification developed for performing the intended experiments. Both external and internal requirements are uncovered and assembled for use in design and implementation of the experiments.
Chapter 4 presents the design and implementation issues of the intended experiments. Also describes what choices were done during this process.
Chapter 5 presents the test and measurement results obtained during the experiments.
Chapter 6 contains discussion of design, implementation, and results collected during the experiments. The chapter also contains conclusion and ending words.
Chapter 7 presents some of the future possibilities and potential areas of development using XMPP.

1.7 Summary

This chapter has given an overview of this thesis, the background and problem definition, description of methods used and a quick presentation of the main results achieved in the experiments performed. The chapter also lists the outline of the following pages of the report.

2.0 Theoretical fundaments and background

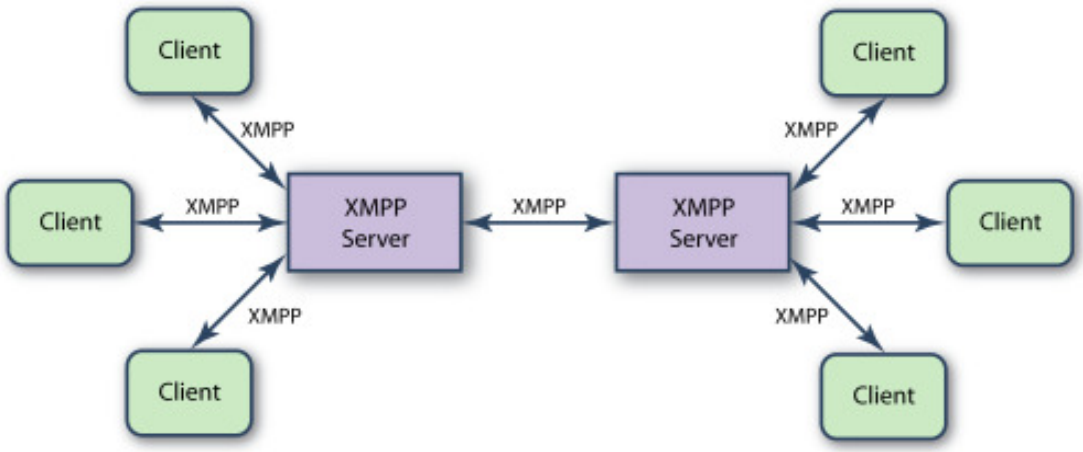
This chapter will give some background information about the technologies discussed and used in this thesis work. The first part will be giving an introduction to XMPP and the functionality provided by the protocol, focusing on the application of file transfer. Also the counterpart of XMPP, e-mail protocols SMPT/POP will be presented with a perspective on the problem definition given for this thesis.

2.1 Technology

The Extensible Messaging and Presence Protocol (XMPP) was developed [9] as a formalization of the base XML streaming protocols for instant messaging and presence, developed within the Jabber community and started in 1999. Jabber [2] is a technology based on XMPP focusing on instant messaging and presence functionality. It has evolved into a huge community contributing to development of clients and servers supporting the Jabber and XMPP principles. The Internet Engineering Task Force (IETF) has formalized the core XML streaming protocols as an approved instant messaging and presence technology under the name of XMPP. The specifications has been published as RFC 3920 and RFC 3921 [2].

Opposed to other instant messaging protocols, XMPP is based on open standards. Most of the Jabber server implementations and clients are also free and open source software.

Figure 1 Overview of XMPP com setup (http://www.isode.com/whitepapers/xmpp.html)



2.1.1 Extensibility

Quote from – www.xmpp.org [10]

> *"The XMPP Standards Foundation (XSF) develops extensions to XMPP through a standards process centered around XMPP Extension Protocols (XEPs). The process is managed by the XMPP Extensions Editor and involves intensive discussion on the Standards mailing list, formal review and voting by the XMPP Council, and modification based on implementation experience and interoperability testing. All documents in the XEP series are available under a liberal IPR Policy for wide implementation. "*

Several extensions have been developed to the XMPP base functionality. This provides the technology with an element of continuous evolvement and makes it scalable and dynamic. Since it is based on the principle of open source software, it makes it a lot easier for external developers to aid in its development. Suggestions for improvements and additional functionality can be submitted to the XSF for approval.

There have been developed two extensions offering binary data streams, or file transfer, to the XMPP base protocol. The two extensions are described in XEP – 0047 In-Band byte stream(IBB) [7] and in XEP – 0066 Out of Band Data(OOB) [11]. One of the possibilities looked into in this thesis is the XEP-0065 Socks 5 extension, where the data stream is initiated through a proxy server that sets up and handles the transmission based on the OOB principle. XEP-0047 and XEP-0065 that is the basis for the problem definition is this thesis, has been developed and approved by the XSF as official extensions.

2.1.2 RTC – Real Time Collaboration server

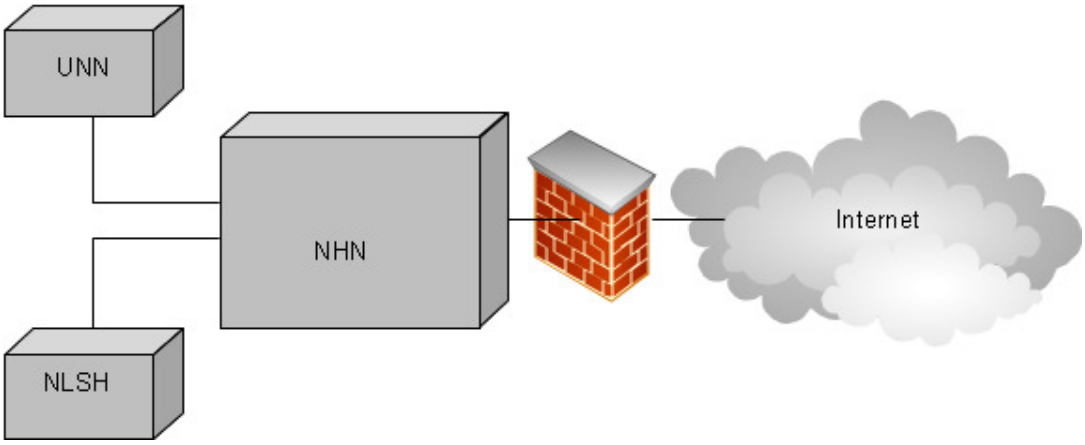
The core of the Jabber communication setup is dependent upon a RTC (Real Time Collaboration) server. There are many available servers designed for Jabber and instant messaging. Some of them are designed exclusively for instant messaging, while others try to achieve a more overall usability score. One can run a single server for a simple local communication network, or the servers can be linked together with others, to form a huge global network of servers. By linking them together, the clients will have the choice of connecting to a server closer to their location to prevent network connectivity issues. During the work on this thesis we have been taking a closer look at Openfire [12] ,eJabberd [13] and Tigase [14]. One of these have been chosen for as the RTC for our experiments . All of these vendors are well known and popular releases of Jabber RTC servers.

2.1.3 Operating environment

A Health network (HN) can be seen as an Internet Service Provider (ISP) for the health regions in Norway, where its main purpose is to provide services for communication between primary healthcare and the specialist healthcare providers. One of the most important forms of communication is the transfer of medical information to and from the hospitals. Examples of data in this communication are discharge letters, lab results, referrals and radiology results. This communication is performed today using the well known protocols POP and SMTP. Due to the sensitivity of data communicated within the framework of the HN, there are strict safety measures established for securing the reliability of the services and the protection of the data communicated within the network [15]. This security consists of several different items, where firewalls, traffic filtering, logging and encryption are some of them. The network is also divided into different zones, based on the level of trust between the communication partners.

Figure 2 shows a simple overview of how the different entities are connected to a HN. UNN is the University Hospital in Tromsø, and NLSH is the county hospital in Bodø. Both UNN and NLSH are health regions with their own LAN designs that can follow internal politics for security and access. The health networks role is primarily to deliver routing of traffic towards other health regions and to deliver the communication service needed towards the "internet".

Figure 2 Health network connection



Through years of development within the field of Telemedicine, the goal of achieving an effective and reliable method for transferring medical data [16] continues. Technological solutions become better and available implementations and infrastructure continues its development[17]. As of today there is no superior solution to the transfer of large quantities of medical data. The backbone of medical communications today is run through the use of e-mail protocols POP/SMTP for transfer of smaller

files mostly containing text or small amounts of binary data e.g. images. A skilled programmer would be able to implement solutions taking advantage of existing technology to achieve the goal of transferring large files from one communication part to another. However, one has to take into consideration what costs and what amount of resources the development, and finally the use of such a solution, would demand. Engineers and specialists that work as operators in health care informatics today, has to address many problems that isn't directly linked to design, implementation and management of new or existing technology. One of the many issues addressed by it technicians is whether the data sent or stored is in violation with any of the laws and regulations that exist for transfer of medical data.

When a new service is introduced into the operating environment of a health region, the first phase of acceptability is performed by the owner of the environment, the health corporation. This acceptance need to be discussed with strategic personnel at the operators (engineers/IT Specialists) where the technological aspects is seen in proportion to the need for succession of the laws and regulations stated for the data to be handled. These sessions would also have to incorporate economic issues related to possible implementation and eventually operational resources needed. It is in the latter phase where IT personnel use a lot of time and resources on deciding what level of discretion the data that is handled, should be under, instead of providing for the best possible operational level.

During the implementation phase there are several important issues that need to be addressed, and one of them is the level of security enabled on the traffic flow within the new system. What gateways will be passed through, what firewalls have to allow traffic and what systems will be exposed to the new resident. When embarking on this analysis one could grade the intrusion into existing security systems in e three fold, for classifying port access through firewalls.

- 0 – No openings, would be the very best and of course keep the systems optimal safe
- 1 – One opening, giving new implementations access but at the same time keeping security to a satisfying level. Would of course need risk and security analysis.
- Many – More than one opened port for access would demand comprehensive risk and security analysis in addition to a much higher demand for resources in the operating department.

An application that would support several services running through the same firewall accessed port could reduce the cost of resources needed by the operational department. It could also decrease the need for preparatory work for the new service to be accepted as a part of the risk and security level that is maintained.

As proposed by Sachpazidis [18] it is possible to use a port dedicated to a specific service for other purposes, and XMPP delivers a possibility by its use of XML streams that can be encapsulated into the HTTP protocol which most times is routed by firewalls. However seeing that XMPP could uphold all services by opening for traffic to and from clients on port 5222 or 5223 to enable both messaging and file transfer, could eliminate the need for any further implementations. As defined in the problem definition, file transfer is the topic of this thesis, and file transfer is made possible with its use of the in-band extension XEP-0047 [7].

2.2 XMPP basics and fundamentals

XMPP is based on the Extensible Markup Language (XML)[19] and basically intended for instant messaging and online presence detection. The protocol functions between and among servers and facilitates almost real-time operation on the communication setup. According to the XMPP – Core [20], the server acts as an intelligent abstraction layer for XMPP communications, and its primary responsibilities are:

- To manage connections from or sessions for other entities, in the form of XML streams to and from authorized clients, servers and other entities
- To route appropriately-addressed XML stanzas among such entities over XML streams.

The core further states that most XMPP-compliant servers also assume responsibility for the storage of data that is used by clients.

With the use of XMPP clients (IM clients) one can connect to a XMPP server and initiate communication with other clients connected to the same server or network. Creating a client account on a Jabber server is very easy, and many servers have the possibility of creating new accounts directly during the first connection attempt. When connected to the server, the user (client) can add other users (clients) to their roster by adding the identifier, Jabber ID (JID) of another user. The server keeps the roster updated with real time information of each contacts presence status. This is done by a continuous dialog simply explained with a message asking the given JID if he is there, whereupon the client answering to that JID answers with its presence status. The JID resembles a combination of an email address and a hosts URL. It consists of node, domain, and resource. Only the domain part of the JID is mandatory. Example of a JID existing on the test environment setup for this thesis: *garm@yme/ph1*. By connecting another client to the *yme* domain and adding user *garm@yme*, would establish these two users in both unique contact lists. During the setup of a new client, one also has to know the server hostname/IP-address and a username/password for authentication. The

connection between the XMPP clients and the XMPP servers is performed on top of a TCP [21] connection.

From the XMPP core document [20] by Peter St.Andre one can read that the communication is based on the concept of XML streams and XML Stanzas.

> *"An XML stream is a container for the exchange of XML elements between any two entities of a network. To start a XML stream one denotes a XML <stream> tag, and to end the stream, a </stream> tag is denoted. As long as the stream is active, the entities can send any number of XML elements over the stream. The tags have to be within the boundaries of a preset namespace to be understood and processed correctly. The stream is like a normal XML document that has the size of the whole communication session". The initial stream enables unidirectional communication from the initiating entity to the receiving entity; in order to enable information exchange from the receiving entity to the initiating entity, the receiving entity MUST negotiate a stream in the opposite direction (the "response stream") [20].*

An XML stanza is a discrete semantic unit of structured information that is sent from one entity to another over an XML stream[20]. The XML stanza can be explained as the XML data elements which build up the XML data stream. There exists 3 core stanzas, *<message/>, <presence/>* and *<iq/>.* The <message/> stanza is used to identify the message element. The <presence/> stanza is used to inform all subscribers the status of the client; "I am here. Update subscribers of my status". Finally the <iq/> stanza describes a info/query – request/response identifier. This will be explained further in the section describing the XML data stream when transferring files.

In addition these stanza's (elements) all have some common attributes defined, that are explained below.

- *to:* JID of recipient
- *from:* JID of sender
- *id:* Can be a unique ID assigned to each stanza
- *xml:*lang: used to specify the human language in the message

The process of connecting to the server, authentication and exchange of certificate data will not be discussed in any detail. For a more thorough view on these subjects, the is referred reader to RFC3920: XMPP Core [20] and RFC3921: Extensible Messaging and Presence Protocol (XMPP) Instant Messaging and Presence [9].

When the clients have connected and authenticated to the server(s), and their presence has been made available to them both, the two XML data streams needed for two-way communication of messages between the clients can be established. The next section will explain in further detail how the XML stream is built up, and what it actually contains of data.

2.3 File transfer

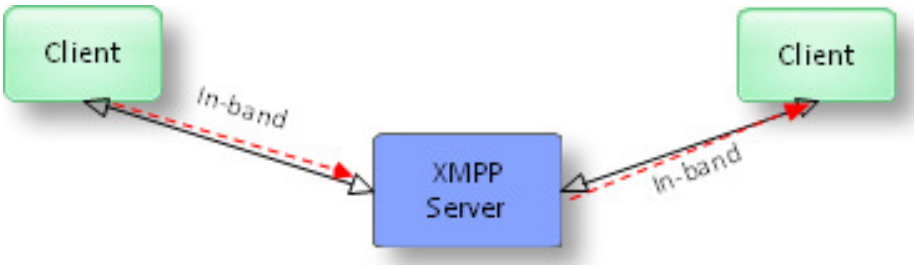
Throughout development of the protocol, the need for further mechanisms soon made its appearance. The possibility of transferring files soon became evident. This resulted in the development of extensions delivering the file transfer possibility.

Several extensions (XEP) have been developed to support the application of file transfer. XEP – 0096 File transfer[22] can be seen as a bundle of XEP – 0065 and XEP – 0047, and describes some of the first attempts of file transfer to be unsuccessful attempts. The *Out of Band data (XEP – 0066)* solution did not work as intended, and has mostly been discarded by the XMPP/Jabber community. There are several drawbacks to using that extension related to reliability and that this solution doesn't work when one of the parties is behind a firewall.
XEP - 0096 File Transfer [22] gives an overview of how to implement seamless file transfer. For this to be possible the implementation need to support both XEP-0065 SOCKS5 Bytestreams and XEP -0047 In-band bytestream. The two choices for file transfer is also to be preferred in that order, making the In-band mechanism operate as a fallback option where SOCKS5 Bytestreams cannot be used. The two extensions, 0047 and 0065, can also be implemented and used as separate methods for file transfer. Even though they *strive* to achieve the same goal, to transfer files between XMPP entities, their design differences are evident.

2.3.1 XEP – 0047 In-band byte streams

The XEP-0047 extension is according to its specification [7] designed to enable a one-to-one bytestream between two entities, where data is broken down into smaller parts and transported in-band over XMPP. The definition “In-band” means that the data is transported within the XMPP stream, as opposed to XEP-0066 Out of Band data [11], where the data is transported outside the XMPP stream.

Figure 3 In-band connection



One of the disadvantages of this method is that the traffic flow inline over the Jabber server where the two clients are connected as sender and receiver. The use of this method creates extra load on the server, which might cause problems for servers that handle a large amount of requests. This should mostly be a problem for the public servers also used for instant messaging, and shouldn’t become an issue in the setting in which these tests are to be performed. Also the fact that this method is “inline”, the XEP[7] mentions that it could experience transfer speeds slower than through the direct connection (Out of Band). The in-band extension is by default intended for transferring small payloads, like text files or small binary images. Data that is to be transferred is base64 encoded and then broken down into smaller parts, with a recommended block size of 4096.

It is then sent with the use of the <iq> or <message> stanzas. Each package sent is enumerated with the <seq> attribute. This is an incremented value assigned to each package starting with 0 and running up to 65535. When the counter has gone through the entire number range it starts at 0 again. The two types of stanzas have some small differences in how they handle send and receive, but this will not be discussed here. If one of the packets fails to be delivered, the sender will consider the bytestream to be closed and invalid. The packets will be routed to the client by the server, and the client needs to acknowledge reception of this package before the next one is sent. This will create an extra load on the server and in fact can be called traffic overhead. The sender doesn’t need to await the acknowledgements before sending the next packet, but this is recommended. When the file transfer is over and the sender wants to close the bytestream, a message is sent giving the

receiver notice of its closure. Receiver accepts it and responds with a message indicating that the bytestream is closed. During reception of data it is important that the data is processed in the order it is received. Packets that are out-of-sequence for a specific bytestream are considered lost. This will also result in the recipient considering the bytestream as invalid and closed.

Among the advantages of IBB as opposed to OOB, one has a layer of anonymity since neither of the client's needs to know the other clients IP-addresses. Only the server can see and knows the IP-addresses of the clients since the transfer is initiated within the existing XML stream. This method also supports what can be called "automatic" file transfer, since the stream is already established; the receiver doesn't have to perform any client configuration when a binary-data stream is initiated inside an existing stream between the two peers.

Structure of XML bytestream

The descriptions of the XML interaction below is a depiction from XEP – 0047 In-band bytestream [7], where the JID's and resource identifications has been altered to the test setup used in this thesis.

Figure 4 Initiation of interaction

```
<iq type='set'
from='radiologyA@domain.xxx/hospitalA'
to='radiologyB@domain.xxx/hospitalB'
id='inband_1'>
<open sid='mysid'
block-size='4096'
xmlns='http://jabber.org/protocol/ibb'/>
</iq>
```

The figure above shows the setup of the initiation of a in-band bytestream. This asks **radiologyB** if it would like to form an In-Band Bytestream connection using the session ID 'mySID'. The session ID is generated by the initiator. The block-size attribute, specifies the amount of data (in bytes) than an IBB packet may contain.

Figure 5 Success response

```
<iq type='result'
from='radiologyB@domain.xxx/hospitalB'
to='radiologyA@domain.xxx/hospitalA'
id='inband_1'/>
```

The figure above shows a success response to the initiation in Figure 2. RadiologyB states that the bytestream is active.

Sending data

Data is sent using either the <message> or <iq> stanzas, and the figure below shows data sent using the <message> stanza.

Figure 6 Sending data using <message> stanza

```
<message from='radiologyA@domain.xxx/hospitalA'
to='radiologyB@domain.xxx/hospitalB' id='msg1'>
<data xmlns='http://jabber.org/protocol/ibb' sid='mySID' seq='0'>
  qANQR1DBwU4DX7jmYZnncmUQB/9KuKBddzQH+tZ1ZywKKOyHKnq57kWq+RFtQdCJ
  WpdWpROuQsuJe7+vh3NWn59/gTc5MD1X8dS9pOovStmNcyLhxVgmqS8ZKhsb1Veu
  IpQOJgavABqibJolc3BKrVtVV1igKiX/N7Pi8RtY1K18toaMDhdEfhBRzO/XBO+P
  AQhY1RjNacGcs1khXqNjK5Va4tuOAPy2n1Q8UUrHbUdOg+xJ9BmOGOLZXyvCWyKH
  kuNEHFQiLuCY6IvOmyq6iX6tjuHehZ1FSh8Ob5BVV9tNLwNR5Eqz1klxMhoghJOA
  </data>
<amp xmlns='http://jabber.org/protocol/amp'>
<rule condition='deliver' value='stored' action='error'/>
<rule condition='match-resource' value='exact' action='error'/>
</amp>
</message>
```

As shown in the figure above, the <data> tag contains the **mySID** attribute generated during the interaction of the bytestream. It also contains the first of the **seq** value, set to its initial value of 0. These attributes are followed by the actual data block of maximum 4096 bytes of base64 encoded data.

Base64 encoding

As mentioned above, the IBB implementation is based on base64 encoding [23] the data that is to be transferred. Base64 encoding was introduced as a solution to the problem with binary attachments in e-mail communications. The main problem, according to Morin [24] is that a binary attachment doesn't read well when encapsulated into other documents. A binary files format is basically a length, followed by a series of bytes. Some of these bytes might be zeroes that in some documents represents end of file. The bytes of the binary file might also consist of other series of bytes that have some special representations that could twist the encapsulation of the attachment. As a solution it was designed a representation protocol that would allow binary data to be encapsulated into another document. The standard is called base64. The actual encoding is done by mapping the data values represented by eight bits, into a subset of ASCII code. There is a base64 alphabet made up of 64 characters and the equal '=' sign, and this is shown in Figure 7.

Figure 7 The base64 alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Problems with base64 encoding

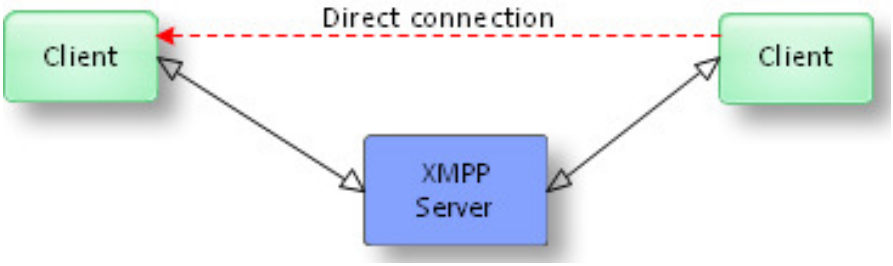
The process of encoding the binary data with base64 creates an overhead of about 30 %. Overhead means that the data will grow in size when encoded. This isn't a mentionable problem when transferring small files, as the 30 % overhead would be represented in kilobytes and will have no effect on the file transfer. When transferring large files (50 – 200mb) the overhead will create large amounts of extra data that will have an impact on the file transfer itself and also on the encoding and decoding processes.

2.3.2 XEP – 0065 Socks5 byte streams

XMPP was initially designed for streaming small fragments of XML between network entities [8], but it soon became evident that it would be valuable to have a generic protocol for streaming binary data between any two entities on the network. The main application for such an byte streaming technology would be file transfer, as mentioned in XEP – 0065 Socks5 Byte Streams [8].

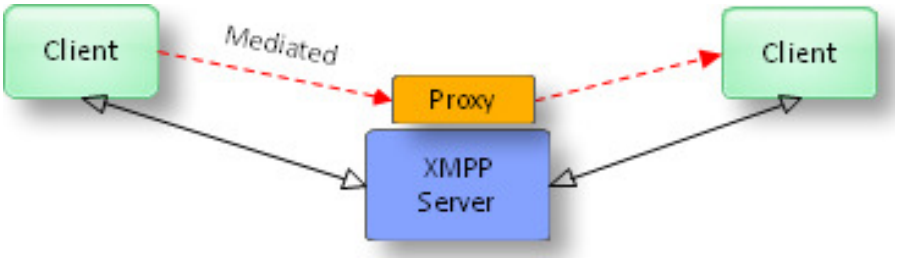
The byte streams are established over standard TCP connections or UDP associations where TCP support is required and UDP is optional. The sockets can be initiated as a P2P (peer-to-peer) option, as shown in Figure 8, or in a mediated connection as shown in Figure 9.

Figure 8 Direct connection



A mediated socket connection is explained by the use of a byte streaming service (proxy) for establishing the socket connection. The XEP describing the protocol also proposes that the Jabber community developing applications supporting the option of file transfer, makes use of the SOCKS 5 protocol [25].

Figure 9 Mediated (proxy) connection



The entities described and mentioned as participants in this solution are:

Initiator: The Jabber entity that wishes to establish a bytestream with another entity.

Target: The entity that the initiator wants to establish a bytestream with.

Proxy: A jabber entity that is not in a NAT/Firewalled environment and is willing to be a middleman for the bytestream between the initiator and the target.

Stream Host: The system that the target connects to and that is "hosting" the byte stream, may be either the initiator or a proxy.

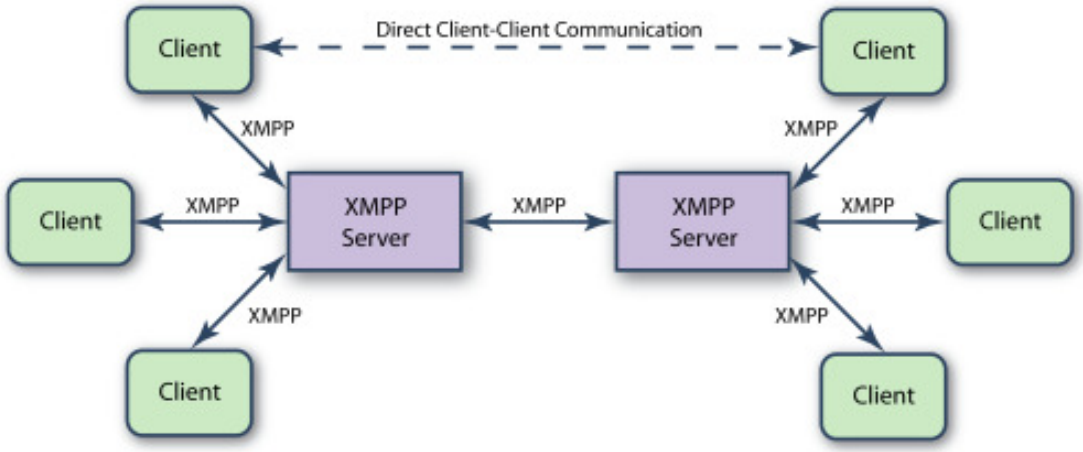
StreamId: A relatively unique Stream ID for the established connection. It is generated by the initiator.

This extension has the option of two different connection methods for establishing the bytestream. The simplest one and easiest to implement, is the direct connection. This situation demands that the initiator of the file transfer knows the network address of the stream host and when to activate the bytestream.

The document entitled XEP – 0065 SOCKS5 Byte Streams [8], describes the process of establishing this byte stream in detail:

1. *Initiator sends IQ-set to Target specifying the full JID and network address of StreamHost/Initiator as well as the StreamID (SID) of the proposed bytestream.*
2. *Target opens a TCP socket to the specified network address.*
3. *Target requests connection via SOCKS5, with the DST.ADDR and DST.PORT parameters set to the values defined below.*
4. *StreamHost/Initiator sends acknowledgement of successful connection to Target via SOCKS5.*
5. *Target sends IQ-result to Initiator, preserving the 'id' of the initial IQ-set.*
6. *StreamHost/Initiator activates the bytestream.*
7. *Initiator and Target may begin using the bytestream.*

Figure 10 Shows client-client communication (http://www.isode.com/whitepapers/xmpp.html)



The alternative to the peer-to-peer based stream initiation is the use of a proxy server where the sender and receiver connect from behind their firewalls. In this situation the stream host is not the initiator but the proxy, which means that the initiator must discover the network address of the stream host before sending the initial IQ-set. The initiator must also negotiate a connection with the stream host in the same way the target does, and it must request that the Stream Host activate the bytestream before it can be used. The entire process of establishing a bytestream is also described in detail in the XEP – 0065 SOCKS5 Byte Streams [8] extension document:

1. *Optionally, Initiator discovers the network address of StreamHost in-band.*
2. *Initiator sends IQ-set to Target specifying the full JID and network address of StreamHost as well as the StreamID (SID) of the proposed bytestream.*
3. *Target opens a TCP socket to the selected StreamHost.*
4. *Target establishes connection via SOCKS5, with the DST.ADDR and DST.PORT parameters set to the values defined below.*
5. *StreamHost sends acknowledgement of successful connection to Target via SOCKS5.*
6. *Target sends IQ-result to Initiator, preserving the 'id' of the initial IQ-set.*
7. *Initiator opens a TCP socket at the StreamHost.*
8. *Initiator establishes connection via SOCKS5, with the DST.ADDR and DST.PORT parameters set to the values defined below.*
9. *StreamHost sends acknowledgement of successful connection to Initiator via SOCKS5.*
10. *Initiator sends IQ-set to StreamHost requesting that StreamHost activate the bytestream associated with the StreamID.*
11. *StreamHost activates the bytestream. (Data is now relayed between the two SOCKS5 connections by the proxy.)*
12. *StreamHost sends IQ-result to Initiator acknowledging that the bytestream has been activated (or specifying an error).*
13. *Initiator and Target may begin using the bytestream.*

One drawback to this solution is that it will not work in a firewall/NAT based network environment [26], unless the proxy is placed outside the firewalls i.e. reachable from both clients. The use of the

proxy server adds a layer of anonymity in this solution since the two involved clients doesn't need to know the other clients IP-address. All requests done by either of the clients is relayed through the proxy opening a possibility for clients outside the firewall to communicate with clients behind the firewall. By using the socks protocol , one makes it possible for applications to transparently use the services of a network firewall [27].

Structure of XML stream

The descriptions of the XML interaction below is an exact depiction from XEP – 0065 SOCKS5 Bytestream [8].

Figure 11 Service discovery to proxy

```
<iq type='get'
  from='initiator@example.com/foo'
  to='streamhostproxy.example.net'
  id='proxy_info'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Figure 11 show the initiator performing an service discovery to check if it is a bytestreams proxy. The proxy will return its information.

Figure 12 Server reply to discovery request

```
<iq type='result'
  from='streamhostproxy.example.net'
  to='initiator@example.com/foo'
  id='proxy_info'>
    <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <identity category='proxy'
    type='bytestreams'
    name='SOCKS5 Bytestreams Service'/>
    ...
    <feature var='http://jabber.org/protocol/bytestreams'/>
    ...
    </query>
</iq>
```

The XEP further describes how the Initiator must request the full network address used for byte streaming if the StreamHost is a proxy. If the StreamHost is the Initiator this is not necessary.

To be able to establish a bytestream after initiation and service discoveries, it is necessary for the Initiator to provide network address information for the StreamHost to the Target. This happens in-band, and the protocol format is shown in Figure 3.

Figure 13 Initiation of Interaction - Network addresses

```
<iq type='set'
  from='initiator@example.com/foo'
  to='target@example.org/bar'
  id='initiate'>
    <query xmlns='http://jabber.org/protocol/bytestreams'
    sid='mySID'
    mode='tcp'>
    <streamhost
    jid='initiator@example.com/foo'
    host='192.168.4.1'
    port='5086'/>
    <streamhost
    jid='streamhostproxy.example.net'
    host='24.24.24.1'
    zeroconf='_jabber.bytestreams'/>
    </query>
</iq>
```

This is just a small part of the complete XML message exchange performed before and during the establishment of an SOCKS bytestream for file transfer. For the complete description and details, please confront the XEP-0065 SOCKS5 Bytestream extension.

2.4 Present solution – POP/SMTP

At the present time there are no dedicated solutions for transferring large files between the communication parties of the health network. The solution used for today's exchange of clinical data is the regular e-mail protocols POP and SMTP [28] . These protocols have been around for several years and are familiar for both their advantages and limitations. As with the XMPP based solution that will be tested in this project, the common e-mail configuration is based on a client- server setup. The clients have unique accounts on the server based upon an identifier attaching them to the server containing the address that matches the identifier. The identifier is commonly known as the e-mail address on the format <username@domain.com>. This is very similar to the configuration setup in XMPP. The e-mail client is polling the server at a given interval to check for new messages. This is a

big difference as opposed to the XMPP solution where the session between the client and the server is kept alive until the client ends it.

There are several suppliers of communication solutions for the healthcare service. Well Diagnostics have developed Well Communicator which is a widespread product for providing communication of clinical information between the different parties in Norway. Well Communicator is an application that can be described as an advanced e-mail client. The application performs several tasks on each message before it is sent. Among these tasks are encryption/decryption, addressing, local routing and validation of message formats and centralized standards. Messages are transported through the regular e-mail protocols POP and SMTP [27]. Each communication part has an account on a mail server with a corresponding email address. Messages are routed as email, with the Communicator working in both the sender and the receiver end of the communication. In the receiver end, the Communicator processes the messages according to local rules decided by the operating personnel.

For transferring small amounts of data (text) the e-mail solution delivers a satisfying service (ref users). When files needed to be transferred increase in size problems suddenly arise. Many mail servers have limitations on how big the message or the attachment can be. Consider thousands of e-mails being transferred where each e-mail was 5-50mb of size; this probably would, and most likely will end in a malfunction on the server side.

The same problems with overhead related to base64 encoding will be present with the use of SMTP as transport protocol for large files.

2.5 Possible solution - FTP

When discussing the subject of file transfer, one should of course include the protocol designed for that purpose, the file transfer protocol [4]. In a one to one communication setup the ftp protocol would probably be the best solution for such file transfers. Files can be transferred almost regardless of their size, meaning that the only thing limiting the size of the files is the space available on the receiver's storage area. The fundamental of the FTP transfer solution is based on a client / server setup. For each repository of files, there needs to be set up some sort of FTP server software that handles user (client) authentication and directs the user (client) to the correct path for its files. A setup based on FTP would meet scalability issues when the amount of communication parties reaches a given amount. Let's picture a scenario where all hospitals in Norway should be able to transfer their medical images to both the primary health care sector and to each of the other hospitals. This would indicate that each hospital has to host their own FTP server, alongside a client

solution that collects the files that is intended for them as the receiving part. The best solution would be to hosts a centralized server solution, where each communication party has their own client accounts for retrieval and submission of their files.
Due to laws and regulations given by the Norwegian Data inspectorate, a centralized solution is not possible and would have to be discarded until such a possibility is given in legislation. There also are some major differences in how the FTP setup would operate as opposed to XMPP. The FTP clients have to do an actual check on the server to see if there are any new files available. This will of course be a routine that needs to be integrated into the runtime environment, but it will most likely result in a lot of unnecessary polling of the FTP servers creating both network and hardware loads. This issue is something that might be utilized better with XMPP's feature of presence. The client will not perform any action on the connection until the server informs that there is a file available on another client. There are some drawbacks to XMPP when it comes to the presence feature. This is discussed in Chapter 6 in the scalability section.

The use of FTP also introduces problems related to the traverse of firewalls and security measures. By its default nature, the FTP protocol uses port 21 for its control stream of commands, and it uses port 20 or other depending on the transfer mode. This result in challenges to network components that might need enhanced logic and configuration setups to handle such issues. The FTP protocol is also stated as a high latency protocol based on the number of commands needed between client and server before it initiates the actual transfer. The FTP protocol also by default transmits usernames and passwords in clear text over the network which should be considered a security risk. A solution to this problem is to use the SFTP protocol (SSH FTP) or FTP over SSL that enables SSL/TLS encryption to the FTP protocol. This is specified in RFC 4217 [29]. Another feature that creates trouble in a network environment where traffic is limited with firewalls and secure zones is the use of its "passive mode". Passive mode is enabled with the FTP server opening a dynamic port which is sent to the client combined with the servers IP address. The use of this method creates problems for NAT devices that have to alter these values so that the IP address the one of the NAT-ed devices.

A setup based on FTP could be a cost demanding option in several ways. There will be costs related to hardware (servers), software (client and server software); operating personnel that will make sure everything run as intended, and related to administration of the solution throughout the organization.

2.6 Future solutions with XMPP

The community surrounding XMPP and the Jabber services push the technology forward with both new extensions but also improvement of the existing ones. During the work on this thesis an impending idea of designing a telemedicine store and forward solution based on XMPP took form. XMPP delivers some new aspects that should be investigated further. Some of these topics have been addressed in Chapter 7 – Future Work.

2.7 Summary

This chapter has given an overview of the XMPP protocol and the two file transfer extensions used for the experiments performed in this thesis. The present solution for file transfer has been presented, alongside with the possible setup of a FTP based file transfer solution.

3.0 Requirements and specifications

This chapter will present external and internal requirement for the test scenario and the operating environment were the tests have taken place. External requirements are specifications of laws and regulations given by the health inspectorate and data inspectorate. Internal requirements are the specifications that the tests are performed within and against.

3.1 External requirements

As mentioned in chapter 2.1.3 there are several demands and requirements that a service handling medical data needs to comply with. On September 7 2006 the department of national health issued a press release stating that a standard for information security [30] in the health sector had been composed. According to the press release the standard states what is needed to achieve a satisfactory level of information security when dealing with medical and patient sensitive data. The standards draft consists of a total of 38 individual data sheets describing the different categories of health information use. One important requirement stated in the standards documentation is the need for encryption of medical data. All messages that contain health and patient sensitive information are to be encrypted in a level equivalent to DES128. There also exists a definition of encapsulated security by using ebXML. ebXML (Electronic Business eXtensible Markup Language) is a standard framework for communication of electronic messages, and it delivers an extra security measure. ebXML is defined as an envelope keeping track of sender, recipient and the data itself is packed into the ebXML envelope. The ebXML standard is extensive and complex and will not be discussed any further in this thesis. It is only mentioned as one of many measures taken to provide for a safe and secure communication link.

When it comes to storage of health and patient sensitive information, the standards track defines several measures that have to be performed before storage can take place. There are several levels of security depending on what storage unit, level of accessibility, storage time, and storage time the data in question is placed under. As the overall intention of the standards track [30] is to make sure the general information security is kept at a acceptable level i.e. existing routines and knowledge level, all demands and requirements aimed at specific technological aspects has to be considered by IT professionals.

The protection of computer networks and what access level is necessary and acceptable is a dynamic and continuous process in the operating environment in question. Some minimum levels of security are always in place and always will be. As a brief overview of security measures in the operating environment used for these experiments, one can mention:

- Encryption
- Logging
- Firewalls / IP filters / Access filters
- Security zones

One has to classify what systems and applications should be covered by what measures, but this is beyond the scope of this thesis.

3.2 System requirements

Each part of the test environment needs to be described before starting the actual tests. Some of these requirements are based on information gathered from operating personnel at Helse-Nord IKT [31] [32].

Files

A single instance or series of CT or MRI images will be in range anywhere from 1 – 200 megabytes of size. The tests will be performed on files of different sizes within this range. Even though medical images in the future might become much bigger, the tests performed in this thesis will be done on the sizes specified below.

Table 1 Chosen file sizes for tests

Filename	Size
tfile1.dat	1024 KB
tfile2.dat	10240 KB
tfile3.dat	51200 KB
tfile4.dat	102 400 KB
tfile5.dat	204 800 KB

All files have been produced using The Dummy File Generator [33] that can be downloaded for free. All files that is transferred will be checked with the FCIV File checksum verifier [34] before and after the transfer. These values are compared after the file transfer to verify that the files haven't changed.

The output from the FCIV utility is an md5 hashed value, and for tfile4.dat this value is "**d0ba30a340655e97a70a214846a136ae**".

When transferring files using the in-band solution the actual size of the file transferred will be bigger than the initial size. This is due to the base64 encoding creating up to 30% overhead. When presenting the performance of the in-band measurements, the speed of the file transfer will be calculated from the initial file size. The reason for this is the basis for comparing the in-band method with the mediated and direct methods.

Clients and server

The chosen solution has to support the extensions needed for this thesis, and it has to be documented to a degree that makes it possible for uninitiated persons to quickly get a grasp of what the software delivers. The choice of software also should be based on a level of stability and reliability. Experimental software might deliver some benefits, but the intension of this work is to find out if the two extensions for file transfer will operate reliable.

Security

A production ready file transfer solution would have to consider encryption /decryption of the data transferred. All medical information communicated electronically today is encrypted before it is sent. In the test scenario this issue has not been taken into consideration. The built in solution for this kind of security in XMPP might not be good enough for approval from the Norwegian data inspectorate. The data might be encrypted by other mechanisms before it is transferred to the receiver where it is decrypted. Traffic encryption is also something that would have to be considered by a production solution. XMPP delivers security measures through both Simple Authentication and Security Layer (SASL) [35] and Transport Layer Security (TLS) [36]that is built into the core specifications. There might also be different views on what level of security is necessary within the different communication parties. Issues regarding security on file level are beyond the scope of this thesis, where the main goal is to find out if the technology itself will perform as an option for file transfer.

Base64 encoding binary data

As mentioned in the previous chapter all binary data transferred in-band has to be base64 encoded prior to transfer. Implementation differences in the various clients might influence the overall performance of the file transfer session. Therefore it is necessary to find out what level of performance the base64 encoding will have on large files. Each file transferred will be base64 encoded separately and timed, to single out time consumption on the encoding itself. The results from the standalone base64 en coding will be presented in the test results presentation in chapter 5.

If it is shown that the encoding is a resource demanding and time consuming operation, it would be useful information for developers of a possible production ready solution for file transfer with XMPP. One could then consider developing a parallel solution for encoding of the binary data.

Network

One of the requirements stated for this thesis, is that the experiments should be performed in a health network. Server and client computers have to be placed within an environment as close as possible to the one described in the problem definition. The main reason for this is that one has to see what limitations the technology has when it comes to network related aspects. One of the subjects here is the use of NAT in the network design. This will also show what administrative tasks has to be performed by the system operators at the locations in question for a possible production solution.

3.3 Measurement specifications

To be able to make a statement regarding the reliability of the XMPP protocol as a file transfer mechanism, a set of measurements have been defined.

Mean time between failures

To get an overview of what level of reliability the protocol delivers it is necessary to measure amount of failed transfers. This will be done by counting the number of failed file transfers during a given time period.

T: amount of time with active transfers

N: number of attempted file transfers.

S: number of successful transfers

F: number of failed transfers during T

Based on these values, one can present a basis for the protocol's reliability shown by a percentage of success/failure.

Scalability (SC)

The solution needs to be scalable. This means that one will have to measure the effect on MTBF when several clients' transfers files at the same time. Each instance of a singular file transfer will be measured according to MTBF and cross checked towards a diagram holding the amount of transfer instances. The degree of scalability will also be dependent on what software solution is chosen, and the level of throughput possible in the operating environment. The effect on the hardware (server)

will also have to be monitored when the solutions scalability is measured. Server loads has to be monitored during the file transfers to measure the effect it has on CPU and memory use.

Resilience to errors

How does the size of files, number of active file transfers, and software solution, affect the overall resilience to errors? It is necessary to get some kind of view into the amount of errors and if this can be linked to any of the previous mentioned factors.

Efficiency

The file transfer must be performed within a reasonable amount of time. The definition of reasonable will be covered in the discussion, but anywhere between 10 – 40 seconds /MB would be used as a maximum of time consumption. This will of course depend on the throughput capacity on the network setup. Efficiency will only be based on time assumption, and have to be measured against the maximum throughput possible with the network media available.

Durability

Durability is understood as a measurement of reliable operation during a period of time with constant activity. The durability tests will be performed by constantly running file transfers randomly between the clients. We will be measuring the amount of failed transfers and be keeping track of CPU and RAM use on the server.
Durability will be tested as follows:

In-band method

- 2 x 12 hours – tfile3.dat
- 2 x 12 hours – tfile4.dat
- 2 x 12 hours – tfile5.dat

Mediated (Proxy) method

- 2 x 3 hours – tfile3.dat
- 2 x 3 hours – tfile4.dat
- 2 x 3 hours – tfile5.dat

Since the software used in these experiments are clients developed for use in IM, the durability tests will have to be performed like the other tests. We will have to manually initiate and accept transfers across the network, and the solution will have to be monitored by us as operators through the entire session. This is one of the reasons why the time periods are split into sessions of 3 and 12 hours. The durability tests should also be performed at different times of day to get measurement of the network variations experienced in a production network.

3.4 General specifications of test framework

The test environment will be as close as possible to the real operating environment of a possible future solution based on XMPP as backbone for file transfer. The tests are performed as a proof of performance to see what capabilities and limitations exist in the protocol and in the technology. To perform the actual file transfers, XMPP clients designed for instant messaging will be used. These clients have been chosen based on their support of the necessary extensions needed for these tests. Since these clients have a user base to attend, they are based upon the latest of extensibility delivered to the XMPP protocol. If any new and decisively better extensions for supporting file transfer had existed, a client supporting this would probably have been released. A lot of clients exist with its main focus on instant messaging and the level of application needed to achieve this. Both the clients and the server software used in the test environment have been chosen based on data sheets describing their supported extensions and the level of usability reported by other users.

One of the most important parts of introducing new technology is the usability of the new solution as opposed to the earlier and familiar solutions. One has to measure the level of adaptability towards the end users, and one should not make assumptions that could end with users discarding the new introductions. In the scenario focused on in this thesis, the end user element will not be a subject. This thesis is focused on the sheer technology of XMPP and if there are any possibilities for using it in covering the problem defined in Chapter 1.2.

3.5 Summary

This chapter has presented the requirements and specifications of the environments these tests will be performed within. It also presents the measurements that the attempted file transfer operations are to be measured against when summing up results.

4 Design and implementation

This chapter will describe the design of the test environment and how the experiment has been preformed. It will describe the different components used and their role. Some parts of the setup make room for discussion, and this will be addressed in chapter 6.

4.1 Design of test environment

During the development of a test environment, it soon became clear that many of the available software solutions, both server and clients, didn't support the needed extensions. This is mainly due to the fact that the clients available is tuned towards a user base and is mainly intended for instant messaging. Not all clients have updated documentation available and this complicates the process of finding the proper client. Two clients have been chosen as participants of the test environment. Spark [37] and Tkabber[38] are two clients found to be supporting XEP-0047 In-band bytestream, with the according documentation available. Of the two, Spark has a huge community through the Jive Software [12] website, which makes information and troubleshooting much easier.

On the server side, there were similar issues when finding the best RTC software for this experiment. Once again most of the available implementations are focused on delivering the best possible experience for instant messaging. There also exist some commercial server variants that deliver a broader set of services than the ones needed for these experiments. During the design of the test environment several RTC implementations was considered. As mentioned in Chapter 2, Tigase, Openfire and eJabberd were all considered as possible choices as RTC implementations. Tigase was dropped due to issues regarding connection problems between the Spark client and Tigase. According to Tigase developer Artur Hefczyc [39] these problems are related to a "*bug*" in Spark, where the client receives responses from the server quicker than it expects. EJabberd where not investigated any further since Openfire soon presented itself as the best choice for this project. Openfire has a very simple and well arranged administrative interface, and it is quick to install on both Windows and Linux based computers. Since both server and client software is needed for the experiments, it is important that the software used would provide stability and reliability to the implemented environment. Based on the preliminary work in software research, the software chosen is the Openfire [40] server installation, and the Spark [37] XMPP client. Both software solutions are developed by the Jive Software organization [12], and are released for free.

Most of the RTC implementations available does support the extensions needed for this thesis, so the choice of software is also based on personal communication with developers and experienced users of the XMPP technology.

4.2 Network – client and server setup

When software for the clients and servers was selected, the test environments had to be designed. Since the main experiment is to be performed within a real health network, the first designs is setup to experiment with the technology. It would also give useful feedback before attempting a setup within the actual health network.
These test networks is designed and implemented with simple network components available in retail. Up to four clients where setup and connected to the XMPP server and added to each other rosters for presence functionality. A set of test files were transferred between the clients to confirm functionality of the different setups.

Table 2 shows the components that have been used in these network scenario tests.

Table 2 Component overview

Component /Name	Specification	OS
Computer A (Garm)	Intel Core2 CPU 4400 @ 2.00ghz 3.50gb Ram	Windows XP
Computer B (Utgard)	Intel Core2 CPU 3300 @ 1.74ghz 1,0gb Ram	WindowsXP
Computer C (Yrgar)	AMD Duron II @ 750mhz 512mb Ram	WindowsXP
Computer D (Magard)	AMD Athlon 2200 @ 1,4ghz 396mb Ram	WindowsXP
Server (Yme)	Intel Core2 CPU 4400 @ 1.74ghz 2.0gb Ram	Ubuntu 7.04
Switch	Linksys WRT54G –v5 - 10/100mbit	
Cables	Category 5 Plus UTP Patch Cable	

The network capacity on a wired connection setup has a theoretical capacity of 100mbit/s, based on the switch and cable components. This includes all computer network interfaces that on computer A and B have gigabit capacity.

4.2.1 In-band client-client communication

The first test network implemented is the one making use of the 0047 In-band Bytestreams extension. The principle of this design is shown in Figure 3, where the red arrows show the file transfer performed within the existing XML stream. As discussed in chapter 2, the in-band communication setup will work without the clients being informed about the other client's network configuration. IP addresses in this setup shows that the two clients is placed in separate LAN's with their own local IP addresses, but the XMPP server is accessible by both clients. Table 3 shows the components and their network configuration for this experiment.

Table 3 Component specification - In-band connection

Name	Software	IP	Component
Garm	Spark	10.0.1.1	Client A
Utgard	Spark	192.168.1.25	Client B
Yme	OpenFire	84.236.250.30	Server

The software used for this experiment (Spark and OpenFire) takes advantage of the functionality delivered by the implementation of the XEP – 0096 File Transfer [22] extension. The 0096 XEP specifies how to implement seamless file transfer, giving the clients an option of falling back on the in-band option if the prioritized solution isn't available. Both clients are placed behind firewalls, removing the option of direct client-client connectivity. Then by disabling the proxy option in the server, the client initiating the file transfer automatically chose the in-band solution by sending an XML element as shown in Figure 4. Upon this initiation the file transfers where started using the in-band protocol.

4.2.2 Direct client-client connection

The next network setup is pictured in Figure 8, and is a direct client-client connection based on the Socks5 bytestream protocol. This communication design is setup within the same network and has no implemented boundaries like NAT or IP filtering. The clients establish a peer-to-peer socket connection as described in XEP-0065 Socks5 Bytestreams [8] extension. Table 4 shows the components used in this experiment and their network configuration.

Table 4 Component specification - direct (peer-to-peer) connection

Name	Software	IP	Component
Garm	Spark	10.0.1.8	Client A
Utgard	Spark	10.0.1.9	Client B
Yrgar	Spark	10.0.1.10	Client C
Magard	Spark	10.0.1.11	Client D
Yme	OpenFire	10.0.1.100	Server

This network was built within an existing home network, where the new clients and the server where added as new domain computers. Figure 10 shows in principle how clients are connected to the server, and show the file transfer as the dotted line between the connected client sockets.

4.2.3 Proxy – mediated client-client connection

The next network setup is a setup to implement the proxy, or mediated, connectivity option delivered by the SOCKS5 bytestream extension. This option is intended to allow for file transfer between clients that are prohibited from peer-to-peer connection due to firewall and security implementations. For this solution to be possible the networks where the clients are components, has to allow TCP traffic towards a predefined port. In Figure 9 a solution based on the proxy being a part of the XMPP server is shown.

Alternatively the proxy can be setup as a standalone implementation and placed in a network zone that both client networks have a trusted relationship with. The component specifications for this network setup are described in Table 5.

Table 5 Component specification - mediated connection

Name	Software	IP	Component
Garm	Spark	10.0.1.1	Client A
Utgard	Spark	192.168.1.25	Client B
Yme	OpenFire	84.236.250.30	Server
Proxy	Yme – OpenFire	84.236.250.30	Proxy / Server

The Openfire installation offers a built-in proxy solution that communicates on port 7777 for the mediated connection between clients. For these experiments this option has been used to prove that the proxy solution will work. The only difference between using the built in proxy solution as supposed to the standalone version, is that the proxy instance uses the same IP address as the XMPP server. By setting up a standalone server as the proxy, one would have to open new access in firewalls and routers for traffic between the client computers and the proxy service.

4.3 Health Network setup

After finishing the pre-face with network and file transfer tests, it was time to place the components into the actual operating environment. Due to ownership issues some of the components had to be replaced with new computers, and this is specified in Table 6. The XMPP server is placed outside the LAN on a connection inside the health network, with clients placed on both sides of natural boundary.

Table 6 Component overview - Production environment

Environment	Component /Name	Specification	OS
Production	Computer A / Radiology A	Intel Core 2 Duo E6550 @2,33GHZ 1.97GB Ram	Win XP SP2
Production	Computer B / Radiology B	Intel Core 2 Duo E6550 @2,33GHZ 1.97GB Ram	Win XP SP2
Production	Computer C / Radiology C	Compaq P4-630 @ 1,0 Ghz 512bm Ram	Win XP SP2
Production	Computer D / Radiology D	AMD Athlon 2200 @ 1,4ghz 396mb Ram	Win XP SP2
Production	Server	Compaq P4-630 @ 2.0Ghz 512mb Ram	Win XP SP2

The server XMPP server is connected to a Cisco 871 –router that is a part of the health networks DMVPN service solution. The core of this solution is 2 x 7206 routers, and the routing table from the XMPP server to the UNN network is shown below.

1. 172.21.137.10 (unn-core-gw0, owned by Norwegian Health network)
2. 10.145.0.133 (unn-gw0, owned by Helse Nord IKT)
3. Next hop is into the UNN / HN-IKT network switches.

To establish a P2P direct connection between the clients using the health network setup, both clients has to be placed in the same LAN. Otherwise one has to enable the computers to access directly through firewalls and to alter NAT tables to recognize the sender and receiver network addresses.

Figure 14 shows the first of two in-band designs used during the experiments. The client named Radiology B was placed on the same switch as the XMPP server enabling it to communicate in-band with the client named Radiology B.

Figure 14 In-band connection - Production

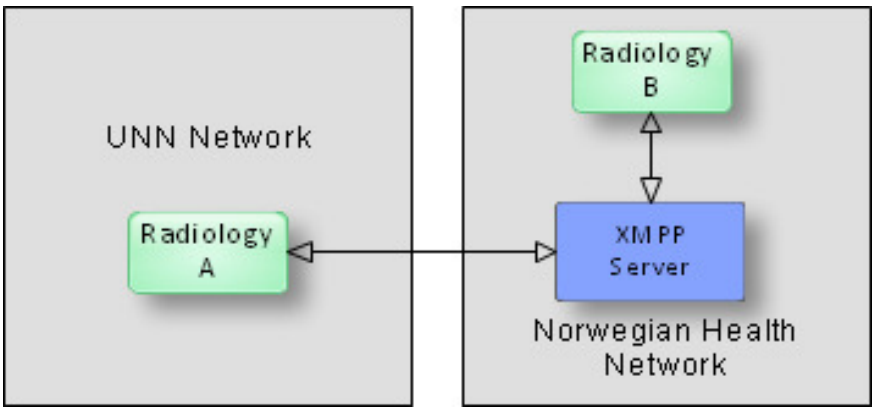
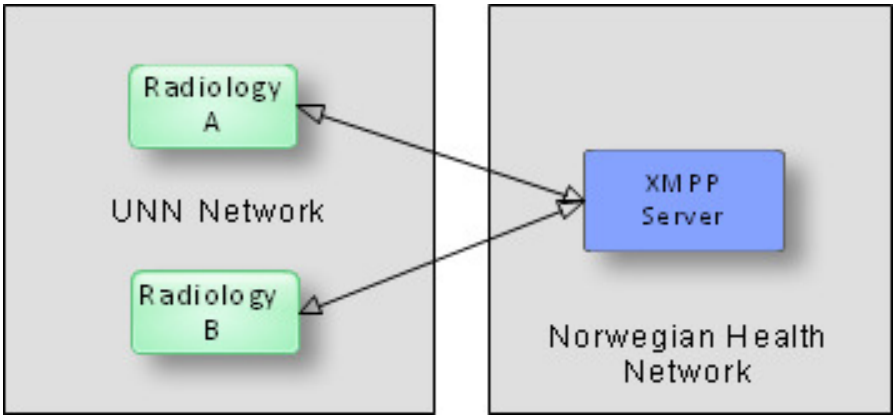


Figure 15 shows the second of the two in-band setups, where both clients are placed inside the UNN network. By disabling all allowed traffic on the firewalls on each client computer, the only traffic allowed was on port 5222 used by the XMPP client. By closing all other connections, the fallback mechanism implemented in Spark [37] as explained in XEP -0096 File Transfer [22] was utilized and all communication was performed in-band.

Figure 15 In-band connection II - Production



The same solution described in Figure 14 and 15 was used for the mediated connection shown in Figure 16 and 17. The only difference this time was that the clients were configured to use port 7777 for proxy file transfer.

Figure 16 Mediated connection - Production

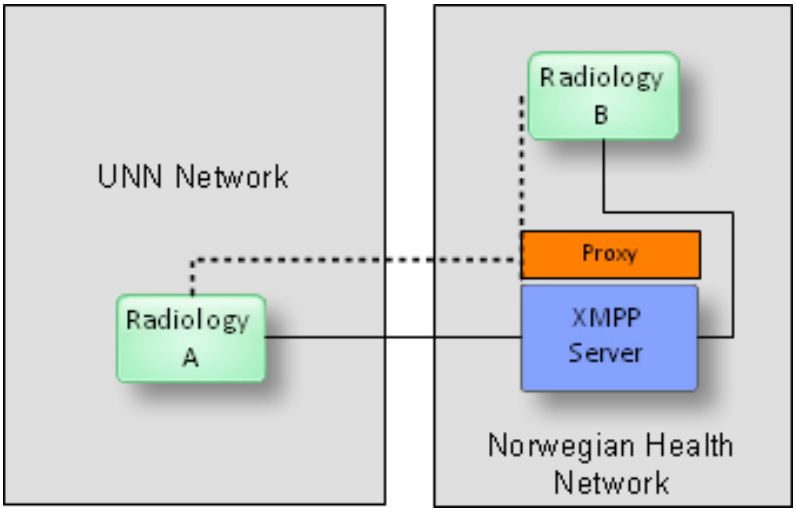
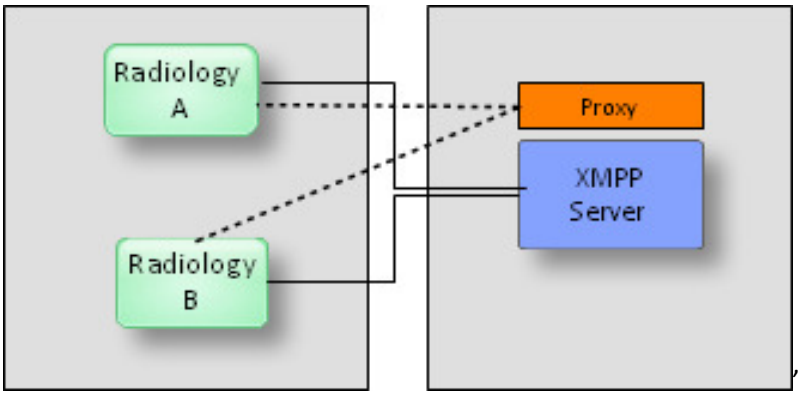


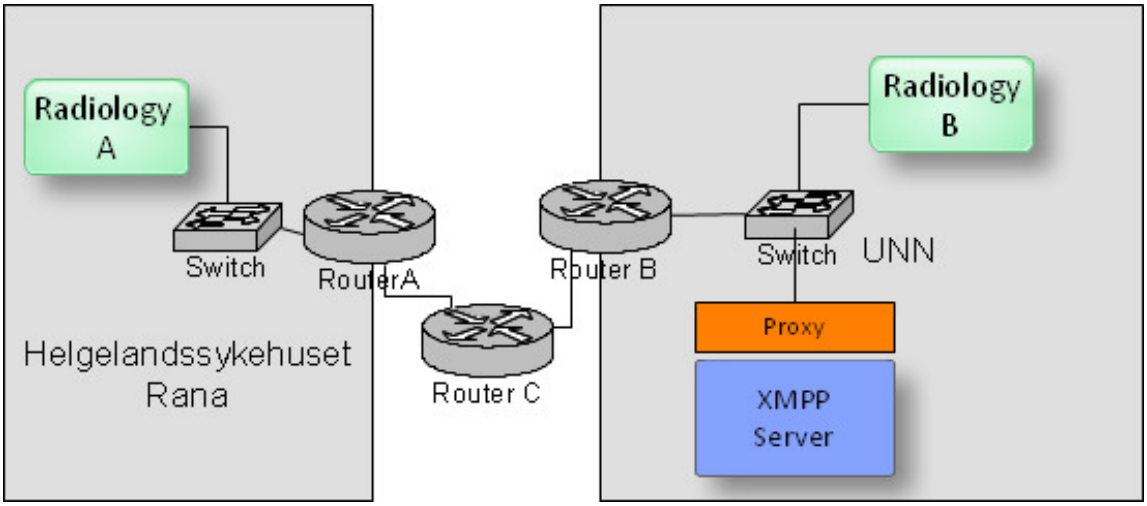
Figure 17 shows both clients on the same in the same network on the same switch. Once again the firewalls had to be enabled but this time opening for file transfer on port 7777. Fallback mechanisms didn't commit and the file transfer was performed using the mediated connection.

Figure 17 Mediated connection II - Production



It is also useful to see whether the performance of the solution is influenced by distributing it into a larger network environment. A communication setup has been set up between the two health regions of UNN and Helgelandssykehuset. These two regions have their own LAN's with unique setups regarded to both design and security. A simplified figure of the communication setup is shown in Figure 18.

Figure 18 Cross region network



As shown in Figure 18 the XMPP server is placed within the UNN network. In figure 2 the block object indicating NHN, is represented as Router C in Figure 18. In this communication setup files are being transferred using the in-band and the mediated proxy solution for transferring files. Ports 5222 and 7777 are opened in firewalls throughout the communication chain. To understand the complexity of the network setup a complete figure showing the communication channel is shown in appendix A, "Cross Region Network setup". Establishing this communication setup between two independent health regions is an administrative task that demands several points of interaction as mentioned in Chapter 2.

4.4 Summary

This chapter has presented the different network designs configured and used for the experiments performed. It has also given an insight into the design issues that has been met during this process, and how they have been solved.

5 Test and measurement results

This chapter will present the data collected from the experiments performed in the previous section. Each method used for file transfer is presented with results of performance matched against the measurements specified in Chapter 3.2. The diagrams, graphs and tables in this chapter will present the performance of each method.

5.1 Pilot and test environment results

After setting up each of the networks, a set of file transfers where performed between two clients to verify that the technology works. The data collected during these transfers is shown in table XX, and should give a sample of what could be expected from the experiments performed in the health network. The same file, tfile2.dat, was used as the object to transfer for all these tests. The files size is 10240 kb.

Table 7 Results - Pilot tests

Setup	Size (Kb)	#Errors	# Transfers	Avergage Time(s)	Average Speed
*FTP	10240	0	15	12	0,8 MBps
Direct (peer-to-peer)	10240	0	15	11,6	0,8 MBps
Mediated (Proxy)	10240	0	15	31,2	0,3 MBps
In-band	10240	0	15	344	0,02 MBps
**SMTP / POP	10240	3	5	577	0,017 MBps

- * The FTP transfer is performed by connecting Computer A directly to the server in a 10/100mbit LAN. The software used for this test is ncftpd for the server and SmartFTP for the client software.
- ** The SMTP/POP test was performed by installing Postfix with accompanying tools on the server and sending a series of messages with tfile2.dat as an attachment. The 3 errors experienced were results of time-out on the outgoing message.

The table indicates that the best performance is achieved by using the direct (peer-to-peer) method for file transfer. In the network used for direct connection, the two clients are connected to the same switch enabling high performance on the LAN.

The performance on the mediated connection shows that the transfer speed is halved as opposed to the direct connection. However in this test, the proxy is placed outside both networks on an ADSL connection with 8mbit capacity. This connection has an 8000/1500 capacity which makes the actual

capacity fluctuate between 0, 5 and 6 mbit, giving an average of 2, 5 mbit for the transfers performed.

The third and last test was performed using the in-band connection for transferring the files. This method experienced an average transfer time of 5 minutes and 44 seconds on the 10MB file, giving an average transfer speed of 0,02 MBps. The XMPP server is placed on the same 8mbit ADSL connection as for the mediated test. During the In-band transfer it was soon discovered that the load on the server was high. The OpenFire XMPP server is built in Java and depends on the Java Virtual Machine (JVM) to run. The default installation of the software dedicates 64 MB of Java Heap space for the server to utilize. When transferring files in-band larger than 64MB, the dedicated memory is quickly occupied by the JVM. If memory usage reaches 98-100 % the file transfer was aborted and all clients disconnected. The memory available for the JVM had t be increased before attempting any further file transfers. Passing some arguments to the JVM by placing a file in the install directory does this. The arguments, *-Xms512m –Xmx1024* tells the JVM to initialize 512MB of Java heap memory and 1024MB of virtual memory for the server to use for handling the load. After increasing the available memory, all attempted file transfers was successfully completed. Memory use was still high but it didn't abort any of the attempted file transfers.

To get a comparable view of the performance of the different setups, the same tests were performed in the LAN used by UNN. This is a high capacity network based on Gigabit Infrastructure.

Table 8 Results - Pilot tests production network

Setup	Size (Kb)	#Errors	# Transfers	Avergage Time (s)	Average Speed
Mediated (proxy)	10240	0	15	1,3	7,6 MBps
*FTP	10240	0	15	1,7	5,8 MBps
Direct (peer-to-peer)	10240	0	15	2.75	3,6 MBps
**SMTP / POP	10240	0	5	225	0,04 MBps
In-band	10240	0	15	302	0,03 MBps

- * The FTP test was performed by using the SmartFTP client and the built in FTP-server service in Windows 2003 Server. A small delay in protocol negotiation for each file transfer
- ** The SMTP/POP test was performed using the existing mail system at Helse Nord IKT where the test systems are deployed. Systems used are Microsoft Exchange with Outlook as the clients.

5.2 Results – XEP – 0065 SOCKS5 Bytestreams

The direct (peer-to-peer) method of file transfer showed during the pre-test to have the best overall performance related to speed and time consumption. An overall of 125 transfers was performed using this method. Table 9 shows the overall results from all the transfers performed using the peer-to-peer connection.

Table 9 Results - Direct (P2P) Connection

File	Size (Kb)	Transfers	Failed	Av Speed	Av time (s)	CPU / Ram load (64mb)
tfile1.dat	1024	25	0	0,6 MBps	1,6	4 % / 12 %
tfile2.dat	10240	25	0	1,05 MBps	9,5	3 % / 9 %
tfile3.dat	51 200	25	0	0,95 MBps	52,3	4 % / 11 %
tfile4.dat	102 400	25	0	0,87MBps	114	3 % / 10 %
tfile5.dat	204 800	25	0	0,89 MBps	223	4 % / 12 %

The mediated method of file transfer had a level of performance during the pre-tests close to the direct connection. Table 10 shows the overall results achieved by using the mediated connection for file transfer in the actual test environment. As seen in Table 10 the JVM load is heavy while transferring files using only 64 MB of JVM.

Table 10 Results - Mediated (proxy) connection

File	Size (Kb)	Transfers	Failed	Av Speed	Av time (s)	CPU / Ram load(64mb)
tfile1.dat	1024	25	0	0,38 MBps	2,6	7 % / 18 %
tfile2.dat	10240	25	0	0,86 MBps	11,5	5 % / 17 %
tfile3.dat	51 200	25	0	0,84 MBps	59,3	6 % / 17 %
tfile4.dat	102 400	25	0	0,78 MBps	127	8 % / 19 %
tfile5.dat	204 800	25	0	0,82 MBps	243	6 % / 18 %

Table 11 shows the results from the Mediated cross region experiment that is described in Chapter 4.3. All of the attempted transfers succeeded but there were some deviations from normal network behavior.

Table 11 Results - Mediated (proxy) connection - Cross Region

File	Size (Kb)	Transfers	Failed	Av Speed	Av time (s)	CPU / Ram load(512mb)
tfile1.dat	1024	10	0	0,71 MBps	1,4	2% / 2.5 %
tfile2.dat	10240	10	0	0,48 MBps	20,5	2 % / 2,5 %
tfile3.dat	51 200	10	0	0,61 MBps	80,7	3 % / 3 %
tfile4.dat	102 400	10	0	0,68 MBps	146,4	3 % / 3 %
tfile5.dat	204 800	10	0	0,73 MBps	270,3	3 % / 3,3 %

The tests performed in the cross region network showed that there where performance differences depending on what direction the files were transferred. When transferring files *from* Radiology A (Rana) the transfer speed was experienced much slower than the transfers started *from* Radiology B (UNN). This was especially noticeably when using the mediated (proxy) method for transfer. For the smallest files, tfile1 and tfile2, the time difference where only a couple of seconds, while time consumption for other three file sizes where up to three times higher. Table 11 shows the results from the cross region network tests, where the average speed is calculated from the total time consumption of file transfer in both directions.

5.3 Results – XEP – 0047 In-band Bytestreams

The file transfers done with the in-band bytestream solution was performed in the network design showed in Figure 14. Table 12 shows the results from the same network design as the tests for the direct and mediated connection. Column three in Table 12 (* transferred) shows the actual amount of data transferred during the file transfer. The actual size of transferred data is higher due to the base64 encoding overhead. The results in the last column in Table 12 are gathered while the server was configured with 64mb of JVM memory.

Table 12 Results - In-band connection 64MB JVM

File	Size (KB)	*Transferred	Transfers	Failed	Av Speed	Av time (s)	CPU / Ram load (64mb)
tfile1.dat	1024	1382	10	0	0,037 MBps	26,8	4 % / 8 %
tfile2.dat	10240	13824	10	0	0,035 MBps	279	4 % / 25 %
tfile3.dat	51 200	69240	10	0	0,035 MBps	1418	8 % / 60 %
tfile4.dat	102 400	138235	10	4	0,026 MBps	3863	11 % / 80 %
tfile5.dat	204 800	278528	10	10	0	0	14 % / 100 %

As shown in Table 12 the JVM load in this experiment increased rapidly as we tried the different files. Four of the transfers of tfile4.dat failed due to the heavy load on the server, whilst all of the tfile5.dat transfers failed for the same reason. This experiment made it evident that 64 MB of available JVM is not enough to support in-band file transfer.

Table 13 shows the results from the in-band file transfer after modifying the JVM settings as described in chapter 4.24. Column three in Table 13 also shows the actual amount of data transferred in this experiment. As we can see, the JVM load has decreased noticeably and all of the initiated file transfers succeded.

Table 13 Results -In-band connection 512 JVM

File	Size (Kb)	*Transferred	Transfers	Failed	Av Speed	Av time (s)	CPU/Ram Load(512mb)
tfile1.dat	1024	1382	10	0	0,040 MBps	24,82	6 % / 2 %
tfile2.dat	10240	13824	10	0	0,038 MBps	257	8 % / 5 %
tfile3.dat	51 200	69240	10	0	0,039 MBps	1289	6 % / 9 %
tfile4.dat	102 400	138235	10	0	0,027 MBps	3652	6 % / 9 %
tfile5.dat	204 800	278528	10	0	0,026 MBps	7480	6 % / 11 %

Table 14 shows the results from the In-band Cross region network setup as described in Chapter 4.3. In this experiment 4 transfer sessions failed from a total of 50 attempted sessions.

Table 14 Results -In-band connection - Cross Region

File	Size (Kb)	*Transferred	Transfers	Failed	Av Speed	Av time (s)	CPU/Ram Load(512mb)
tfile1.dat	1024	1382	10	0	0,037 MBps	26.6	2 % / 2 %
tfile2.dat	10240	13824	10	0	0,032 MBps	312	4 % / 5 %
tfile3.dat	51 200	69240	10	0	0,035 MBps	1409	4 % / 7 %
tfile4.dat	102 400	138235	10	2	0,025 MBps	3930	4 % / 11%
tfile5.dat	204 800	278528	10	2	0,026 MBps	7409	5 % / 14 %

As in the mediated transfers, the performance difference between the two directions of file transfer is noticeable in the in-band tests as well. However with the in-band solution the average transfer speed is much slower than the mediated, so the difference is not worth mentioning since it won't have an impact on the overall performance measurement.

5.4 Results - Durability

To get a picture of what reliability the file transfer mechanisms in XMPP can deliver, a set of transfers where run over a longer period of time. Continuously file transfers where performed during this period to measure failure and other weaknesses. During these periods the number of file transfers where counted and readings of CPU and Ram load where done periodically. Two sets of file transfers were performed during this test, to measure both the in-band and mediated option of file transfer.

Table 15 Results - In-band connection - Durability

File	Time	Transfers	Failed	% Success	CPU / Ram Load(512 mb)
tfile3.dat	2 x 12 hours	50	0	100 %	4 % / 4 %
tfile4.dat	2 x 12 hours	20	2	90 %	6 % / 7 %
tfile5.dat	2 x 12 hours	10	1	90 %	6 % / 9 %

In Table 15 it is shown that the level of reliability is high for the in-band method of file transfer. Only 3 out of a total of 80 transfers failed, and the failed transfers was a result of connection drops between the server and client computer. From a total transfer time of 72 hours, and a total amount of 80

transfers for the in-band connection, the total fail rate was **3, 75 %.** Turning the numbers it gives us a **96, 25%** success rate.

Table 16 Results - Mediated (proxy) connection - Durability

File	Time	Transfers	Failed	% Success	CPU / Ram Load(512 mb)
tfile3.dat	2 x 3 hours	100	0	100 %	2 % / 3 %
tfile4.dat	2 x 3 hours	100	4	96 %	2 % / 3 %
tfile5.dat	2 x 3 hours	50	0	100 %	3 % / 4 %

As shown in Table 16, the durability tests performed with the mediated connection failed 4 times. The test was performed in a time span shorter than the one for the in-band connection due to the amount of files possible to transfer. The 4 failed transfers were due to network related issues, where connection was dropped by the receiving client during the mediated negotiation. From a total of 250 file transfers, the 4 failed amount to a fail rate of **1.6 %.** This gives us a **98, 4%** success rate for the mediated durability test.

5.5 Results – Scalability

Another measurement that is specified in chapter 3.3 is XMPP's power of scalability. To measure this, number of clients was increased to 4. Files were transferred simultaneously between the clients to see what impact this would have on the XMPP server. Using the direct connection for scalability tests was not possible due to firewalls. As shown in Table 17 and 18 each of the scalability tests was divided into sessions consisting of two transfers involving 4 clients. Each session is described in the column "*From-To"* in both tables where direction of file transfer and involved clients are listed. The smallest file (tfile1.dat) was not used during these experiments.

5.5.1 Scalability with XEP – 0047 In-band Bytestreams

The in-band connection setups only allow the client to transfer files in one direction at a time, giving the setup a limitation for manual testing. This resulted in the clients only participating one time in each session as shown in Table 17. The table shows that both the RAM and server load were increased during these tests.

Table 17 Results - In-band connection – Scalability

Session	Size (Kb)	From - To	Transfers	Failed	Speed	CPU / Ram Load(512 mb)
1	10240	A – B & C – D	5	0	0,031 MBps	7 % / 8 %
2	51 200	A – C & B – D	5	0	0,029 MBps	8 % / 11 %
3	102 400	D – A & B – C	5	0	0,027 MBps	8 % / 14 %
4	204 800	B – D & C – A	5	0	0,024 MBps	8 % / 19 %

5.5.2 Scalability with XEP – 0065 SOCKS5 Bytestreams

The same test was performed for the mediated connection, to achieve a heavier load on the server. The average speed and server load during the transfers is shown in Table 18. During the first phase of scalability testing on the mediated connection we replicated the experiment done with the in-band setup. This was done to make grounds for comparison on the hardware load.

Table 18 Results - Mediated connection - Scalability

Session	Size(Kb)	From - To	Transfers	Failed	Speed	CPU / Ram Load(512 mb)
1	10240	A – B & C – D	5	0	0,85 MBps	2 % / 3 %
2	51 200	A – C & B – D	5	0	0,74 MBps	2 % / 3 %
3	102 400	D – A & B – C	5	0	0,72MBps	3 % / 4 %
4	204 800	B – D & C – A	5	0	0,70 MBps	3 % / 4 %

Since the mediated results in Table 18 was gathered by replicating the experiment from the in-band scalability tests, we decided to increase the load further for the mediated scalability tests. In an attempt to do this, the session 3 and 4 transfers were performed simultaneously. This was done manually and the session 3 transfer was started about 10 seconds into the session 4 transfer. The results from session 4 and 3 combined are shown in Table 19.

Table 19 Results - Enhanced mediated connection - Scalability

Session	File/Size (Kb)	From - To	Transfers	Failed	Speed	CPU / Ram Load(512 mb)
3 and 4	tfile3.dat(102 400) & tfile5.dat (204 800).	D – A & B – C B – D & C – A	5	0	0,54 MBps	3 % / 6 %

5.6 Mean time between failures

As a total overview of the file transfer capabilities of the XMPP protocol, the measurement MTBF (mean time between failures) will be used. This factor is found by summing the defined values from the measurement specifications (Chapter 3.3).

Table 20 Results - Mean time between failures

Variable	Description	Value	Comment
T	Amount of time with active transfers	343133 seconds	5710 minutes
N	Number of attempted file transfers	730	
S	Number of successful transfers	709	
F	number of failed transfers during T	21	

The values in table 18 are all based on the recorded operations performed within the production environment. Based on these values one can calculate the MTBF

MTBF = T / F = 343133 / 21 =**16339seconds**

This shows that the average time between each failed file transfer during the recorded part of the experiments is 272 minutes.

5.7 Base64 encoding – decoding performance

As a part of the performance enhancement it was interesting to see what performance the base64 operations would have on the experiments. As shown in table 7 and 8, the base64 encoding and decoding doesn't introduce remarkably large time consumptions. The performance here will also be reliant on what CPU power present in the computer used for decoding/encoding. As mentioned in chapter 2 the base64 operations create vast amounts of overhead, and this becomes clear when studying these results. The average overhead for these test encodings is about 40 %.

Table 21 Results - Base64 encoding

File	Size (Kb)	Post base64 encoding size (KB)	Time
tfile1.dat	1024	1404	0,5 seconds
tfile2.dat	10240	14033	0,75 seconds
tfile3.dat	51 200	70 163	2,32 seconds
tfile4.dat	102 400	140 326	3,44 seconds
tfile5.dat	204 800	280 652	4,75 seconds

Table 22 Results - Base64 decoding

File	Size (Kb)	Post base64 decoding size (KB)	Time
tfile1.dat	1404	1024	1,1 seconds
tfile2.dat	14033	10240	2,21 seconds
tfile3.dat	70 163	51 200	3,72 seconds
tfile4.dat	140 326	102 400	5,13 seconds
tfile5.dat	280 652	204 800	7,75 seconds

5.8 Telemedicine solution for XMPP based file transfer

Based on the experiences and the knowledge gathered from this thesis work, we have proposed a design for a telemedicine solution for file transfer based on XMPP. The design consists of functionality descriptions that we believe should be incorporated into such a solution. The solution consists of a client, a server, and an administrative interface on the server side.

5.8.1 Scenario

A solution for file transfer should be versatile in regard to what environment it could be deployed in. During this thesis we have been focusing on a health network environment, and this proposed solution is designed for use in such an environment.

The proposed solution is designed around a master node (server) for the purpose of administration. The server will not be storing any of the medical data transferred between the participating entities. The server should be placed within the health network where all participants will have to enable connectivity according to previous discussed network requirements for file transfer. The definition of a client in this scenario would be a hospitals radiology department. Since transfer of files of these sizes mostly is performed between larger health care organizations i.e. hospitals, the solution will be focused towards such participants.

5.8.2 Client

We have not considered graphical design issues when describing the proposed client solution. Keywords for a graphic interface should be intuitivism and ease of use.

Functionality overview

- Base extensions - The client should be implemented with utilization of the *XEP – 0096 File Transfer* [22] extension providing seamless file transfer. This indicates implementation of both the *XEP – 0047 In-band Bytestreams* and *XEP – 0065 SOCKS5 Bytestreams* extensions. By implementing the 0096 extension one covers up any discrepancies in protocol negotiation that could result in transfers failing. The default mechanism for file transfer should be the mediated (proxy) option that has the best performance in transfer speed. All other extensions needed for basic functionality of XMPP communication setups is not considered in this design.

- Client Configuration – There should be as little as possible configuration performed on the server side. If possible this should be limited to URL / IP to server, login credentials, and path to file storage where the medical data transferred should be stored for retrieval and transmission. Since communication partner data is stored on the server the client

implementation would not have to extend its functionality towards such tasks. Adding, deleting and editing communication partners are performed on the server side.

- Presence utilization – The basic presence functionality handled by the server on an instant messaging solution, consist of setting your status in the client. This could be setting yourself as away, occupied, out to lunch or any other choice of status. If you set your presence as *busy* in an instant messaging client, your communication partners could still send messages to you. We propose to extend this possibility to also control the communication flow between the participants and to add an extra layer of security to the actual file transfer. If Radiology A (Rad A) is *occupied* with receiving a file from Radiology B (Rad B) , the presence status on both clients should be set to *occupied,* or any other predefined values that describes the client status. This should also result in negative negotiation of any attempted file transfers initiated by Radiology C (Rad C) during the session between A and B. Radiology C would have to ask the server what presence status the intended receiver has. If the server responds with presence of receiver is *occupied*, Radiology C would abort the intended transfer and wait. By adding this functionality on would eliminate problems related to scalability where the client cannot handle simultaneously transfers. Possibilities of tweaking these settings should be discussed when files are smaller, but for large files one should make sure such situations are reduced. Period of waiting or other possible operations like queue handling, will have to be considered in building a data model for logic in client handling. This is not taken into consideration in this design.

- Security – The development of the solution will have to consider if any secure protocols should be used for communication between the participants. The data transferred from A to B will have to be encrypted according to laws and regulations and this will have to be implemented besides the client. Any encryption on the communication channel should be addressed as an extra level of security. XMPP has the option of implementing TLS [36] as part of its services, and a description of how the TLS handshakes are negotiated in XMPP is presented by Liuhto [41].

- Addressing – Building unique JID's to both recognize and maintain the communication partners in the solution. The JID's should be built based on a standard for naming and addressing that the developers should establish.

- Local and remote storage – Storage of the data communicated between the communication partners is an issue that needs to be addressed in a larger scale. The ultimate solution would be to integrate the XMPP client as close as possible to the modality producing the data i.e. an MRI machine. If the client is supposed to be the main transfer solution for an entire organization (hospital i.e.) it has to be interacting with API's from the different systems producing medical data intended for transfer.

5.8.3Server

On the server side we have many options to base a possible file transfer solution on. Several of the existing server implementations would provide for reliable and satisfactory services to the proposed solution. Since most of this software is based on open source, one has the option of stripping one of the implementations of its excess services and, utilize the functionality needed for file transfer and roster handling.

- Presence overhead – The level of traffic between clients and server related to awareness and presence could be minimized in a file transfer solution. By running a pilot system one could build a trend analysis providing information about what frequency the different communication parties interact with each other. Based on this one could decrease the amount of presence communication since all connected clients probably won't be involved in the same amount of activity. Another option is to implement a solution where each client constantly updates a presence table on the server, and any client that wants to interact with another part has to ask the server for last updated presence on the communication part in question. This would cut the amount of presence traffic to a minimum but it is not the ultimate solution for keeping the participants constantly updated. A third option would be seen as a extension to the presence utilization mentioned on the client side. By adding a layer to presence utilization that forces the clients to ask for presence before initiating any file transfers, one would limit the traffic. If Rad A wants to transfer a file to Rad B, he would have to ask Rad B (through the server) what presence he has. Rad B would answer the request and return the answer in-band to Rad A. To avoid simultaneous requests and answers, both clients should be set in occupied mode until the initiated session has ended with either a file transfer or a denial from Rad B.

- Roster – The server would keep track of all "contact" information that is stored in rosters for each client. Administration of these should be centralized and done through a roster administration interface that we also have proposed.

- Logging – The server should add a layer of integrity check to the solution by logging each file transfer session and every negotiation performed between the clients. This is not confidential data and could be stored on a centralized unit like the XMPP server itself.

- Broadcast possibilities – The server should utilize the possibility of broadcast that is implemented in most RTC's today. This is useful when system maintenance is performed or if any unforeseen incidents have occurred. The server should then be able to set all clients to status as occupied and instantly update presence to everyone through broadcast functionality.

5.8.4 Roster administration

As an add-on to an existing RTC's default setup, an implementation of a roster administration tool should be developed. By default the XMPP user roster is built up by adding participants from the client side, and accepting storage on the unique roster belonging to each client. This should be moved to the administration tool, giving operating personnel control of what clients should have enabled the communication setup. This also removes any difficulties for the users (hospital departments) and increases the level of reliability and integrity in the solution. Users on the client side should not have to interact with the client at all, except maybe starting it if necessary.

The roster administration should enable:

- Operating personnel to add edit and delete single clients (users).
- Operating personnel to link and add clients to the unique rosters.
- Manually setting presence status.
- Definition of any variables that can be utilized by the clients.
    - Client wait/sleep time
    - Queue status
    - Schedules; should the clients have periods during the day where file transfers isn't performed.

A roster administration tool could be seen as a dynamic part of this system. Since the base functionality in both clients and server are implemented to support its intended purpose, the roster administration would not affect the lower level of operability. Any administration that would include the roster of either client could eventually be moved to a roster administration tool.

6 Discussion

When discussing the results from the experiments performed, it is necessary to have something to compare it with. Since no existing solution was available for transfer of large files, the comparison has to be done towards other possible solutions for transfer of large files. The closest and most relevant technology to compare with is FTP, which is designed for file transfer.

6.1 Efficiency performance

As the measurements in Chapter 5 shows, the overall performance of the two transfer solutions based on XEP-0065 SOCKS5 Bytestreams is acceptable compared to FTP. Both of the SOCKS5 based transfer mechanisms has a throughput that equals and even exceeds FTP in the pilot tests. It is evident that the biggest influence on performance when using SOCKS5 and FTP is the capacity of the network. As shown in Table 8 the average speed when using the mediated connection method exceeds the average speed when using FTP. This delay of 0, 4 seconds in the FTP protocol is a consequence of negotiation between client and server when starting each file transfer. When all files where queued in the FTP client, the average time was lowered to 1.5 seconds. This would indicate that the XMPP technology is a feasible option for file transfer, simply based on its performance.

When the communication setup was deployed within the health network, the overall performance of the transfers dropped considerably. This is of course related to the capacity of the network connection where the XMPP server is placed. The SOCKS5 extension still provides results within the pre defined requirement for efficiency in chapter 3.3, which makes it a probable mechanism for file transfer. Both the direct and mediated connection performs within the predefined requirements.

When comparing XMPP's two methods the in-band method has an overall performance that is several times slower than the best performing SOCKS5 solution. Table 23 shows the ratio between these two extensions.

Table 23 Ratio XEP - 0047 / XEP - 0065

File	IBB / SOCKS5	Ratio
tfile1.dat	24,8 / 1,6	1/15
tfile2.dat	257 / 9,5	1/27
tfile3.dat	1289 / 52,3	1/24
tfile4.dat	3652 / 114	1/32
tfile5.dat	7480 / 243	1/30

As the results show in Table 12 and 13, the average transfer speed is from 0.026 MBps to 0.040 MBps which isn't within the level of efficiency stated in the measurement specification (Chapter 3.3).

The difference in performance between the two extensions makes a solution based on the in-band method for file transfer less relevant. Also, compared to FTP, the in-band solution doesn't provide results that should imply that it has any advantages when it comes to performance.

After deploying the test environment in a cross-region environment as shown in Figure 18 and Appendix A, we collected performance material from the same environment and locations that an actual production ready solution would run. Based on the results shown in Table 11, the mediated file transfers performed in the cross-region setup only deviates by an average of about 0.10 MBps from the results in Table 10. This small difference is also a result of the high impact the average speed of *tfile1* has on these results. As seen in Table 10 and Table 11, the two measurements for *tfile1* are 0.38 MBps and 0.71 MBps. If the result for *tfile1* is left out, we get an average deviation between the two setups for the mediated transfer method of 0.21 MBps. This difference might be explained with the notion that the results in Table 10 were measured while the server was running only 64MB of JVM. The preface tests showed that the amount of JVM on the mediated connection wouldn't have a huge impact on the transfer speed, so we have accepted that the deviation would be related to network latency.

As mentioned in the SOCKS5 Bytestreams results part (Chapter 5.2) we experienced some deviations in transfer speed depending on what direction the transfers was done. When transferring files from Helgelandssykehuset to UNN Tromsø, we experienced remarkably slower transfer speeds. We performed troubleshooting on both server and client installations on both locations without getting any clear answers to this deviation. The transfers succeeded but the average transfer speed was slower. Later on it turned out this was something related to the LAN on Helgelandssykehuset, where there were some interface problems between the different equipment used in the network.

For the in-band experiments performed in the cross-region network setup, the average transfer speed is about 0.3 MBps slower than the experiments from the single region setup. This is a result of the network environment which in this experiment is much more complex. In this experiment we also experienced that 4 of the transfers was terminated about halfway into the session. We found that the terminated transfers were a result of the client loosing the established connection to the server. As explained in Chapter 2, the connection is re-established, but the file transfer session is lost and has to start over. These problems were only experienced during the transfer of the two largest files. As discussed in the theoretical background (Chapter 2), the transmitter doesn't get his acknowledgement for each package; the transfer session is terminated and presumed broken. This is a very good example of what difficulties the in-band solution will meet when operating in a complex network environment. The difficulties related to what direction the transfer was initiated was also noticeable in the in-band experiment. However as mentioned in the results (Chapter 5.3) the average speed for in-band traffic is so slow that the difference is not considered an issue.

As mentioned in the system requirements (Chapter 3) the calculation of the average speed for the in-band solution is based on the initial file size, not the actual amount of transferred data. This is a choice we made to have a common ground for comparing the performance of the two extensions. However, this is not beneficial to the in-band solution when presenting its performance. It has to be mentioned that the actual amount of data transferred in the in-band experiments is about 30 % higher than its initial size. If calculating the average speed from the actual amount of data transferred, the in-band solution would have a jump in performance of about 30 % overall.

Figure 19 Graph comparison - Performance all methods

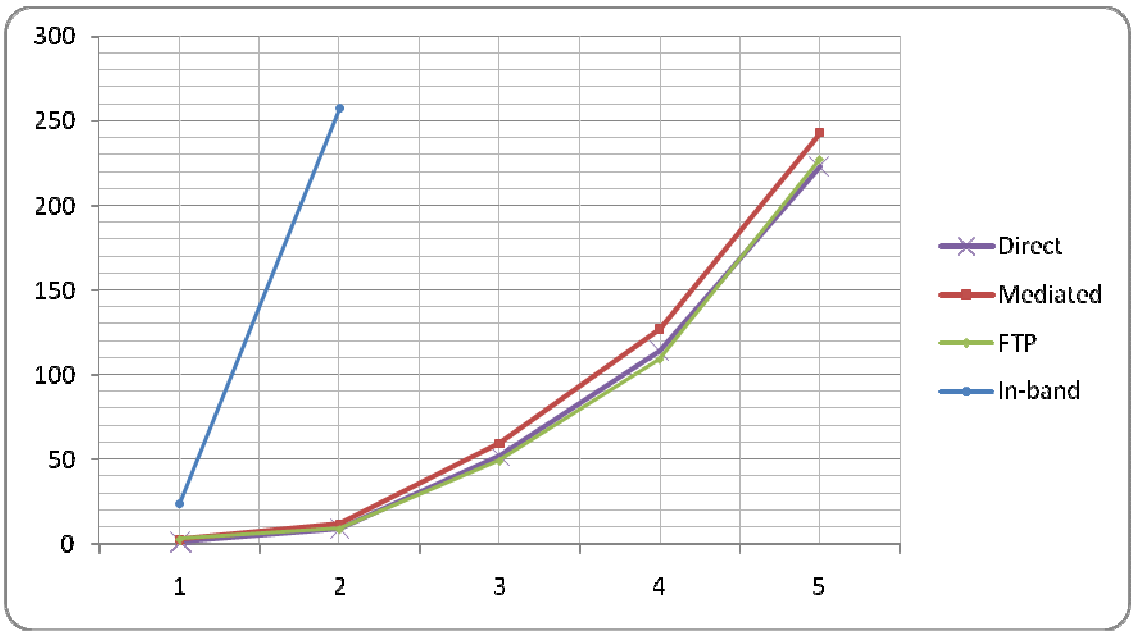


Figure 19 shows 4 of the used methods for file transfer and the comparison between them. Along the X-axis the different test files (1-5) is listed, and the Y-axis is enumerated in seconds. The figure shows that the in-band method escalates in time already at tfile2. To be able to show all methods in the same figure we have cut the in-band line after tfile2. The figure shows that the other three methods perform almost identically. The mediated method for transfer performs slightly better than both FTP and the direct connection.

Figure 20 Graph comparison - JVM Load

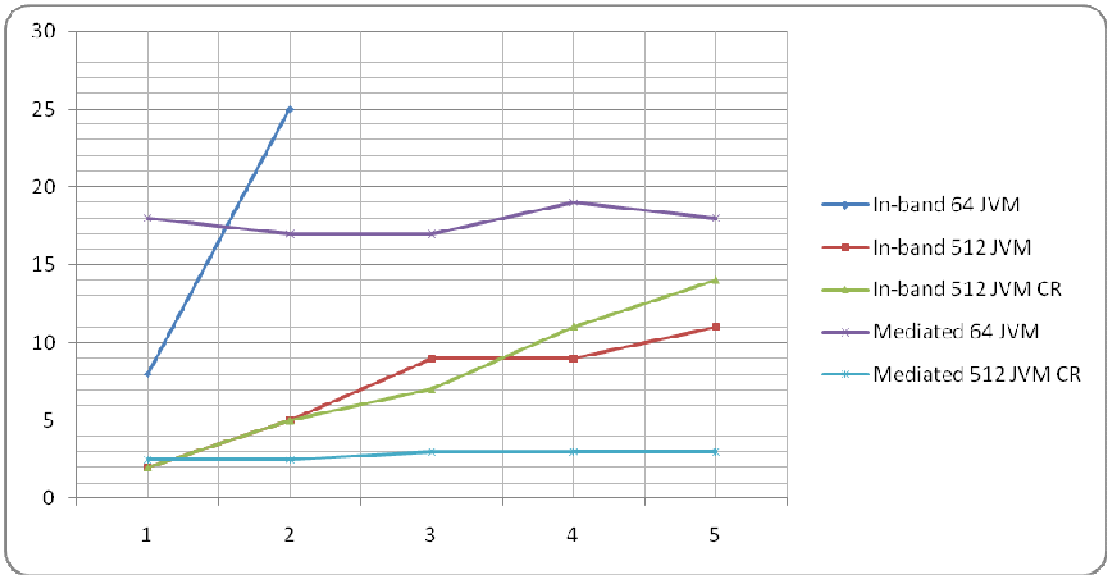


Figure 20 shows the load each of the setups had on the Java memory on the server installation. The Y-axis is formatted with numbers 0 – 30 indicating percentage (%) and the X –axis as in Figure 19 shows the different test files. The In-band 64 JVM line has been cut short since it increased to reach 100 % memory use when transferring testfile5. The Mediated 64 JVM line shows that the server was under heavy load during the mediated (proxy) file transfers performed on that specification. When increasing the JVM to 512 MB the figure shows that the Mediated cross region transfer used well below 5% of the available Java memory. There could be several aspects influencing these measurements, especially since we didn't use hardware dedicated to running server software. We have used desktop computers running standard Windows XP OS. These values could be experienced very different if the RTC was run on a real production server. However, these values give a very nice view of what problems the different methods of file transfer with XMPP might encounter.

6.2 Durability

The measurement of XMPPs performance of durability is another requirement that is defined in chapter 3.3. Because the file transfers was done manually the test periods was divided into time partitions of 2 x 12 hours for the in-band transfers and 2 x 3 hours for the mediated transfers. A number of transfers were performed during these time periods, counting the total number of transfers, number of successful and number of failed transfers. The in-band results are shown in Table 15 displaying an amount of 80 transfers performed over a total of 72 hours. During this time period 3 transfers failed due to connection drops between the server and client. For the mediated option a total of 250 transfers were performed during a total time span of 18 hours, with 4 failed transfers. The failed transfers were also related to connection drops between client and server.

Based on the values registered during the durability tests it is evident that the protocol has capabilities of providing reliable service over periods of time. Of the total 7 failed transfers, 4 of them are directly related to the poor hardware used in the server computer, and the last 3 is related to network issues with connection drop. Considering that this network setup is not accommodated to supporting dedicated file transfer solution the values for measured durability is very good. By adding values from the entire experiment into the equation, and to argue that a file transfer solution would support both in-band and mediated transfers, one could join the values from these tests and get a fail rate of 2, 1 % from a total of 330 transfers. This makes the durability of the XMPP solution acceptable considering the operating environment and the methods used for collecting the data.

6.3 Operating environment

One of the topics stated in the problem definition is the deployment of services like this in a network environment intended for medical and health related applications and services. A “health network” is not that different from other networks when it comes to technologies and components, but the data transferred within the network is placed under very strict laws and regulations. Since much of this data is personal sensitive information the measures taken to protect the data is much more complex than in a regular LAN. To add complexity, each health region might have their own adaption to fulfill the imposed security measures. When deploying a new service into such a sphere, there are many issues that need to be considered by both the organization buying the service, and the system engineer developing it. As mentioned in chapter 2.1.3, there are several issues that need to be clarified before services are accepted as a part of the operation.

As development of a standards draft [30] has been announced one has to assume that the level of information security and integrity on both infrastructure and administrative levels are satisfactory. Deploying a new service into an environment where demands and requirements have no definitions of standard and might change rapidly is a very difficult process. It is difficult not only for the developer but also for the operating department that might end up with a solution that demand a lot more resources than the organization had expected. In a health network there are many participants with unique and strict regulations of what services could be implemented into their environment. What level of security and what degree of intrusion of daily and established routines the new service will have are often key factors in negotiating deployment of new solutions. Any benefits the base technology might have in these negotiations could bring both the developers and organization closer to the common ground providing a reliable and secure solution for their services.

6.4 Firewalls

Seen from a system engineer’s point of view one of the most important issues are the traffic that needs to be allowed between the interacting parts of the communication setup. The network consists of several firewalls controlling access to and from the computers and systems, and a new service will have to get allowed traffic to and from its dedicated ports. As mentioned in chapter 2.13, there are three categories for the action of allowing traffic. The three categories are defined as 0 open ports, 1 open port and at last many open ports. This doesn’t only affect the actual work that needs to be done, but also the security and risk analysis that follows such an action.

If the In-band mechanism in XMPP was to be used for any service within such a network it would actually only need to use 1 port for the purpose of transferring files. Opening port 5222 or 5223 will enable a connection between the client and the server, and all communication might be done in-

band. The other option for file transfer, SOCKS5 would need to use the proxy option for the file transfers. It would not be possible to use the direct-connection if either of the clients is placed behind a firewall. The proxy solution depends on opening a second port for transfer of the file. In the test setup in this thesis port 7777 has been used as the proxy connection. The communication and control stream is still performed over the standard ports defined by the XMPP protocol. This is in many ways similar to the FTP protocol. As mentioned in chapter 2.5, the FTP protocol uses port 21 for its control stream, and the actual data transfer is done over port 20 or other depending on the file transfer mode. This isn't all to different from the SOCKS5 proxy solution implemented in XMPP and delivers the same amount of work related to opening ports in firewalls.

The two extensions used for file transfer in this thesis have also been bundled into an extension called XEP-0096 File Transfer[22] . This extension implements seamless file transfer giving the client a fallback option if the preferred choice of transfer method doesn't comply. This does in fact start the file transfer using the in-band solution if any of the two SOCKS5 methods fails. Even though the in-band solution would perform slow, the file would reach its intended recipient.

6.5 Scalability

Scalability was one of the predefined requirements that were the most difficult to measure. This is mainly a result of the software used for the experiments. To establish a scalable solution, we had to install several clients and they had to transfer files simultaneously over the communication channel. Up to 6 clients where connected to the XMPP server at the same time, but only 4 of them participated in actual file transfer. Two of the clients where used to continuously change their presence status to see if there were any noticeable performance issues affecting standard operations. The in-band method could only transfer data one direction at a time, so it was only possible to achieve 2 simultaneously transfers using the in-band method. As shown in table 15, the performance of the in-band transfers dropped when two transfers was performed at the same time. CPU and RAM load where also considerably higher during this experiment. This is not a surprising result since the in-band method is heavy on the system by default, it would logically increase further when increasing the amount of traffic over the in-band channel.

In the scenario used for the experiments a setup consisting of two radiology departments and a communication link, the XMPP server, is used to transfer files from one department to the other. Using FTP in its basic form indicates using a similar setup with FTP clients and a FTP-server to relay the files. Adding more departments as senders and receivers using XMPP would need to add a new JID to the roster of each XMPP client. The roster would be stored and kept updated by the presence functionality built into XMPP. By adding a new department, the transmitting XMPP client just has to

address the file with another JID, or implemented addressing mechanisms, and the file transfer would be possible. Using FTP this would be a bit more complicated; since a file transmitted from Radiology A to Radiology B would have to be addressed in some way, one would probably have to use more than 1 FTP-server to achieve this. Each FTP-client has their own user account on the FTP-server, allowing them to login and upload or download a file. The client itself doesn't by default know who will be downloading the file after it is stored on the FTP-server. This means that the downloading client will have to be tuned towards looking for a unique identifier in the filename, or to check a unique account on the FTP-server, appurtenant to each FTP-client. This would result in a complex structure if there is to be enabled an N: N relationship based on the FTP protocol. It would probably be possible to develop dedicated and automated solutions for transferring files without having these scalability issues related to the base functionality of FTP.

XMPP also has its scalability issues by its default nature, especially related to presence overhead. A lot of data is transmitted through the XMPP server to keep all clients statuses updated on all connected clients. In the scenario setup for the experiments in this thesis, presence overhead would be unnoticeable hence the number of clients is so small. When the number of connected clients reaches 500-1000 the amount of presence data communicated through the XMPP server would be noticeable. This issue is something that would have to be addressed when implementing a telemedicine solution based on XMPP, where the traffic between the communication parties is of a different nature.

The direct connection wouldn't impact the server performance as the connection is a P2P link directly between the clients.

6.6 Store and forward

Another issue that should be mentioned as positive with the use of XMPP as the transfer solution, is its lack of centralized file storage. Consider departments Radiology A and Radiology B using their dedicated file transfer mechanism based on XMPP, the file transferred from A to B would never be stored on any third part location. It would be transferred directly from the transmitting department to the receivers department. This would be the case for departments taking part in the file transfer solution. Assuming that each department has their systems and medical applications bundled with an approved XMPP implementation, files could be directly transmitted from the medical application and transferred to the receiving department. One would never experience files being stored on remote locations for a period of time while waiting for it to be picked up by the receiver. With XMPP, if the receiver isn't available the transmitter will be notified of this by the built in presence functionality.

By using FTP in its basic form, all files need to be stored on a FTP-server awaiting the receivers download. After the receiver has completed the download, the file needs to be deleted in a safe and non-reconstructable matter. If the receiver for some reason should experience any trouble, the file will be stored on the FTP-server. The risk of data loss or unauthorized access might not be any bigger on the FTP-server than other storages, but the by an intrusion on a centralized storage unit a lot more data is accessible to the intruders. Centralized storage of files is something that the data inspectorate is passing strict regulations against, and all such storages has to be placed under very rigid and strict security measures. The use of a rapid accessible FTP-server for such purposes is in direct violation with such guidelines.

6.7 Operating costs

The need for resources and operating costs might also be more satisfactory by using an XMPP based solution for file transfer. Based on issues mentioned in both scalability and store and forward sections, one could argue that the FTP based solution would demand a higher level of interaction from the operating department. As mentioned in chapter 2 this interaction would be defined by what level of intrusion the new solution would have on the existing routines and level of security. This is not only a subject of network and technological aspects, but also administrative decisions that needs to be made by the organization. Based on our assessment we argue that if measurements where done towards the amount of sources that could fail in the two compared technologies, we believe that FTP under its base functionality has more possible elements of failure. Seen from an organizational perspective and a possible customer of such a solution, one would have to assume that the final product would perform according to intended purpose whether it is based on FTP or XMPP as the backbone.

6.8 Conclusion

Given that a proposed solution needs to transfer large files, the in-band solution does not achieve the proposed levels of performance. It performs to slow and the load on both the hardware and software used is unacceptably high. However for smaller files and even non-urgent transfers, the in-band solution has some elements that give it a possible level of acceptance. As a solution for transferring large files in any network setting, it does not meet the needed level of performance.

The SOCKS5 solution has its limitations when it comes to network security like firewalls and NAT-ed network devices. It relies on a proxy placed outside the LAN's involved in the communication link, or that the clients transferring the files have a possibility of publishing its network information. In a NAT-ed network a direct connection will not be possible, and a proxy is the only solution. Its performance is satisfactory compared to both the pre-set requirements and other solutions for file transfer. Given that introduction of a file transfer solution will have to involve interaction from network operators despite the technology used, XMPP delivers several positive and new aspects that makes it feasible as a choice.

As a measurement of what level of reliability the XMPP technology delivers, we defined the "Mean time between failures" variable. As described in the results presentation (Chapter 5) this variable has been calculated to: MTBF = T / F = 343133 / 21 =16339seconds = **4, 53 hours.** Considering the software and especially the computer equipment at our disposal during these experiments, we would argue that this is an acceptable level of reliability.

We have also proposed a telemedicine solution for file transfer using XMPP. This designed is based on the experiences gained during the work on this thesis. We believe a solution like this could provide the necessary service to the users and that it could be integrated into a health network.

7 Future work

The XMPP technology was used for file transfer in this thesis. Even though it is not the intended task we have shown that it is a feasible solution in the given environment. Through my position as an EDI consultant at Helse Nord IKT I have discovered some scenarios where XMPP might be utilized in other ways.

One of the most important areas of the electronic data interchange today is the transfer of emergency referrals. Throughout the health region, most of the primary health care providers are connected to the health network and has the possibility of sending emergency referrals to the hospital. This can be referrals to radiology or to hospital paramedic departments. Both in rural areas and in cities there are casualty wards with 24/7 duty as a public service, in case inhabitants would need emergency medical treatment. In some cases a patient arrives at the hospital department for an emergency consultation before the accompanying referral has arrived. There can be several reasons for this, but on the regular basis the referral arrives within 15-45 minutes after it has been produced by the referring doctor. These referrals are transmitted from the doctors' offices using the existing solution for transfer of medical data, based on POP/SMTP. The present solution provides satisfactory service in most situations. From the departments view, some of the referrals should arrive as fast as possible, and 15 minutes is often regarded as the maximum allowed time by the health care professionals. One could look into developing a service where the use of XMPPs instant messaging and presence capabilities is utilized for achieving the ultimate communication channel for the most critical emergency referrals. This could be done by developing a system consisting of clients and server implementations tuned towards the service intended, and built upon the basic features delivered by XMPP. It would be installed at the primary health care offices, and connected to servers at the different departments and hospitals that are potential receivers of these referrals. To achieve the level of operability needed one would have to include the vendors of the electronic patient systems used at the primary health care offices. This is a big project, but the XMPP technology might provide the needed level of rapidness needed for the most critical emergency referrals.
Another topic that should be investigated further is the possibilities of increasing the performance of the in-band extension. Could any technology on network or application layers be utilized better or with added features to increase the level of performance on the in-band functionality? As both network infrastructure and computer hardware has increased in level of capacity and capability the last couple of years, there might be some new aspects that could aid in increasing the in-band performance. One should take a look at topics like TCP Window sizing, increasing the default package chop size of 4096 and the possibility of re-designing the XML structure of the extension. We would argue that if the in-band method for file transfer had provided a higher level of performance it would have been an excellent option for a file transfer solution in an network environment as the one we have worked with.

References

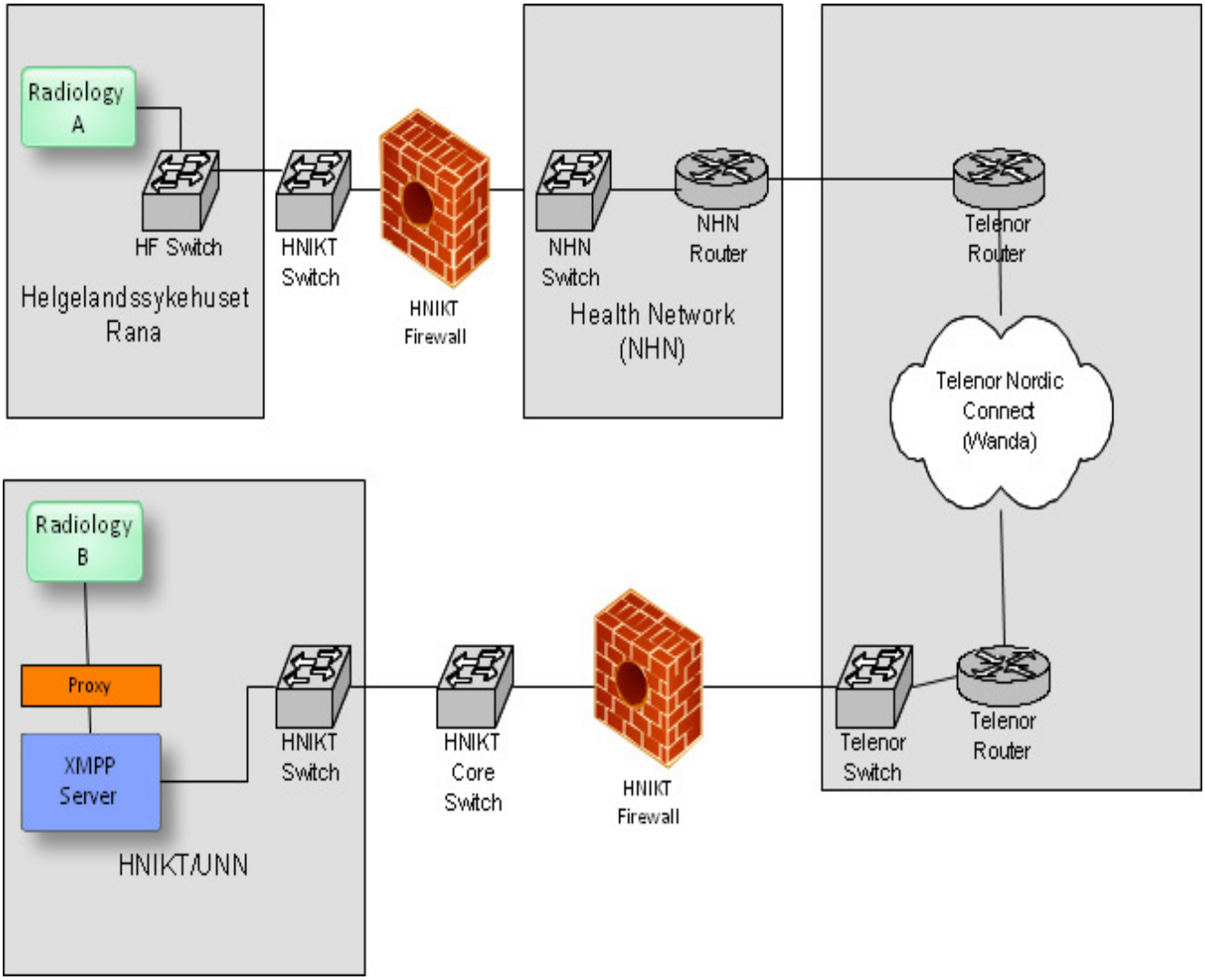
1. Ewasif.Wael, P.J., Beck.Micah, Wolski.Rich (1999) *IBP-MAIL: Controlled Delievery of Large Mail Files*. 9
2. Jabber.org. *Jabber - Overview*. 2005 [cited 2008 21.01]; Available from: http://www.jabber.org/about/overview.shtml.
3. Laukanen, M. (2006) *Extensible Messaging and Presence Protocol*. 7
4. Postel.J, R.J. *File Transfer Protocol (FTP)*. 1985 [cited 21.01]; Available from: http://www.ietf.org/rfc/rfc0959.txt.
5. Saint-Andre, P. (2005) *Streaming XML with Jabber/XMPP*. **Volume 9**, Issue 5
6. XSF and X.S. Foundation. *XMPP Extensions*. 1999-2008 [cited 2008; Overview of published extensions to the XMPP protocol.]. Available from: http://www.xmpp.org/extensions/.
7. Karneges.Justin (2006) *XEP-0047 : In-Band Bytestreams (IBB)*. **1.1**,
8. Smith.Dave, M.M., Saint-Andre.Peter (2007) *XEP-0065: SOCKS5 Bytestreams*. **1.7**,
9. Saint-Andre, P. *Extensible Messaging and Presence Protocol (XMPP): Core*. 2004 [cited 2008 21.01].
10. XMPP.ORG. *XMPP Standards Foundation*. 1999-2007 [cited 2007 09/10]; Available from: http://www.xmpp.org/.
11. Saint-Andre, P. (2006) *XEP-0066: Out of Band Data*. **1.5**,
12. Software, J. *OpenFire Ignite Realtime*. 2008 [cited 2007 26.12]; Available from: http://www.igniterealtime.org/projects/openfire/index.jsp.
13. eJabberd. *ejabberd - the Erland Jabber/XMPP daemon*. 2008 [cited 2008 15.01]; Available from: http://www.ejabberd.im/.
14. Tigase.org. *Tigase Open Source and Free Jabber/XMPP environment*. 2008 [cited 2008 18.01]; Available from: http://www.tigase.org/.
15. Baardsgaard, A., *The Scandinavian Health Network: Connecting the Scandinavian countries' health networks*, in *IFI*. 2006, UIT. p. 77.
16. KITH, *Paraplyprosjektet: Digital røntgen, PACS og RIS*. 2002, SHDIR. p. 43.
17. Helsenett, Ø.-N., *Multimedianettverk mellom sykehus*. 2002, Norges forskningsråd. p. 5.
18. Ilias Sachpazidis, O.H. (2005) *Instant messaging communication gateway for medical applications*.
19. W3C. *Extensible Markup Language (XML)*. 2008 [cited; Available from: http://www.w3.org/XML/.
20. Saint-Andre, P., *RFC3920: XMPP Core*. 2004, Jabber Software Foundation.
21. Parziale Lydia, B.D.T., Davis Chuck, *TCP/IP Tutorial and technical overview*. 8 ed. 2006: Redbooks.
22. Muldowney, T., M. Miller, and R. Eatmon (2004) *XEP-0096: File Transfer*. **1.1**,
23. Josefsson, S., *RFC4648: The Base16, Base32, and Base64 Data Encodings*. 2006, Network Working Group.
24. Morin, R.C. (2001-2002) *How to Base64*.
25. M. Leech, e.a., *SOCKS Protocol Version 5*. 1996, Network Working Group. p. 9.
26. Phifer.Lisa. *The Trouble with NAT*. 2000 [cited 2008 02.02]; Available from: http://www.cisco.com/web/about/ac123/ac147/ac174/ac182/about_cisco_ipj_archive_article09186a00800c83ec.html.
27. E. Zwicky, e.a., *Building Internet Firewalls*. Second ed. 2000: O'Reilly.
28. Klensin, J., *RFC2821 : Simple Mail Transfer Protocol*. 2001, AT&T Laboratories.

29. Ford-Hutchinson, P., *RFC4217: Securing FTP with TLS*. 2005, Network Working Group.
30. SHDIR. *Norm for information security in Healthsectors*. 2006 [cited; Available from: http://www.nhn.no/informasjonssikkerhet/norm-for-informasjonssikkerhet-i-helsesektoren.
31. Jørgensen, V., *Personal Communication*, K. Andreassen, Editor. 2008.
32. Wikbo, L.-A., *Personal Communication*, K. Andreassen, Editor. 2008.
33. Cheng, N., *Dummy File Creator*. 2002, Nikko Cheng.
34. Microsoft, *File Checksum Integrity Verifier utility*. 2007, Microsoft.
35. Myers, J., *RFC2222: Simple Authentication and Security Layer (SASL)*. 1997.
36. Dierks, T. and E. Rescorla, *RFC4346 : The Transport Layer Security (TLS) Protocol Version 1.1*. 2006.
37. Software, J., *Spark*. 2008, Jive Software. p. Spark is an Open Source, cross-platform IM client optimized for businesses and organizations. It features built-in support for group chat, telephony integration, and strong security.
38. Shchepin, A., *Tkabber*. 2007.
39. Hefczyc, A., *Personal communication*, K. Andreassen, Editor. 2008.
40. Software, J., *Openfire*. 2008. p. Openfire (formerly Wildfire) is a cross-platform real-time collaboration server based on the XMPP (Jabber) protocol.
41. Liuhto, L. and V. Mäntysaari (2005) *Extensible Messaging and Presence Protocol*. XMPP,

Appendix

A - Cross-region network setup

A - Cross-region network setup



This setup of this figure is owed to Torstein Meyer at Helse Nord IKT for de-briefing me in what network infrastructure is present between and within the two regions.

Traceroute – UNN/HNIKT -> Helgelandssykehuset

Tracing route to 172.yy.xxx.x over a maximum of 30 hops

1	<1 ms	<1 ms	<1 ms	Cat-Core-1.rito.no [172.yy.x.x]
2	<1 ms	<1 ms	<1 ms	telenor-unn-gw0-200.rtr.nhn.no [10.145.x.x]
3	*	*	*	Request timed out.
4	24 ms	25 ms	24 ms	172.yy.x.x
5	27 ms	27 ms	27 ms	172.yy.x.x
6	28 ms	27 ms	28 ms	hsrp-rana-gw-200.rtr.nhn.no [10.145.x.x]
7	28 ms	27 ms	27 ms	172.yy.x.x