Hochschule Darmstadt

- Fachbereich Informatik -

Trustworthiness in Peer-to-Peer Communication for Commercial Applications

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

vorgelegt von

Roman Korn

Betreuer:	Prof. Dr. Michael Massoth
Korreferent:	Prof. Dr. Harald Baier

Ausgabedatum:	13.04.2010
Abgabedatum:	13.10.2010

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 13.10.2010

Kurzfassung

Peer-to-Peer (P2P) ist ein weit verbreitetes und ausgiebig untersuchtes Paradigma der Kommunikationstechnologie. Es wird erfolgreich im Bereich der Ressourcenverteilung eingesetzt. Ressourcen können in diesem Kontext als beliebige, große Datensätze (z.B. Multimedia-Inhalt) betrachtet werden. Obwohl P2P-Systeme viele Vorzüge mit sich bringen, sind bisher kaum kommerzielle Anwendungen zu finden. Das Fehlen adäquater Sicherheitsmechanismen scheint Unternehmen davon abzuhalten diese Technologie einzusetzen. Einige Bedenken werden offensichtlich, wenn man es mit dem traditionellen *Client/Server* Modell vergleicht. Im Gegensatz zum traditionellen Modell werden Ressourcen auf der Kundenseite, dem so genannten *Peer* (engl. Nachbar), vorgehalten. Jeder Peer darf Ressourcen mit jedem anderen Peer austauschen. Ein solches direktes Zusammenspiel unter Peers ist durch den Ressourcenanbieter viel schwieriger zu kontrollieren als ein geschützter, zentraler Serverbereich. In Geschäftsmodellen, bei denen vertrauliche digitale Ressourcen vertrieben werden, müssen einige sicherheitskritische Fragen neu beantwortet werden. Insgesamt muss studiert werden, wie ein Peer so vertrauenswürdig wie ein geschützter Server des Client/Server Modells werden kann.

Aus diesem Forschungsbereich liegen bereits einige konzeptionelle Ergebnisse vor. Insbesondere die Protokollspezifikationen des *Nano Data Centers* (NaDa) Projekts wurden gestaltet, um Sicherheitsmechanismen in P2P Netzwerken zu verankern. Der vorgeschlagene Ansatz beruht auf Erweiterungen aus dem Bereich des *Trusted Computing* (TC). Allerdings konnte die Umsetzbarkeit des Konzepts bisher nicht erwiesen werden. Es muss gezeigt werden, dass die Protokolle implementiert werden können und dass das Ergebnis ein realistisches Leistungsverhalten aufweist. Dafür wird ein Demonstrator in einem physischen (nicht-virtuellen) Labor installiert. Die Experimente im Labor werden durch das *cOntrol and Management Framework* (OMF) durchgeführt, um Reproduzierbarkeit und Objektivität der Ergebnisse zu unterstützen.

Der realisierte Demonstrator vereint eine Open Source P2P Anwendung, kryptographische Bibliotheken und auf Hardware basierende Sicherheitsfunktionalität eines *Trusted Platform Module* (TPM). Die Messtechnik und detaillierte Maße sind für externe Komponenten und alle Protokollschritte definiert. Eingeführte Schlüsselindikatoren ermöglichen Vergleichbarkeit mit anderen Implementierungen. Sowohl für die original- als auch für die erweiterte, vertrauenswürdige Anwendung wurden Experimente durchgeführt. Die Ergebnisse zeigen einen zu erwartenden Leistungsverlust im Austausch für integrierte Mechanismen, die Vertrauens-würdigkeit gewährleisten. Auf Grund des Protokolldesigns liegt der größte Anteil an der Gesamt-verarbeitungszeit noch vor der eigentlichen Durchführung des Protokolls. Leistungsschwache Bereiche und Engpässe konnten identifiziert werden. Die Ergebnisse ermöglichten es Verbesserungen zu konzipieren und umzusetzen. Insgesamt zeigt sich die Lösung als vielversprechender Kandidat für den kommerziellen Einsatz.

Abstract

Peer-to-Peer (P2P) is a widely used and extensively explored communication paradigm. It is successfully applied in the area of resource distribution which, in this context, may be seen as any large data file (e.g. multimedia content). Though P2P systems offer many benefits, commercial applications are still rare. The lack of adequate security mechanisms seem to preclude companies from deploying this technology. Some concerns become obvious when it is compared to the traditional *client/server* model. In contrast to this traditional model, resources are stored at the customer's site, the so called *peer*. Each peer may exchange resources with any other peer. Such a direct interaction between peers is much harder to control by resource providers. But controlling is important in business models where confidential resources are distributed in the P2P network, instead of keeping them in protected server areas. Who is allowed to share resources with whom? How can be assured, that only authorized peers access the network? If each peer becomes a server, how can its integrity be retained? In summary, it has to be studied, how a peer can become as trustworthy as the protected server in the client server model.

Some conceptual work is already done in this field of research. Especially, some protocol specifications of the *Nano Data Centers* (NaDa) project are designed to establish security mechanisms in P2P networks. The suggested approach relies on architectural extension from the area of *Trusted Computing* (TC). But there is no proof of concept available, yet. It has to be ensured that the protocols can be implemented and that the solution shows a realistic performance behavior. Therefore, a demonstrator is set up in a physical network laboratory which is expected to offer more meaningful measures than a simulation environment would. Experiments within the laboratory are deployed and conducted by the *cOntrol and Management Framework* (OMF) in order to support reproducable and objective results.

The solution is an implemented and evaluated demonstrator of a trustworthy P2P system. It integrates an open source P2P application, cryptographic libraries, and hardware based security functionality from a *Trusted Platform Module* (TPM). The introduced measurement technique is based on measures, that are obtained from system time stamps. Detailed measures are defined for external components and all protocol steps. Introduced key performance indicators provide comparability to other implementations. Appropriate sets of experiments are conducted, to provide statistical certainty of the results. Results are obtained for the original P2P system and the trustworthy P2P system. They show an expected performance loss in exchange for mechanisms that assure trustworthiness. Due to the protocol design, most of the overall processing time is located previous to the actual protocol processing. Areas of low-performace could be identified as well as bottlenecks. Based on the results, some optimizations are implemented and some suggested. Altogether, the solution has turned out to be a promising candidate for the commercial application.

Acknowledgement

Ich danke meiner Mutter Ewa, meiner Ehefrau Nataliya, meinem Sohn Niklas, meinem Bruder Daniel und allen lieben Verwandten und Freunden, die auf viel gemeinsame Zeit verzichten mussten um mir den notwendigen Freiraum für die Arbeit an dieser Thesis zu schaffen.

Besonders danke ich meinem Betreuer im Fraunhofer Institut, Nicolai Kuntze, der sich immer die Zeit für meine sinnvollen sowie unsinnigen Fragen genommen hat. Vor allem danke ich meinem Betreuer Professor Michael Massoth und meinem Korreferenten Professor Harald Baier für alle Anregungen und die viele konstruktive Kritik, durch die ich eine Menge lernen konnte.

Vielen Dank an Nico Lincke und Jürgen Repp, ohne die mir die vielen Arbeitsstunden lange nicht so viel Freude bereitet hätten. Ebenfalls bedanke ich mich bei allen Kollegen für die gelegentlichen Gespräche die mir neue Einsichten gegeben haben oder auch nur zu dem positiven Arbeitsklima beigetragen haben.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Description

Peer-to-Peer (P2P) is a widely used and extensively explored communication paradigm. It is successfully applied in the area of resource distribution which, in this context, may be seen as any large data file (e.g. multimedia content). Though P2P systems offer many benefits, commercial applications are still rare. The lack of adequate security mechanisms seem to preclude companies from deploying this technology. Some concerns become obvious when it is compared to the traditional *client/server* model. In contrast to this traditional model, resources are stored at the customer's site, the so called *peer*. Each peer may exchange resources with any other peer. Such a direct interaction between peers is much harder to control by resource providers. But controlling is important in business models where confidential resources are distributed in the P2P network, instead of keeping them in protected server areas. Who is allowed to share resources to whom? How can be assured, that only authorized peers access the network? If each peer becomes a server, how can its integrity be retained? In summary, it has to be studied, how a peer can become as trustworthy as the protected server in the client server model.

Some conceptual work is already done in this field of research. Especially, some protocol specifications of the *Nano Data Centers* (NaDa) project are designed to establish security mechanisms in P2P networks. The suggested approach relies on architectural extensions from the area of *Trusted Computing* (TC). But there is no proof of concept or characteristics of such an application available, yet.

1.2 Motivation and Goals

This thesis examines how a typical P2P application can be enhanced to a trustworthy P2P application. The application shall be integrated as a building block of the NaDa content delivery system. Software and hardware security mechanisms are incorporated, to reach this goal. The core tasks are the design and implementation of the communication application as well as its evaluation in relation to performance behavior. With the application, an already proposed protocol design from Kuntze et al. [KUN-2010] is realized. It consists of several concepts like the Diffie-Hellmann key exchange, remote attestation and ticketing to establish trust between multiple parties.

The evaluation offers valuable insight in direction of future development and commercial application. It includes the definition of experiment scenarios, a measurement technique, measures and key indicators. Results are reproducible and comparable. All experiments are conducted in a laboratory that is equipped with physical components (no emulators). Each participating entity of the Peer-to-Peer system is represented by a network node. All are interacting over a TCP/IP network. The *cOntrol and Management Framework* (OMF) is applied to deploy, monitor and control experiments.

Areas of low performance and bottlenecks are identified during evaluation and measurement. Based on the evaluation, concepts for an optimized application are discussed. Technical designs are created and implemented for promising concepts.

1.3 Commercial Application

Business application

The results of this thesis will be applied in a system for distribution of virtual goods called *Nano Data Center* (NaDa). A Nano Data Center is a network of platforms (similar to set-top boxes[1]), that are based on P2P and Trusted Computing technology. It is established and maintained by an *Internet Service Provider* (ISP). Virtual goods may be distributed by several *content providers (CP),* utilizing only one platform. Two main applications are characterized in an executive summary of the NaDa project [THO-2010b]. The business field of a pay-per-view VoD is in the main focus of this thesis:

- VoD: In context of the NaDa project, *Video-on-Demand* (VoD) applications are divided in the three types: *User Generated Content* (UGC), *Catch-up TV*, and *pay-per view.* For each, a different revenue model is defined. With generated content services, as provided by e.g. YouTube[2], the content providers could not earn significant revenues in the past [HUA-2007]. This is similar for Catch-up TV. These services provide television content at no charge. Typically, contents are available some days after the television broadcast. In the business field of pay-per-view VoD, content is provided just for a short while. Especially, that field requires protection of the content.

- Multiplayer games: Peer-to-Peer approaches are described as very promising for applications like Second Life [VTT-2010]. Second Life realizes a social network platform as a virtual (visual) world. Large amounts of data need to be stored and transported to provide the service. However, current client server architectures show poor scalability and become overloaded very soon. That limits the acceptability and the growth of the system. Distributed caching mechanisms that are based on P2P technology, are seen as a solution to overcome such limitations. P2P may become an enabler for revenues.

[1]A set-top box is a network device stored at the costomers site with the ability to receive virtual goods.
[2]www.youtube.com

Benefits of P2P systems

According to research results of the *Fraunhofer-Institut für Sichere Informationstechnologie* (SIT)[SIT-2010a] and an industrial partner Technicolor[TEC-2010] (a global provider of set-top boxes), this approach is expected to provide relevant benefits compared to technologies that are already established at the market. Major benefits are:

- Bandwidth Efficiency: Kuntze and Rudolph[KUN-2009a] describe, that P2P networks allow optimized bandwidth utilization. Especially, capacities of the so called “last mile” are not efficiently utilized in centralized networks. It is assumed, that the P2P network substitutes the centralized core network. Optimized bandwidth utilization is desirable for network operators. Costs can be reduced if only the necessary capacities are provided.

- Energy Efficiency: According to an evaluation in context of the NaDa project [THO-2010], the energy efficiency of a P2P like content distribution system can be much better than a comparable centralized content distribution. Some selected scenarios of interest, show energy savings from 28% to 40%.

- Disk Space and Computing Power Efficiency: Within centralized systems, the customer’s equipment is only used, to casually consume selected content. Most of the time, disk space and computing power are simply not used. P2P systems enable the equipment to actively take part in content distribution. This is beneficial to the content provider that can reduce storage and computing power at the back end.

- Scalability and Complexity: Scalability is seen as one of the most critical drawbacks of centralized systems [EUC-2009]. Peer-to-peer systems are naturally scalable. The degree of scalability depends on the employed techniques of a concrete system. Regarding this aspect, modern P2P systems exceed centralized systems. The operation of badly scaling systems is generally seen as costly. Especially the expansion of a system becomes increasingly expensive. The same applies to the complexity of systems [KUN-2009]. P2P systems show low complexity compared to centralized systems.

- Availability and Connectivity: Availability describes the proportion of time a system is in a functioning condition. This can be expressed as the ability to establish a connection (Connectivity) as well. Because of the distributed character of P2P, the critical components of a system are duplicated many times (Redundancy). There is no single point of failure anymore. Hence, the availability is provided by the whole network instead of a connection to a central system. It certainly depends on the implemented techniques and the type of P2P system, as well. Availability is always a necessary measure for commercial applications. If services are less available, less can be sold.

1.4 Related Work

NaDa Project

The thesis is part of the NaDa [SIT-2010a] research project that is funded by the European Union (FP7 Project Media Distribution)[EUC-2009] and coordinated by Thomson [TEC-2010]. Industrial partners and research institutes work together, in order to research and develop a next generation content distribution network. Numerous contributions have already been created. They are basis and environment for the results of this thesis. The Fraunhofer Institute SIT contributes IT-Security concepts, designs and solutions to the project. The most important results are:

- Kuntze et al. published a scientific paper, called “Trust in Peer-to-Peer content distribution protocols”[KUN-2009]. It describes the integration of hardware based trust concepts from the domain of Trusted Computing in the BitTorrent protocol suite. An extension to BitTorrent protocols is suggested. The solution allows detection and suppression of malicious or manipulated nodes from the network. The resource consumption is considered in the design as well.
- Leicher et al. contributed an “Implementation of a Trusted Ticket System”[LEI-2009]. It describes an approach to establish trust between multiple parties. The approach is reused in the protocols that are implemented within this thesis.
- Kuntze and Rudolph studied “Trust in Distributed small sized Data Centers” [KUN-2009a]. They propose a security architecture for commercial P2P applications, where devices are not under the physical control of the owner. The solution allows the protection of resources (content) and software. Manipulated nodes can be excluded and certain applications can be isolated. The isolation enables several content distribution services on one platform. With this thesis, parts of the suggested architecture are implemented.
- Kuntze et al. present an architecture, including security functionalities, in “Trust in the P2P Distribution of Virtual Goods” [KUN-2009]. It focuses on the business area of virtual goods distribution. They identify, that the efficiency, in terms of latency, is crucial to the applicability of the architecture. Some of the security functionalities are realized within this thesis, as well as, measurements of the performance.

Within the ongoing thesis “Entwicklung eines Konzepts für ein sicheres Peer-to-Peer System mit dezentralem Management”, by Lincke [LIN-2010], results of the current work are reused. The thesis mainly addresses the management of a trustworthy P2P system. Where this is currently realized as a central entity, a decentralized approach is expected to provide many desirable advantages. Furthermore, advanced management functionality shall be defined.

Other research activities

P2P & TC. Some effort has already been made on the research of P2P networks that are based on trusted computing. A related approach is described by Balfe et al. [BAL-2005]. They suggest applying features from the TCG specifications to enhance security of P2P networks. Ideas are discussed in the context of commercial P2P networks, too. Though many aspects of the current approach are already discussed, a concrete P2P network is not described. The proposed extensions apply to the protocols SSL/TLS[3] and IPSec[4]. Measures of such an application or a commercially applied solution are not known, yet.

Other Approaches. The security concerns for the business fields, relevant in NaDa, are discussed in several scientific papers. But the solutions are not implemented on infrastructural level. They can be classified in reputation- and detection based approaches.

- A reputation based approach is described by Ayyasamy and Sivanandam in “Trust Based content Distribution for Peer-to-Peer Overlay Networks” [AYY-2010]. The scientists identify adversary nodes as a major risk in P2P content distribution systems. They suggest introducing a trust index for each participant of the network. Based on a bad value of the index, single participants may be excluded from communication.

- A detection based approach is described by Sherman et al. in “Adding Trust to P2P Distribution of Paid Content” [SHE-2009]. They suggest special network participants that police the system. Characteristics of malicious peers are defined and measures. Suspicious peers are detected and excluded from the network.

Other Business Applications. Similar approaches are already described for other business applications.

- For business models in the area of *P2P Voice over IP* (VoIP) and *decentralized dissemination control* (DCON), Shandu and Zhang [SAN-2005] described an approach to control the access and dissemination of data objects. The approach is based on trusted computing technology like the work described in this, but different business models are addressed.

- The P2P Next project [VTT-2010] is an international consortium of academic and industrial players, who aim to create a P2P content delivery platform. It is focused on the distribution of real-time streaming media over the internet, especially broadcasting. That focus is different from the focus of the NaDa project, since broadcasting of e.g. television channels does not need the same security level. NaDa is based on the assumption that the distribution of resources needs to be strongly protected.

[3]*Secure Socket Layer* (SSL) / *Transport Layer Security* (TLS)
[4]*Internet Protocol Security* (IPSec)

1.5 Outline

The thesis is organized in six chapters. This chapter provides a problem description, motivation and goals of the thesis. An introduction is given to related work in this field of research. Furthermore, the benefits of a commercial application are summarized. Chapter two contains the foundation for the subsequent work. Fundamental concepts of P2P communication, cryptographic techniques and trusted computing are introduced. In chapter three the development of an enhanced application is explained in detail. It begins with an overview of the application environment and continues with a detailed comparison of the original and the enhanced protocols. From a functional perspective, all implemented tasks of the communication parties are described, according to the protocol flow. The enhanced communication framework is presented and also the integrated cryptographic engines. In chapter four, the performance evaluation is described. Initially, an overview of the experiment process is given. The measurement configuration and technique are described in detail, as well as the conducted experiments. Results are discussed and its conclusions lead to chapter performance optimization. It includes implemented and suggested modification that, are expected to increase the processing time. A description of the focus and the approach is provided. The thesis concludes in chapter six.

The layout of the thesis follows the guidelines and recommendations from the department of computer science of the University of Applied Sciences Darmstadt. As far as applicable, UML 2.0 [OMG-2010] is applied as a common language for the description of software systems.

2 Foundations

This chapter provides an overview of technologies and theories on which the following work is founded. Fundamental terms and concepts are introduced for the area of trustworthiness and *Peer-to-Peer* (P2P) communication. The assurance of trustworthiness is based on concepts of *Trusted Computing* (TC) and the underlying cryptographic techniques. An overview of P2P technology should allow a deeper understanding of the subsequent work.

2.1 Peer-to-Peer Communication

With the Internet a worldwide communication network has evolved over the last decades. Information is exchanged between its participants. Services are provided and accessed every day. Since the early days of the *Advanced Research Projects Agency Network* (ARPANET), the dominant paradigm for distributed systems (e.g. based on the file transfer protocol) was the *client/server* model. It realizes a distributed system, wherein central *server* (S) (e.g. Web server) offers resources to many *clients* (C). Nowadays, the main application is the *World Wide Web* (WWW) with web servers and client browsers. In this model, the network components are arranged according to a star topology (figure (a) 1). The central server distributes resources and it is responsible for discovering the resource location. Resources are typically located at the server or its backend system. When contents are consumed, the resources do not have to be preserved on the client. A server does not request resources from a client in order to deliver it to another one. Furthermore, a client is not intended to obtain resources from other clients.

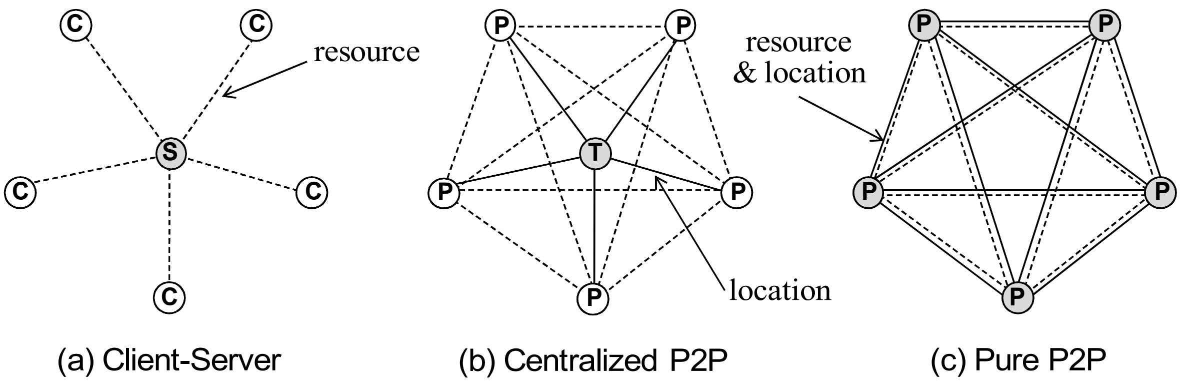


Figure 1: Classification of Peer-to-Peer Systems.

Nowadays, the contrary P2P paradigm is applied in many areas. Network nodes in such a system are termed *peers* (P). They are equally combining server and client functionality. Each peer typically communicates directly with many other peers. Any peer can provide resources to any other peer. Resources may include bandwidth, storage space, computing power, digital content or others. This thesis focuses on digital content (e.g. multimedia).

Any peer can exchange (share and download) resources with any other peer. In that way, resources are replicated and stored on peers instead on a central server. Very often the term *overlay* network is associated with Peer-to-Peer. It describes network topologies that are formed above other (underlying) network topologies. From a layered network perspective, all of the Peer-to-Peer networks are present on the application layer. The Internet up to the *Internet Protocol* (IP) layer may be seen as its underlying network.

Terminology. Peer-to-Peer *is a paradigm for communication on the Internet that is avoiding central services. A Peer-to-Peer system is a self-organizing system of equal, autonomous entities called* peers.

In view of the applied paradigm for resource discovery, Peer-to-Peer systems may be classified in pure and centralized Peer-to-Peer systems. This thesis focuses on the centralized approach with a centralized resource discovery. An example is shown in figure (b) 1 on the previous page.

Centralized systems represent a mixture of both paradigms. The Peer-to-Peer paradigm is applied for the exchange of resources and the client-server paradigm is applied for resource discovery. One or many central servers, termed *tracker* (T), keep track of the resource location and provide contact information of all participating peers. All peers are equal and autonomous. In terms of resource exchange, the system is self organizing. A peer may join or leave the network in own discretion. This approach is already applied for commercial systems. Examples are Skype[5] in the area of *Voice over IP* (VoIP) or Zattoo[6][CHA-2009] in the area of *television* (TV) broadcasting. In the area of file sharing, the BitTorrent[7] system can be operated in that way, too. But commercial applications, especially in the area of file sharing, are rare.

Peers in pure Peer-to-Peer systems are responsible for the resource exchange and also for the resource discovery (figure (c) 1 on the preceding page). The discovery is realized by a self-organizing and autonomous network. No central service is applied anymore. All peers are equal regarding resource exchange and resource discovery.

2.1.1 Segmented Resources

Peer-to-Peer systems are usually applied to distribute large sized resources (e.g. video file). But it is not applicable to share the whole resource at once between two peers. Since peers act autonomous, a sharing peer may leave the network before the exchange of the resource is

[5]www.skype.com
[6]www.zattoo.com
[7]www.bittorrent.org, www.bittorrent.com

completed. Then the remaining peer would have peer to start the exchange again with another peer. A typical solution is the segmentation of a resource. A peer that is interested in a resource may receive different segments (pieces, chunks, etc.) of it from many peers at once. Another major advantage is that peers can start to share the resource even if they haven't obtained the entire copy, yet.

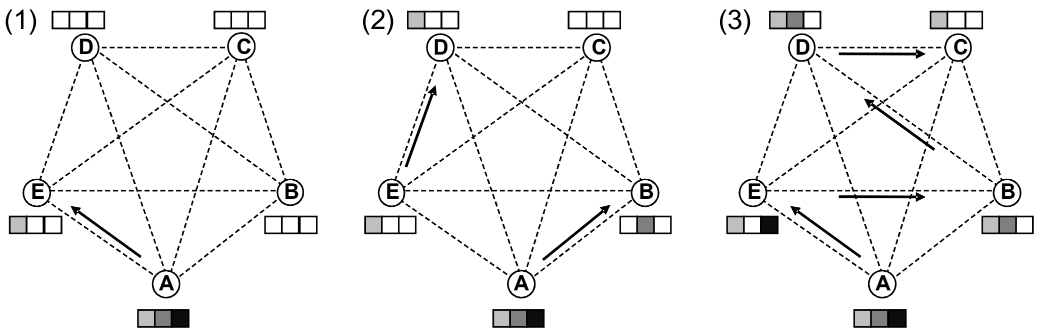


Figure 2: Resource Segmentation.

An example is shown in figure 2. It is assumed that each peer has a connection to all other peers that are interested in the same resource. Only peer A has a complete copy of the resource which is divided in three segments. Each peer may have the same upload and download capacity of one segment per time unit. (1) Initially only peer A can provide a segment to another peer (E). (2) Thereafter, peer A and E can provide a segment to two other peers (D,B) at the same time. (3) With the next step, and until all have obtained the complete resource, four segments can be provided at a time.

2.1.2 Networks

The introduced peer-to-peer systems (pure and centralized) realize different overlay networks for resource exchange and the resource discovery. In the previously depicted topologies, all peers are connected to all other peers. But such topologies are rare in practice, since they fail on a large scale. Alternative mechanisms for the construction of the overlay network(s) have to be applied. They are meant to meet an acceptable degree of relevant requirements. An example is redundancy to enforce network cohesion. Within this thesis it is strongly distinguished between resource exchange and resource discovery network. This is due to the fact that the subsequent work focuses only on a centralized resource discovery in combination with a (simplified) managed resource exchange and storage. A content provider is intended to decide who may exchange and store content. Advanced concepts are subject to current research activities, see related work of Lincke and Kuntze in secion 1.4.

2.1.3 Resource Exchange Network

The resource exchange network, or delivery network from a commercial perspective, is typically very restricted by the interests of each peer. A different network will emerge for each resource, since not all peers are interested in the same resource (see figure 3). Within the network each peer has contact information of all other peers that are sharing the same resource. The contact information is provided by entities of the resource discovery network (e.g. central server).

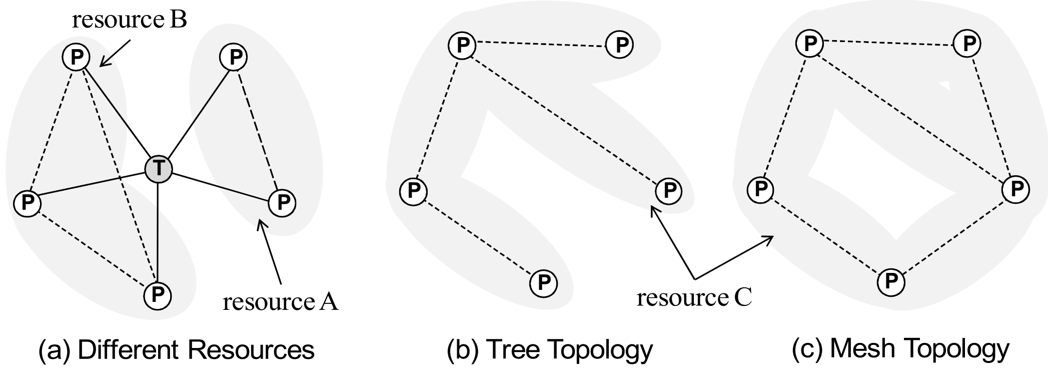


Figure 3: Resource exchange networks.

An optimal topology of an exchange network depends on many requirements as resource utilization, degree of stability, degree of connectivity, delivery performance, etc. The approaches can be generally divided in two classes: *tree-based* (figure 3b) and *mesh-based* (figure 3c). A comparative study, from the area of P2P live streaming, is provided by Magharei and Rejaie [MAG-2007]. In tree-based approaches, joining and leaving of nodes may have consequences to the whole tree of peer connections. A single peer can't contact other peers at will. This is done by a global tree construction algorithm. In some areas push mechanisms are applied. They are flooding the network of a certain resource. That is similar to a multicast or a broadcast.

In mesh-based approaches, peers may join or leave the network (churn), and create or cancel connections randomly. There are no explicit mechanisms to construct the topology. But still some mechanisms are needed to guarantee for a resource exchange network of high performance [LEG-2006].

The so called *choke* algorithm is introduced to exclude or limit the network connection of a peer. Decisions are based on the uploading rate of each peer. It becomes necessary, since an autonomous acting peer might refuse to share resources (free-riding). The behavior can be based on a rational decision. If the network capacity is used for download only, an increasing download rate could be expected. That would be advantageous for the peer. The game theory, a branch of applied mathematics, describes optimal strategies for such situations. They can be applied in P2P networks as well. An example is the tit-for-tat strategy. It assumes that cooperating peers will achieve a better result in the long run. In BitTorrent, that is implemented as *optimistic unchoke* which is giving a worse rated peer a chance to cooperate (share).

Another mechanism, termed *rarest-first*, supports the availability of resources. In Peer-to-Peer networks the resource is distributed. Some segments of a resource might be stored on more peers than other segments. In some cases a peer is not able to receive a complete resource if segments are not available at the connected peers. Hence, peers are interested to obtain the rarest segments at first. In the view of the whole network, this strategy increases the overall availability of resources.

2.1.4 Resource Discovery Network

Peer-to-Peer systems are characterized by a redundant distribution of resources to peers. Unlike the client-server model, there is no central storage anymore. As a consequence, resource discovery mechanisms have to be implemented. They have to cope with highly dynamic network entities that frequently join or leave the network and frequently store or erase content.

Many approaches have been implemented in Peer-to-Peer systems during the last decade. They are founded on a variety of scientifically contributions from the area of graph theory and, very often, practical experiences. A detailed overview can be found in [STE-2005] and [MAH-2007]. Typical strategies of resource retrieval are a central server, flooding search or distributed indexing. Because of the variety of emerged approaches, a detailed introduction can't be provided within this thesis. Furthermore, the created solution is independent of the concrete approach. An exemplary system is used to provide a proof of concept. For that purpose, the traditional BitTorrent system with a central tracker is chosen.

Central Server

As already stated, in the traditional BitTorrent system, a central tracker is responsible for the resource discovery. It keeps track of the distribution of a resource within the network. This is ensured by a *tracker protocol*. Each peer has to inquire contact details of other peers that are sharing segments of the requested resource. That approach is similarly implemented in Napster, eDonkey, Zattoo, and others. Solutions of this class have to deal with many requests to the central server.

Pure Peer-to-Peer

In parallel, pure Peer-to-Peer systems have been developed. They either rely on arbitrary connecting peers or mechanisms to organize the structure of the discovery network. Gnutella is an example of a pure and unorganized Peer-to-Peer system. It uses flooding as resource discovery mechanism. An advantage is that there is no single point of failure. Each participant can be replaced by another. But such networks have shown poor results for the discovery of resources.

It is typically very slow and in some cases it fails completely to retrieve a resource, even if it is available.

Structured Peer-to-Peer

An organized structure of the network and the storage of content are enforced in many systems. Exemplary systems are CAN, Chord, Pastry, Tapestry, Kademlia and others. All of these are based on *distributed hash tables* (DHT). They differ in structure, routing algorithms and performance. Most of these approaches are not limited to P2P systems. They are successfully applied in the area of large scale storage management systems, too [RAT-2001].

DHTs decouple the naming scheme from the name resolution process. A contrary example is the *Domain Name System* (DNS). Its naming scheme addresses keys (DNS addresses) to values (IP-Addresses) by a hierarchical tree structure. The root of the structure is the top-level domain, followed by the 2nd level domain and so on. Values (IP-Addresses) are resolved according to this structure by dedicated DNS servers. Disadvantages of the DNS approach are e.g. the dependency on the roots, potential single point of failures and no real load balancing. In DHTs, keys are opaque. They are created using a hash function that is mapping each data object (or its reference e.g. name of a file) to a unique key in a defined key space (e.g. 160-bit in SHA-1). A node ID, out of the key space, is assigned to each participant (peer) of the network according to defined rules. Herewith, the node becomes responsible for an address space of referenced data that is close to its own ID. The notion of closeness varies amongst the approaches. Well known are definitions like: the ring space in Chord[STO-2001], cartesian n-dimensional space in CAN[RAT-2001] and the XOR metric of Kademlia[MAY-2002]. Each node stores $(key, value)$ pairs and provides the basic operations $store(key, value)$ and $retrieve(key) = value$.

2.1.5 BitTorrent

BitTorrent defines a distributed system that is enabling P2P communication, especially for the purpose of file sharing. It was initially specified by Bram Cohen [COH-2008] in 2001. The communication between peers is defined by two protocols (see section 3.2). The *Peer-Wire Protocol* (PWP) describes the communication between peers and the *Tracker Protocol* (TP) between peers and trackers. Initially BitTorrent was based on central trackers, but some of its later extensions allow a DHT based resource discovery (e.g. Kadmelia). BitTorrent establishes its own terminology. Relevant terms in context of this thesis are:

- *Seeder*: A peer that has a complete replica of a resource.
- *Leecher*: A peer that downloads, but refuse to share a resource.

- *Downloader*: A peer that is downloading and sharing, but hasn't got a complete replica of a resource, yet.
- *Swarm*: A collective group of peers that are sharing the same resource.
- *Metainfo file*: A metafile (typically named .torrent) that contains meta information of a resource. The file is sometimes called Torrent, too. Throughout the thesis it termed *metafile*.
- *Tracker*: A network node that keeps track of a swarm of peers which are sharing the same resource.
- *Publisher*: The initial seeder that is creating and publishing (to specified tracker) the metainfo file.

BitTorrent is broadly used in conjunction with additional web servers. They store metafiles and provide them to anyone. Peers may download them and contact the trackers contained in the metafile.

Metafile

The metafile (also metainfo file or torrent) is a unique reference to a certain resource (e.g. multimedia file), and it refers to at least one tracker. That file can be freely exchanged in any available way. It is created by a peer that owns the initial resource replica. By publishing it to a tracker, the peer offers to share it with other peers. Therefore, the tracker stores the metainfo file and peer contact information. Contact details are provided to other interested peers. Main attributes of the metainfo file are described below. The file is encoded in a BitTorrent specific, platform independent encoding format, called *Bencoding*.

- `Announce(-list)`: Contains the URL of the tracker. Peers can join the swarm, using this URL.
- `Info`: Contains information about the files to download:
  - For single file torrents: `length`, `md5sum`, `name`, `piece length`, `pieces`
  - For multi file torrents: `files`, `name`, `piece length`, `pieces`
- `Creation date`,`Comment`, `Created by`

In BitTorrent networks, a resource is segmented in so called *pieces*. The publisher defines the length of a piece and therewith the total amount of pieces. This total amount is defined as the size of the resource divided by the piece length. Typically, the piece length is a power of two.

For each piece a hash value (160-bit SHA1) is computed and stored as concatenated string in field `pieces` of the metafile. That enables integrity measurement of each piece. During data exchange between peers, the pieces of data are further segmented and delivered as blocks of data.

2.2 Cryptographic Techniques

Within this section, cryptographic techniques, terminology and notations that are applied in the following chapters are introduced. Notations are in the style of the project, which provides a context to this thesis. In this way, they are applicable to all project related resources. Another reason is that in scientific literature no common notation seems to be used. The mathematical foundation of the cryptographic techniques is not discussed in detail within this thesis. Where applicable, appropriate references for further reading are provided. The same applies to cryptographic algorithms.

Cryptographic techniques have a very long history. They are based on the longing to keep and transmit information secretly. They are mostly used by encryption and decryption, digital signatures or secure key exchange. History has shown that techniques are successfully applied for a period of time until ways are discovered which allow to access or manipulate the secret information. The same is true for modern techniques.

Nowadays cryptography is characterized by a paradigm which became popular in the second half of the last century. Techniques (functions, algorithms, protocols) have been made public. The only remaining secret was the cryptographic key. This approach has turned out to be advantageous. It allows quality control by all participants and it enables standardization. Furthermore, it allows users to leave the group users that know the algorithm. The consequence for secret techniques was that a new secret technique had to be established between the remaining users.

Terminology. *A* cryptographic key *(denoted by* K*) is a string (binary for computers) of a certain length depending on the algorithm used. The length of the key determines a key space including all possible keys.*

The key is computed by cryptographic algorithms to transform information into another format ensuring that the information can't be deduced. Recovering the information within a realistic period of time should only be possible by the secret key. Mainly two differing concepts of keys have evolved, symmetric and asymmetric keys (private key and public key). However some techniques like hash functions (see 2.2.3) do not use keys at all.

Notation

Cryptographic algorithms with key involvement are depicted as $name\{parameter\}_{Key} = output$. It can be read as a function which maps parameter values and the key(-parameter) to an output value. In some cases the type of function may already be implied by the type of the applied key. The unnecessary name could be omitted. But it seems to be easier to read (and understand) in this way. For cryptographic functions in this context, the parameters can typically be reconstructed using a reverse function and a reverse key. If this is not provided, the common notation $name(parameter) = output$ is used.

2.2.1 Message Encryption & Decryption

The encryption and decryption of messages is expected to fulfill certain security goals. Most important, the plaintext is kept confidential (Confidentiality), even if it is transmitted over an open network. Only the recipient should be able to interpret the message. It must be ensured, that only the sender is able to encrypt a message and only the receiver is able to decrypt the message. Therefore, the receiver is able to ascertain the origin of the message (Authentication). On the opposite, the sender can't deny that he encrypted the message (Non-repudiation) if he is ensured that only the sender can create the respective encrypted message. That is why, modifications of the encrypted message should always be recognizable by the receiver (Integrity).

Terminology. *A message is* plaintext *(sometimes called cleartext). The process of disguising a message in such a way as to hide its substance is* encryption*. An encrypted message is* ciphertext*. The process of turning ciphertext back into plaintext is* decryption*. Encryption and decryption is performed by an algorithm called* cipher.

Cryptographic operations are depicted as follows. In modern cryptography both operations rely on cryptographic keys. In symmetric systems, the same key is utilized for encryption and decryption. Keys are called symmetric keys, here.

$$enc\{plaintext\}_K = ciphertext$$

$$dec\{ciphertext\}_K = plaintext$$

In asymmetric systems different keys are applied for each operation. These are public keys K_{pub}, that can be accessed by everyone and private keys K_{priv}, that are in possession of only one owner. Even more, both keys can be applied for both operations.

$$enc\{plaintext\}_{K_{pub}} = cyphertext$$

$$dec\{cyphertext\}_{K_{priv}} = plaintext$$

$$enc\{plaintext\}_{K_{priv}} = cyphertext$$

$$dec\{cyphertext\}_{K_{pub}} = plaintext$$

A message that is encrypted with a private key can be read by everyone using the corresponding public key whereas a message that is encrypted by a public key can only be read by the owner of the corresponding private key.

Cipher

Modern ciphers perform mathematical operations (e.g. substitution, transposition) on a fixed length input. Since the length of the input string is not fixed in practice, these strings are divided in a stream of single characters or blocks of characters. Accordingly, ciphers can be classified in block and stream ciphers. Ciphers within the context of this thesis apply cryptographic keys as additional input. Bock and stream ciphers are mainly applied in symmetric key cryptography. They provide different characteristics, hence they are applied in different areas in practice. Commonly used ciphers are the *Advanced Encryption Standard* (AES) and its predecessor *Data Encryption Standard* (DES). AES is also known as Rijndael-algorithm which was developed by Joan Deamen and Vincent Rijmen.

Block ciphers operate in different modes [ECK-2008, Chapter 7.5.2]. Typical modes are *Electronic Code Book* (ECB), *Cipher Block Chaining* (CBC), *Cipher Feedback* (CFB) and *Output Feedback* (OFB). These modes differ mainly in the level of security (vulnerability) and their performance. Some of them require a commitment of both communication parties to an additional shared and secret initialization vector. Often the input string is not a multiple of the block size. In that case, some characters are missing for the encryption of the last block. This is compensated by padding schemes that attach defined or calculated values to the input string.

Stream ciphers operate similar to one-time pads [ECK-2008, Chapter 7.4.3]. A pseudo random key-stream is used for the encryption and decryption operation. This approach is seen to be faster than block ciphers, but it bears its own vulnerabilities.

In the area of public-key cryptography, *Rivest, Shamir, and Adleman* (RSA) developed cryptographic algorithms that are nowadays established as de facto standard. RSA ciphers and many other standards are described in the *Public-Key Cryptography Standards* (PKCS) of the RSA laboratories [RSA-2010]. Some of these standards can be found in other standardizing organizations, too (e.g. RFC's from the IETF[8]). In practice, RSA is combined with padding schemes to prevent a number of attacks. Some encryption schemes are standardized in PKCS#1 [RSA-2002], these are RSAES-PKCS1-v1_5 and RSAES-OAES (Optimal Asymmetric Encryption Padding) for encryption, and RSASSA-OAEP and RSASSA-PKCS1-v1_5 for signatures. The definition of a padding scheme is included in PKCS#5 [RSA-1999].

[8]www.ietf.org

2.2.2 Digital Signature

Digital signatures are an essential component in the application of modern cryptography. They represent a digital equivalent to an ordinary handwritten signature. Digital texts like documents or messages can be signed, whereby certain goals are achieved. These are the same goals handwritten signatures are meant to achieve [SCH-1996]. The signed document would be any kind of arbitrary string containing relevant information.

- Authentic: The signature convinces the document's recipient that the signer deliberately signed the document.
- Unforgeable: The signature is a proof that the signer, and no one else, deliberately signed the document.
- Not reusable: The signature is part of the document; it can't be moved to a different document.
- Unalterable: The document can't be altered after signing.
- Non-repudiation: The signer can't later claim that he didn't sign it.

Terminology. *A* digital signature *is the output of a signing algorithm which is applied to an arbitrary string using a cryptographic key. Such a signature provides the same characteristics that are expected from handwritten signatures.*

In recent applications (and within this thesis) the creation and verification of digital signatures is based on algorithms using asymmetric key cryptography. A signature is bound to the private key that is utilized during the creation of the signature. Depending on the algorithm used, signing can be seen as an encryption of a string with the private key (e.g. RSA[SCH-1996, Ch. 2.6]). Then the verifier can compute the signature using the public key of the signer. The resulting string is equivalent to the signed string. Since the verification key is public, everyone can compute the verification.

$$signing:\ sig\{string\}_{K_{priv}} = signature$$
$$verification:\ ver\{signature\}_{K_{pub}} = string$$

With this approach authenticity is given, since it is assumed that only the signer possesses the private key. The signature is unforgeable as far as this is provided by the underlying encrypting algorithm respectively its mathematical foundation. Signing can only be done with a certain private key and for everyone it should be hard to calculate or guess that key. No one can reuse the signature since different strings will always result in different signatures. Otherwise the verification would lead to many different strings and no one would know which string was signed. The string can't be modified once it is signed (encrypted). Not even by the signer, because an altered string would lead to an altered signature. Finally, the signer can't repudiate the signing of a string by his private key since the verification can only be successfully computed with the appropriate public key.

2.2.3 Public-key Certificate

A public-key certificate (within this thesis called certificate) is typically used to bind an identity to a public key. This is done by a certifying third party, termed *Certification Authority* (CA). Certificates confirm that a key is registered for a certain identity, which has been validated according to the CA's commitments. All participating parties have to trust the CA.

Terminology. *A* public-key certificate $Cert_{issuer}^{owner}$ *is someone's public key, signed by a trusted entity (issuer). Depending on the format definition it may contain credentials referring to the public key owner and additional data.*

The certificate may be generated by an appropriate digital signature algorithm using the private key of the CA. Everyone possessing the CA's public key can obtain the key and credentials for another party of interest[9].

$$sig\{credentials, K_{pub}\}_{CA_{priv}} = Cert_{issuer}^{owner}$$
$$ver\{Cert_{issuer}^{owner}\}_{CA_{pub}}$$

Depending on the chosen certificate standard (e.g. ITU-T[10] X.509 / [RFC-5280]) respectively application design different credentials are included. Very common are: the name of the owner, declaration of algorithms used, identifier of the certificate, validity of the certificate, identifier of the CA (issuer) and restrictions to the application of the key[BUC-2008, Ch. 17.2.3]. The origin of a certificate can be verified by the public key of the issuer.

Hash functions

In cryptography hash functions are often used to verify the integrity of messages (or more generally strings). Typically a so called hash function computes an arbitrary-length message string to a smaller fixed-length string called hash. Keys are not used.

$$h(message) = hash$$

[9]In practice a certificate may be represented as an object containing credentials and key as well as a signature over the hash of these values.

[10]www.itu.int/ITU-T

The hash is sometimes referred to as a fingerprint, implying that each message has a unique fingerprint. If this is ensured by the hash function, an altered message can easily be recognized by comparing the hash of the original message with the hash of the altered message. Nowadays, commonly used hashing algorithms like those described by the *secure hash algorithm* (SHA)[SCH-1996, Ch. 18.7] represent one-way hash functions.

Terminology. *A* one-way hash function *is a hash function that works in one direction: It is easy to compute a hash value from a message, but it is hard to generate a message that hashes to a particular value. And it is hard to find another message that can be computed to the same hash (collision resistant).*

Whether a function can be classified as on-way depends on the existent knowledge and technological abilities. It may change over time and new functions have to be discovered. Hash functions may be further classified in weak collision-resistant and strong collision-resistant[BUC-2008, Ch. 12.1]. Weak, if it is not possible to find a collision for a given message in practice and strong, if it is not possible to find any collision in practice. Furthermore it is important to notice, that the output of the hash function does not depend on the input in a way that a single bit change would lead to minor (predictable) changes in the resulting hash. Ideally, on average half of the hash is changed then.

2.2.4 Protocol

Usually cryptographic techniques are applied in communication between two or more parties. In order to accomplish certain tasks (data exchange, key exchange, authentication, attestation, etc.), communication steps are defined. Series of communication steps form protocols.

Terminology. *A* Protocol *is a series of steps, involving two or more parties, designed to accomplish a task. A* cryptographic protocol *is a protocol that uses cryptography.* [SCH-1996]

Several primitive protocols may be implemented on different communication layers or they may be combined to a more complex protocol on one layer. Such a protocol is presented and implemented within this thesis. It incorporates the *Diffie-Hellman* (DH) protocol as a primitive [RFC-2631].

With the DH protocol (see protocol 1 on the next page) two parties establish a symmetric shared secret key over a public network. Therefore they exchange public parameters and values computed by a one-way function. One-way functions are relatively easy to compute, but significantly harder to reverse. An example is the discrete logarithm. There is no algorithm known, able to solve this problem efficiently [BUC-2008].

Protocol 1 Diffie-Hellman

```
1. Pre-computation:
        $Alice: \quad a, A = g^a mod\, p, parameter := \{g, p\}$
2. Send public key and parameters:
        $Alice \rightarrow Bob: \quad \{g, p\}, K_{pub} := \{A\}$        (1)
3. Compute public key:
        $Bob: \quad b, B = g^b mod\, p$
4. Compute common secret:
        $Bob: \quad K = A^b mod\, p$
5. Send public key:
        $Bob \rightarrow Alice: \quad B$        (2)
6. Compute common secret:
        $Alice: \quad K = B^a mod\, p$
```

Previous to the protocol $Alice$ chooses a random private key a within {0,1, ..., p-2}. Furthermore, Alice generates the public parameters $\{g, p\}$ where g is a primitive root modulo the prime p with $2 \leq g \leq p - 2$. Parameters and private key are applied in the creation of the public key A. This public key and the public parameters are transmitted to Bob. Now, Bob generates its own random private key b and its own public key B in the same way $Alice$ did before. Additionally, the shared symmetric key K is computed using Alice's public key A, Bobs own private key b and the public prime p. Finally, Bob transmits its public key B to Alice who computes the shared symmetric key K using the received public key B the private key a and the public parameter p. Both communication parties established a common secret that can be used to encrypt and decrypt further communication. It has to be mentioned, that this protocol does not provide authentication of the communication parties and therefore it is vulnerable to certain threats (man-in-the-middle[BUC-2008, 9.5.3]).

2.3 Trustworthiness

Trust is a very universal term, describing a degree of reliance on the (predicted) behavior of a system. If the observed behavior is as expected a system may be accepted as trustworthy. Within the area of information and communication technology systems may be seen as computing systems (equipped with a processing unit) or a group of connected systems forming a network system.

Regarding IT-Security, systems may be classified as trustworthy if they meet certain security goals, especially integrity and authenticity. Today's IT-systems are characterized by an increasing vulnerability that is attended by an increasing complexity. Within the last years, efforts have been made to design and realize systems with a higher degree of confidence in its state and the attestation of its state to remote systems. Especially, members of the industry consortium *Trusted Computing Group* (TCG)[TCG-2010] have contributed to this development. It is strongly based on cryptographic techniques, mainly from the area of asymmetric cryptography. Together it is commonly known as Trusted Computing.

Terminology. „Trusted Computing *describes the execution of security crucial software on a system, which current state was classified as* trustworthy *and which provides methods to measure the current state.*“

2.3.1 Trusted Computing System (TCS)

An overview of a *Trusted Computing System* (TCS) is given in figure 4. It shows several independent layers that are typical to modern personal computers. The TCG specifies extensions to some layers respectively components so that they become trustworthy. However, services and applications that are not extended may be operated on such a system as well. This is necessary, since it is practically not possible to extend all applications of all providers. Instead, dedicated applications have to be classified as trustworthy or untrustworthy by the operator of the system. This approach is well-established anyway.

All extended components together form a so called *Trusted Computing Base* (TCB), that is able to measure and report the state of all software applications on the system. The specification of a *Trusted Computing Platform* (TCP) describes new and amended hardware components as well as firmware (e.g. *Basic Input/Output System* (BIOS)). Most important is an additional hardware chip called *Trusted Platform Module* (TPM). An extended operating system, called *Trusted Operating System* (TOS), may operate on the basis of the TCP. Furthermore, the TOS is offering TPM functionality to the service layer which is offering the functionality to applications. This is called a *Trusted Software Stack* (TSS).

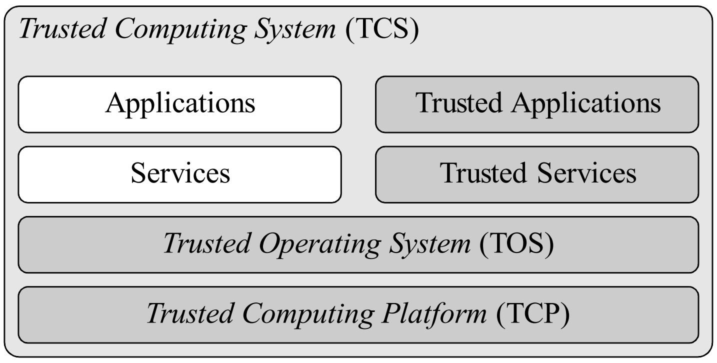

Trusted Computing System (TCS)
Applications
Trusted Applications
Services
Trusted Services
Trusted Operating System (TOS)
Trusted Computing Platform (TCP)

Figure 4: *Trusted Computing System* (TCS).

Chain of trust

All trusted components together establish a chain of trust in which each component measures the integrity of the next component from the start of the system. Measurement is performed by the root of trust or a program that is previously measured. Therefore hash values of platform properties like the ROM of a hardware component, an executable or operating system libraries are generated. Such a measurement is rather passive since it just creates and stores hash values. Whether a trustworthy system can really be trusted or not, has to be decided by a third party. Therefore the TCS offers reporting functionality. Amongst others a typical chain of trust includes the TPM as a root of trust, a *Core Root of Trust Measurement* (CRTM) as part of the BIOS, the *Master Boot Record* (MBR), an operating system loader, an operating system kernel, device drivers and services.

Roots of trust

The TPM is specified to provide three *roots of trust* forming the basis for all subsequent trust decisions. This is referred to as *Trusted Building Block* (TBB) as well. Such roots of trust are already well known in the area of IT-Security. An example is the *Root-Certification Autority* (Root-CA) of a *Public Key Infrastructure* (PKI). If a root can't be trusted anymore, the same applies for all subsequent decisions.

- Measurement: The *root of trust for measurement* (RTM) is a trusted implementation of a hash algorithm for the initial integrity measurement.
- Storage: The *root of trust for storage* (RTS) is a trusted implementation of a shielded non-volatile location to hold the *Storage Root Key* (SRK).
- Reporting: The *root of trust for reporting* (RTR) is a trusted implementation of a shielded non-volatile location to hold the unique platform identity key, called *Endorsement Key* (EK). That key provides the cryptographic identity to the TPM.

It has to be noted, that there is still one more root to be trusted: the manufacturing process. That process is highly critical since it includes implementation of the EK, which is the root for all subsequent decisions. Another certifying trusted party is intended to guarantee with an EK-certificate for a trustworthy creation of the EK. According to the current specification (version 1.2), the TPM itself does not provide any functionality to perform a manipulation (create, delete, modify) of an EK. All other keys are created subsequently. Even the SRK is created later on by an invocation (`Take-Ownership`) of the TPM owner. That however, may be repeated with a change of ownership that is protected by an owner password.

Capabilities

The TCS provides additional capabilities compared to ordinary systems (e.g. personal computer). It is able to measure, store and report the integrity of all software components in a trustworthy manner. Further capabilities like cryptographic operations that can't be influenced by any software component on the system are available. These operations can be used to provide a secured storage on the hard disk. The access to storages can additionally be bound to a certain state of the system. An attestation mechanism is introduced to provide evidence of the current state to a third party. Since the EK is unique and protected, the TPM and subsequently the whole TCS can clearly be authenticated. Processes that are running on the system can be protected from other processes that might be not trustworthy. Moreover, trustworthy devices (e.g. I/O devices) can be introduced. They establish a protected communication to the TCS.

2.3.2 Trusted Platform Module (TPM)

The aim of trusted computing is to establish trusted computing systems and consequently a trusted infrastructure based on trusted systems. An important part of a trusted system is the *trusted platform module* (TPM) which may be implemented as a software (e.g. [STR-2010]) or hardware component. A specification of such a module is provided by the TCG [TCG-2005]. Depending on the type of system (e.g. mobile phone, personal computer, embedded systems, ...) and its specific requirements, different modules may be applied. An example is the *mobile trusted module* (MTM) specified by the TCG. Compared to *personal computer* (PC), these mobile phones are characterized by much stronger trust requirements, more complex patterns of ownership and more often subject to casual theft [MAR-2008].

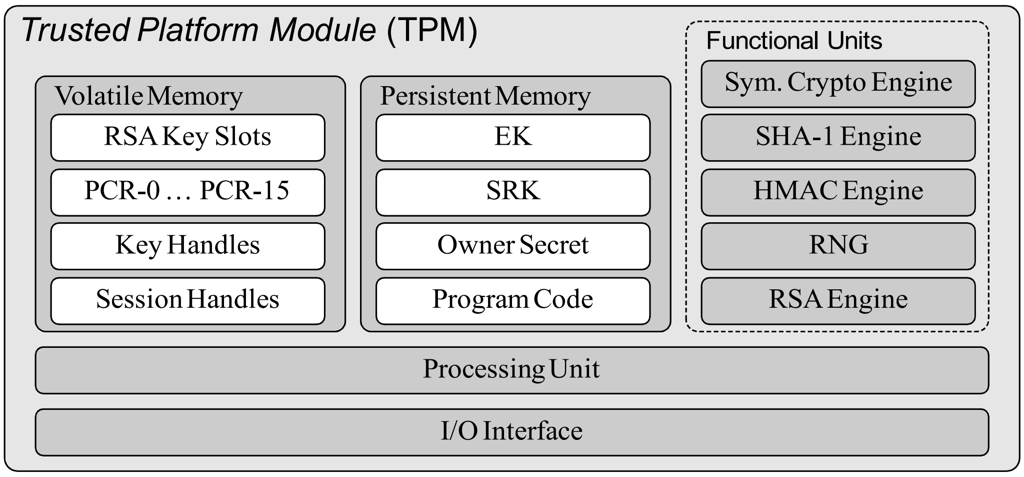


Figure 5: Overview of the main TPM components.

An overview of a TPM is given in figure 5. It is part of the TCP and it contains all parts of a typical microcontroller. The *Input/Output* (I/O) interface is connected to the *Low Pin Count*

(LPC) bus of the motherboard together with other components like the boot *Read Only Memory* (ROM) that is storing the BIOS. A processing unit handles requests and accesses services from the functional units. Values can be stored on a persistent and a volatile memory.

Functional units respectively engines provide cryptographic operations (see section 2.2). These are engines for symmetric and asymmetric key cryptography, hashes, *keyed-hash message authentication codes* (H-MAC) and a *random number generator* (RNG).

The volatile memory contains session handles that are applied in communication with external components. Key handles and RSA key slots are used to provide a trustworthy implementation of key storage and loading from an external storage space (e.g. hard disk). An amount of *Platform Configuration Registers* (PCR) store concatenated hash values (see 2.3.4) of all executed programs.

The persistent memory contains the previously introduced asymmetric keys: EK and SRK (both RSA, 2048 Bit). If a new owner of the TPM performs a `Take-Ownership` operation a password is inquired. It is stored as owner's secret on this memory. The program code enables the operation and interaction of the TPM. In the current specification (version 1.2) of the TCG [TCG-2005], ninety-one mandatory, platform independent functions of the TPM are defined. Examples are: the previously operation to take the ownership or the `TPM_Quote` operation which necessary for attestation of the TCS's state to a remote party (see 2.3.4).

Attestation Identity Key (AIK)

An additional important asymmetric key (RSA, 2048 Bit), called *Attestation Identity Key* (AIK), might be stored in the persistent memory, too. The AIKs are applied as pseudonyms for the EK during attestation as described in section 2.3.4. In practice however, more than one AIK can be expected. Then they are stored on an external device that is protected by the SRK. Typically, AIKs are certified by a trusted third party. An AIK certification protocol is described in [MUE-2008, P. 59]. A TCS creates a new AIK and provides its public part together with an EK certificate to the certification authority. If the authority is able to verify the EK certificate and the TCS identity, it certifies the AIK.

2.3.3 State Measurement

In order to understand the attestation, at first the measurement has to be understood in more detail. Based on the chain of trust, it is ensured that each component is measuring the next component to be executed. A component within this context can be any block of data that is known to take part in execution. This might be a boot-loader image, the operating system kernel, device drivers, executables, libraries and others. For each relevant block of data a hash value (SHA-1, 160 Bit) is computed and given to the TPM by an invocation of the `TPM_Extend` function. Additionally, each hash value and a corresponding, identifying string (e.g. file name of an executable) are stored in the *Stored Measurement Log* (SML). That SML (see table 1) is not part of the TPM. It is stored on another device like a hard disk. Typically the SML is split in several files for e.g. BIOS, operating system and application measurements. Furthermore, it does not have to be protected since it just represents the claimed state of the TCS. Evidence has to be provided by the TPM.

PCR Index	Measurement Value (SHA-1)	Description
10	9797edf8d0eed36b1cf92547816051c8af4e45ee	boot_aggregate
10	ea8239dfed9dd11bd538f9c3234e0d7b71672fff	/bin/sh
10	ebb4f3db0b83c1e717e3d05f702e4608a9c2ea08	/lib/ld-linux.
10	671aba5cb6df951463e57c963e8c327fc6cfb5ab	/lib/libcrypt.
10	445babe91e586090cb7f9782b44ba115ceec6b7f	/lib/libm.so.
10	411ab19e995e06b1d7378e1daea9926ccba1ea20	/lib/libc.so.
10	68b297cd8fe07a3e54e6ce9d2e66afa799e472a3	/sbin/depmod
10	407285ba377ea035f2ff6d61c47ead8befa7cd5f	/sbin/modprobe

Table 1: Excerpt of a SML from an IMA system [SAI-2006].

As already explained, simultaneously to an extension of the SML, the measured hash value is given to the TPM. However, the TPM does not have to store all the provided hashes to provide evidence of the state. Instead each added hash value is concatenated with a hash value that was previously stored in an appropriate *Platform Configuration Register* (PCR).

$$newPCR_i = hash(oldPCR_i || addHashValue)$$

According to the current specification (version 1.2) of the TCG each TPM should at least provide 23 PCRs for different purposes. Mainly, they group the measured software components in classes in order to use a dedicated register. Accordingly an attestation can be performed for a selected group of components. An example of the PCR usage is given in table 2.

2.3.4 Attestation

A TCS is able to measure and report its state. A state can be reported locally to users and devices, or it can be reported to a remote system. Together with a report (SML), the attesting TCS provides evidence that it is really in the claimed state. This is called attestation. Based on this information an appraiser decides if the evidence can be trusted or not. Such an appraiser does not necessarily have to be a TCS itself. Any system that is capable of interpreting the attester's evidence, may decide about it. An appraiser has to verify (i) the root of trust, (ii) the report (SML) and (iii) the evidence.

Terminology. "Attestation *is the activity of making a claim to an* appraiser *about the properties of a target by supplying evidence which supports that claim. An* attester *is a party*

PCR Index	Usage
0	CRTM, BIOS and Host Platform Extensions
1	Host Platform Configuration
2	Option ROM Code (e.g. firmware)
3	Option ROM Configuration and Data
4	*Initial Program Load* (IPL) Code (usually the MBR)
5	IPL Configuration and Data (for use by the IPL Code)
6	State Transition and Wake Events
7	Host Platform Manufacturer Control
8-15	Defined for the use by the Static Operating System
16	Debug
17-23	Defined for the use by the Dynamic Operating System

Table 2: Usage of the *Platform Configuration Registers* (PCR).

performing this activity. An appraiser's decision-making process based on attested information is appraisal"[COK-2010]

Remote Attestation

The remote attestation between two TCSs can be performed over an open network using a public key infrastructure. Therefore a remote attestation protocol that ensures the integrity of the evidence can be applied. A simplified version is depicted in protocol 2 below. It is performed between the attesting TCS tcs and the appraiser app. Similar protocols and vulnerabilities (e.g. man-in-the-middle) are described and discussed in [MUE-2008].

Protocol 2 Remote Attestation Protocol

1. Pre-computation:

$$app: \quad nonce, CA_{pub}$$
$$tcs: \quad AIKCert_{ca}^{tcs}$$

2. Challenge Request:

$$app \rightarrow tcs: \quad nonce \qquad (1)$$

3. Compute Quote / load SML:

$$tcs: \quad SML, Quote := quote\{nonce, PCR_{0..n}\}_{AIK_{priv}}$$

4. Challenge Response:

$$tcs \rightarrow app: \quad Quote, AIKCert_{ca}^{tcs}, SML \qquad (2)$$

5. Verification:

$$app: \quad \text{(i)}ver\{AIKCert_{ca}^{tcs}\}_{CA_{pub}}, \text{(iii) } \textit{verify SML}$$
$$\text{(ii)}ver\{Quote\}_{AIK_{pub}}, \textit{verify SML\&PCR}$$

Initially the appraiser has to create a *cryptographic number-used-once* (nonce) that is a random value, valid for only one attestation execution. With message (1), called challenge request, the nonce is transmitted to the TCS. Now, the TCS invokes the `TPM_Quote` function of the TPM with the received nonce as parameter. The TPM signs the nonce together with a selection of PCRs using its private AIK key. With message (2), called challenge response, the TCS transmits the quote object, the corresponding SML and its AIK certificate. Now, the appraiser is able to verify:

- The root of trust: Verification of the certificate by a public key of CA.
- The evidence: Verification of the PCR within the quote (signature) by the public AIK key. Furthermore by calculating an expected PCR and comparing it to the received PCR.
- The reported state: SML values are compared to a previously defined list of valid components.

The calculation of an expected PCR is performed in the same way that is performed by the TPM (see section 2.3.3). Each entry of the SML is concatenated and hashed with the next value.

2.4 Conclusion

This chapter provides profound knowledge of basic terms and concepts that are applied in the following chapters. They originate from the domain of trusted computing, Peer-to-Peer technology and cryptography. Most important are the general architecture of trusted computing systems and the trusted computing platform, state measurement and state attestation. Following chapters focus on a centralized Peer-to-Peer system with a distributed resource exchange network and a centralized resource discovery network. NaDa management functionality and distributed approaches for the resource discovery network are not subject to this thesis (see section Related Work 1.4).

3 Application Development

The development of a demonstrator is the next step towards commercial application of a P2P system that is based on the principles of Trusted Computing. The demonstrator is neither a simulator nor a prototype. Contrary to simulation, the demonstrator will be deployed in a physical laboratory close to a real commercial environment. It is not intended (and possible within this thesis) to provide all the necessary functionality and reliability that would be expected from a prototype. Mainly, the demonstrator should serve as a basis for empirical studies. The results shall give direction to further development and research efforts.

Two already proposed protocols [KUN-2010] serve as high level specification for the demonstrator. This chapter provides the description of the realization, including design and implementation. Several components like an open source BitTorrent application are integrated, amended and extended. As far as possible, the original BitTorrent protocols are kept unchanged, in order to preserve the approved processing. Unaltered BitTorrent specific functionality is not examined in detail. Extensions are described in detail on protocol, communication framework and message level. Furthermore, an overview of the integrated cryptographic engines is provided.

3.1 NaDa Environment

The implemented protocols are embedded in the NaDa [KUN-2009] architecture for distribution of virtual goods (e.g. multimedia contents). The architecture describes several components of all parties that are participating in the distribution of content. At the customer's household the physical NaDa platform is installed. It realizes communication services to an *Internet Service Provider* (ISP), one or more *Content Providers* (CP) and platforms of other customer's. Customers may access content from this platform via a personal computer, a set top box or similar devices. Even though such a platform is physically located at the customer's household, it remains under control of the ISP. For this reason, it is equipped with a TPM. The architecture enables such a platform to deliver content from more than one CP at once. Specific applications for each participating CP can be installed. Separation is achieved by a virtualized and protected context for each CP.

From each CP context a communication component, that is installed and controlled by the ISP, can be utilized to download and share content. That component is mainly based on Peer-to-Peer technology, enhanced with the trustworthy *Tracker Protocol* (TP) and the *Peer-Wire Protocol* (PWP). The protocols are implemented as extension to the *Java BitTorrent* (jBitTorrent) [DUB-2007] application. jBitTorrent provides tracker and peer services. On each platform an instance of the peer application is running as part of the communication component. It may interact with peers of other customer platforms or trackers. Both, the ISP and each of the CPs run and maintain trackers. The ISP tracker is keeping track of ISP level software updates and trackers of CPs are keeping track of the distribution of content. A dynamic view (sequence diagram) for the download use case is shown in figure 15 on page 68. Within this model, central trackers are assumed. Unique identities are provided to the system by a certification authority. Responsibilities are assigned to the ISP, though another trusted party might be involved in practice. The NaDa architecture describes further components for management, storage and monitoring. However, they are currently not involved in Peer-to-Peer communication and therefore, they are not in focus of this thesis.

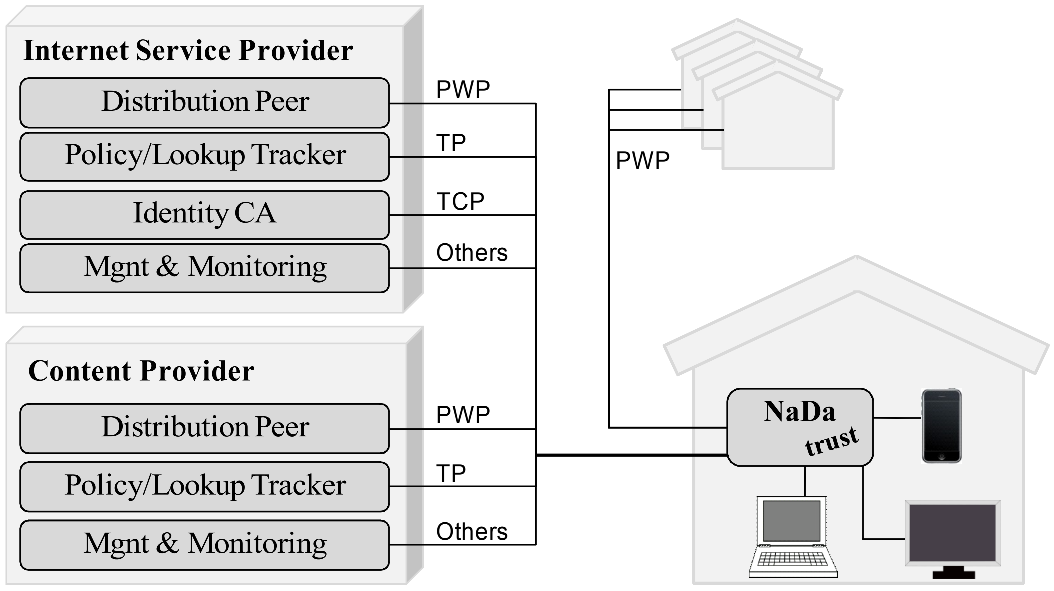


Figure 6: High-level view of *Nano Data Centers* (NaDa).

3.1.1 Component Interfaces

Trustworthy BitTorrent

The *trustworthy BitTorrent* (tBitTorrent) protocols are implemented as an extension to the open source jBitTorrent [DUB-2007]. It includes several applications that are written in Java version 6. BitTorrent functionality is provided by a group of applications. Graphical interfaces are not available, yet. According to the configuration, detailed log files for each application are stored on each node. They can be evaluated for management purposes.

Tracker. Within the current model, trackers are operated by CPs and the ISP, only. The tBitTorrent system provides functionality of the original jBitTorrent, but achieves additional security goals.

- Start: Start of the tracker application and listening on designated port.
- Upload: Upload (store) a metafile to the tracker/creation of an empty swarm.
- Announce: Peers can register to a swarm, receive other addresses peers and authorization tokens (ticket) for members of the swarm.

Peer. Peer components are utilized by all participants (ISP, CP and customer). Three applications provide an interface to other systems.

- Create Metafile: Metafiles can be created according to the BitTorrent specification (see section 2.1.5).
- Publish Metafile: Metafiles can be published to the designated tracker.
- Download/Share: Peers can download and share content according to the Peer-Wire Protocol. Certain content is specified as an argument on startup.

Management & Monitoring

Management and monitoring functionality is available for CPs and the ISP. CP management is restricted to their context and ISP management targets the whole infrastructure, respectively each platform. Interfaces between management and NaDa platforms are currently not specified in detail. As an initial approach, they are implemented in the scripting language Ruby [RUB-2010]. Communication is realized by *remote procedure calls* (RPC) over TCP (see [RFC-1831]). Management is intended to invoke functionality of the realized tBitTorrent. Within this initial version, there is no interface for policy or billing implemented. The demonstrator allows communication between all peers that have successfully accomplished both protocols.

Customer Interfaces

Appropriate customer interfaces are not realized in the current state of NaDa development. However, the work of this thesis provides functionality to download resources in a trustworthy manner. It can be expected, that a NaDa platform will offer graphical user interfaces, similar to non commercial BitTorrent applications, in combination with content stores like iTunes[11].

Identity CA

A CA-Server is providing functionality of a *Certification Authority* (CA) to a CA-Client on the NaDa platform. Both are utilized for the creation of AIKs (see Attestation Identity Key

[11]http://www.apple.com/de/itunes/

in section 2.3.2) of a platform. The applications are derived from the Ethemba [BRE-2008] framework and demonstrator for TPM applications. Currently, client and server are integrated in tBitTorrent to provide functionality of a CA. Concepts of attestation and validation are integrated, too. Both implemented protocols (PWP, TP) rely on the certification of *Attestation Identity Keys* (AIK). AIKs represent the identity of the TPM during attestation. Authenticity can be proven by the appropriate public key of the CA. This key is assumed to be securely transferred to each peer.

3.2 Protocol Specification

3.2.1 Original Tracker Protocol

The tracker protocol is performed between a peer and a tracker. It specifies messages that are transferred by the *Hypertext Transfer Protocol* (HTTP). Hence, the tracker provides an HTTP service. Protocol 3 depicts the original tracker protocol. Only the required and none of the optional values are shown (see [COH-2008] for details). This protocol accomplishes the following tasks:

- Announce membership in a swarm.
- Request a list of swarm members.

Protocol 3 Original Tracker Protocol

0. Setup previous to protocol.

$$p: \quad metafile := \{announce,\ info := \{name,\ pieceLength,\ pieces,\ ...\}\}$$
$$t: \quad metafile$$

1. Peer sends request message.

$$p \rightarrow t: \quad url\{info\_hash, peer\_id, port, uploaded, downloaded, left, event\} \qquad (1)$$

2. Tracker sends response message.

$$t \rightarrow p: \quad interval, peers := \left\{\{peer\_id, ip, port\}_{p1..pN}\right\} \qquad (2)$$

(1a) Peer sends request message. If a peer is interested in downloading a resource or sharing a resource, a corresponding $metafile$ has to be obtained from any kind of file source, e.g. a known file server. In field *announce,* the *metafile* includes a URL of a tracker that is keeping track of the corresponding swarm tracker. The URL is enriched by several values forming a message to the tracker. Most important is the parameter $event$ that may be set to $started,$ $stopped$ or $completed.$ With event $started$, the peer announces that it wants to become member in a swarm. Leaving the swarm is announced by *stopped.* If a full replica of the resource is received the value *completed* is transmitted. A swarm is identified by the

$info\_hash$ that is known from the $metafile$. The values $peer\_id$ and $port$ are provided as contact information for other members of the swarm, whereas the fields $uploaded$, $downloaded$ and $left$, refer to the peer's sharing activity and the completeness of the local resource replica.

(1b) Tracker receives request message. Depending on the event of the message, the tracker processes its content. An initial call from a peer contains the event $started$. The tracker adds its contact details (ip, $port$) together with identifier $peer\_id$ to the list of swarm members. Other values are stored for further processing.

(2a) Tracker sends response message. If the $info\_hash$ is known to the tracker and there are already other peers associated to this swarm, the tracker adds the contact information of the peers to the response. All responses contain an interval, indicating a period of time that the peer should wait before repeating the request. Peers will do this frequently in order to the local list in synch and to signal the tracker that they are still participating.

(2b) Peer receives response message. With receipt of the response, the peer obtains an identifier and contact information of all peers that are currently sharing the requested resource. Each of them may be contacted using the PWP.

3.2.2 Extended Tracker Protocol

The extended protocol, as described by [KUN-2010] [12], is performed between a peer and a tracker. All steps have to be performed synchronously. The protocol accomplishes all tasks of the original version and several additional tasks:

- Mutual authentication of both parties (PKI).
- Integrity (software) of the peer platform is validated (attestation).
- An encrypted communication channel is established (shared secret exchanged).
- A download authorization token (called ticket) is exchanged.

Once all the tasks are accomplished, data may be exchanged between both parties using an encrypted and trustworthy connection. The complete extended tracker protocol is depicted in protocol 4 and described in detail below. The application is implemented accordingly. Key format, algorithm and provider configuration are described in section 3.4.

[12]The notation within this thesis is slightly amended: $AIKCert^{t}_{peer}$is depicted as $AIKCert^{peer}_{ca}$; $K^{pA}_{pub}, K^{pB}_{pub}$ is depicted as $hash(K^{pA}_{pub}||K^{pB}_{pub})$

All messages, except of the last, are synchronous messages. They have to be executed in the depicted order. Apart from the depicted content, a peer sends the fields $peer\_id$, $port$ and $event$ with each transmission. They are included as URL parameters. The tracker additionally sends an $event$. Events are used to identify current message type. All of these values are not shown in the following protocol description.

Protocol 4 Extended Tracker Protocol

0. Setup previous to protocol.

$$p: \quad metafile, (AIK^p_{pub}, AIK^p_{priv}), AIKCert^p_{ca}, S^t_{pub}$$

$$t: \quad metafile, (S^t_{pub}, S^t_{priv}), C^{ca}_{pub}$$

1. Peer sends an encrypted request.

$$p \rightarrow t: \quad enc\left\{request, K^p_{pub}, SML^p, AIKCert^p_{ca}\right\}_{S^t_{pub}} \tag{1}$$

2. Tracker computes the shared secret.

$$t: \quad K^{p,t} = K^t_{priv} \circ K^p_{pub} \tag{2}$$

3. Tracker sends a signature and a public key.

$$t \rightarrow p: \quad sig\left\{hash(K^p_{pub}||K^t_{pub})\right\}_{S^t_{priv}}, K^t_{pub} \tag{3}$$

4. Peer sends a quote.

$$p \rightarrow t: \quad quote\left\{hash(K^t_{pub}||K^p_{pub}), PCR_{0..n}\right\}_{AIK^p_{priv}} \tag{4}$$

5. Peer computes the shared secret.

$$p: \quad K^{p,t} = K^p_{priv} \circ K^t_{pub} \tag{5}$$

6. Tracker sends the encrypted response.

$$t \rightarrow p: \quad enc\{Data\}_{K^{p,t}} \tag{6}$$

(1a) Peer creates message. Message (1) is defined as a collection of values that are encrypted using a public key of the tracker ($enc\{values\}_{S^t_{pub}}$). The appropriate S^t_{pub} has to be inquired from a CA (or obtained from an extended metafile), already. This step is not part of the protocol. Furthermore, it is expected that the TCS is equipped with the certificate $AIKCert^p_{ca}$ in advance.

The $request$ field of message (1) contains the `info_hash`. It is a hashed value of the `info` field within the $metafile$. This $metafile$ has to be load from file system in order to extract the `info_hash` and to obtain the address of the tracker in field `announce`. The transmission of a public key K^p_{pub} is the inital part of the shared secret exchange, as described in protocol 1 on page 30. Before creation of the key pair $\left(K^p_{pub}, K^p_{priv}\right)$, appropriate random DH parameters have to be generated. Furthermore, peer p has to load its local SML^p from the file system. Loading has to be done rather close to protocol processing, since an out-dated SML^p could not be validated against the quote object in message (4). Finally, the certificate $AIKCert^p_{ca}$ has to be read from a key storage and added to message (1), thereafter.

Since asymmetric encryption is known to be comparably slow, symmetric encryption is applied instead. Message (1) may now be depicted as $enc\,\{values\}_{R^{p,t}}$,$enc\,\{R^{p,t}\}_{S^t_{pub}}$. Though the same security goals are achieved, it has to be noted that a gain in performance is only given in case of sufficiently large messages (see section 5.3.1). Before the previously described values can be encrypted, the random symmetric key $R^{p,t}$ has to be freshly generated. Decryption can only be computed using this symmetric key $R^{p,t}$. Thus this key must be transmitted to the tracker. A confidential transmission of key $R^{p,t}$ is ensured by encryption using the known public key S^t_{pub} of tracker *t*. Hence, $enc\,\{R^{p,t}\}_{S^t_{pub}}$ is added to message (1) as another value.

(1b) Tracker processes message. The tracker receives two encrypted values as described before. Initially the private key S^t_{priv} is applied to decrypt the symmetric random key $R^{p,t}$. Afterwards this key is applied to decrypt message (1). Both, $SML^p$and AIK certificate, are stored until message (4) is received because the values of that message are necessary to complete the appraisal. The signature is verified by the public key of the certification authority C^{ca}_{pub} (e.g. ISP) and the expiry of the certificate is verified using the current system time. The C^{ca}_{pub} has to be provided previous to protocol processing, e.g. during setup of the platform by the ISP. The $request$ is stored until creation of message (6).

(2) Tracker computes shared secret. With message (1) K^p_{pub} is transmitted to the tracker. Based on the peer key's parameter, the tracker computes an own key pair $\left(K^t_{pub}, K^t_{priv}\right)$ and the shared secret $K^{p,t} = K^t_{priv} \circ K^p_{pub}$ (symmetric key). The key is applied with message (6) and subsequent communication steps.

(3a) Tracker creates message. With message (3), the signature $sig\,\{hash(K^p_{pub}||K^t_{pub})\}_{S^t_{priv}}$ and the tracker public key K^t_{pub} have to be transmitted. The signature is computed on the hash of two concatenated keys. It ensures authenticity for the transmitted tracker public key and it acknowledges the receipt of the peer public key K^p_{pub} in step (1).

(3b) Peer processes message. With message (3) a signature and the public key of the tracker are received. The tracker key is stored for further processing during the following steps. The signature is verified with help of previously received values. The local peer public key K^p_{pub} and the received tracker public key K^t_{pub} are concatenated and hashed. For the verification the already available tracker public key S^t_{pub} is used ($ver\{signature\}_{S^t_{pub}}$).

(4a) Peer creates message. With message (4) a quote object transmitted to the tracker. The object is created by the local TPM. A selection of PCRs (currently PCR_{10}) and a nonce are signed by the private key AIK^p_{priv} of the TPM and a nonce. This nonce is obtained by concatenating and hashing the previously received tracker public key K^t_{pub} and the local key K^p_{pub} . The signature on K^p_{pub} confirms its authenticity which is not provided since receipt of message (1). The signature on K^t_{pub} acknowledges the receipt of this key. With the signed PCRs, the tracker is able to appraise the attestation.

(4b) Tracker processes message. The tracker receives a quote object including the hash of the two concatenated keys K^t_{pub} and K^p_{pub}(nonce), a list of PCRs (currently PCR_{10}) and a signature on both values. Now the tracker concatenates and hashes the locally available keys and compares the result with the received hash.

With the SML, received in message (1), the expected value of the PCR_{10} is calculated as described in chapter 2.3.4. The result is compared to the received PCR in message (4). They have to be equal in order to complete the appraisal of the integrity.

The SML contains identifiers and hashes of all measured applications on the peer system. These hash values were computed on the application data (binary, etc.). For each application hash in the SML a matching hash is searched in the list of known (valid) applications, the so called *Known Hash List* (KHL). Any unknown application hash in the SML would cause the appraisal to fail.

The quote object was created using the private key AIK^p_{priv} of the TPM. It can be verified by the previously calculated nonce, the PCR and the certified public key $AIKCert^p_{ca}$ received in message (1). A valid signature proves authenticity and integrity of the signed values and completes the attestation process.

Now, it is assured that the peer's integrity is provided. Since, integrity and authenticity of the received peer key K^p_{pub} and the transmitted tracker key K^p_{pub} is assured by validation of the quote object, it can be assumed that the previously created shared secret is only known to the peer and the tracker. It can be utilized in further communication.

(5) Peer computes shared secret. With message (3), the public key K^t_{pub} of the tracker was received and its integrity and authenticity could be verified. That key is based on parameters that are equal to the local public key K^t_{pub}. As described by the DH key exchange (see protocol 1 on page 30), the peer can now compute the shared secret key $K^{p,t} = K^p_{priv} \circ K^t_{pub}$ (symmetric key). It is applied in message (6) and subsequent communication.

(6a) Tracker crates message. The last communication step of the tracker protocol is message (6). During execution of the previous steps, a shared secret key $K^{p,t}$ could be established. That secret key can be used to encrypt all following messages ($enc\,\{Data\}_{K^{p,t}}$). Message (6) can be seen as an answer to the request in message (1). A list of data objects $Data = \{data_{p1}, data_{p2}, ..., data_{pn})$ has to be transmitted over this channel, now. The tracker searches appropriate peers that are sharing the requested resource. They must be successfully connected via the tracker protocol in advance.

A data object $data := \big(Address_{pB}, AIKCert_{ca}^{pB}, ticket\big)$ is created for each available peer. All data objects are added to the list. Each data object permits the peer, to perform the PWP with a dedicated peer (e.g. pB). The $Address_{pB}$ contains the *peer_id*, the IP-Address and the TCP port peer pB is listening on. A certificate $AIKCert_{ca}^{pB}$ is provided to verify the identity of pB during PWP communication.

Moreover, a $ticket := enc\left\{AIKCert_{ca}^{p}, resource, time\right\}_{K^{pB,t}}$ is included in each data object. Such a ticket authorizes peer pB to share the specified $resource$ with the owner of the certificate $AIKCert_{ca}^{p}$ within a period of $time$. It has to be transmitted to peer pB during PWP processing. Only peer pB is intended to decrypt its content. Since the ticket is not directly transmitted to peer pB, its integrity and confidentiality must be assured. This is achieved by encryption using a shared secret between peer pB and the tracker *t*. It implies that such a ticket can only be created if peer pB and tracker *t* have successfully performed the TP in advance.

(6b) Peer processes message. This is the final step in TP processing. The peer has to decrypt the message using the shared secret ($dec\left\{Data\right\}_{K^{p,t}}$). Elements of the data list are either stored or updated. Further processing is done using the PWP as described in the following sections.

3.2.3 Original Peer-Wire Protocol

The peer wire protocol is performed between peers. One peer initiates the communication with a handshake request, the other responds with a handshake acknowledge. Thereafter, each peer may asynchronously send and receive messages. This protocol accomplishes the following tasks:

- Both parties agree to exchange a certain resource.
- Both parties agree to communicate according to a certain protocol.
- Both parties exchange pieces of a certain resource.

Protocol 5 Original Peer-Wire Protocol

0. Setup previous to protocol.

$$pA: \quad info\_hash$$
$$pB: \quad info\_hash$$

1. Peer pA sends handshake message.

$$pA \rightarrow pB: \quad peer\_id^{pA}, info\_hash, protocol, reserved \tag{1}$$

2. Peer pB sends handshake response message.

$$pB \rightarrow pA: \quad peer\_id^{pB}, info\_hash, protocol, reserved \tag{2}$$

3. Peer pA and pB send asynchronous messages.

$$pA \leftrightarrow pB: \quad type, payload \tag{3}$$

(1) Peer pA sends handshake message. Peer pA creates the initial handshake message with the intension to contact peer pB. Contact information must be provided by a tracker previous to this protocol. The $info\_hash$ identifies the resource that is requested to share. Peer pB is expected to share it, since the tracker provided pB's contact information for the specific resource. Both peers agree on a protocol that is specified in field $protocol$. The field $reserved$ is not used, yet. With receipt of message (1), peer pB stores the senders contact details and its $peer\_id$.

(2) Peer pB sends handshake response message. If the protocol is known and agreeable, peer pB creates an own handshake message. The message includes its own $peer\_id$ together with an $info\_hash$ and a $protocol$ identifier. The values of the fields *Info_hash* and *protocol* are expected to be equal to the previously received values. They confirm the request. Field $reserved$ is not used, yet. Peer pB receives the acknowledging handshake and verifies its content.

(3) Peer pA and pB send asynchronous messages. Both peers exchange information about a specific resource to share (e.g. which pieces are available). Pieces may be downloaded until both parties obtained the same set of pieces. This might be a complete resource or only parts of it. Communication is ended at any point of time by one of the peers.

3.2.4 Extended Peer-Wire Protocol

The extension to the peer-wire protocol is based on a proposal of Kunzte et al. in [KUN-2010]. It accomplishes tasks of the original protocol and the additional tasks:

- Mutual authentication of both parties (PKI)
- Indirect attestation by a third party token (ticket).
- An encrypted communication is established (shared secret exchanged).
- A download authorization token (called ticket) is exchanged and verified.

This implementation (see 6 on the following page) embeds the enhancement between handshake request and response of the original protocol. Accordingly, message (1) serves as initial handshake and message (6) carries the acknowledging handshake. Currently, the fields *protocol* and *reserved* of the original handshake are not used and not transmitted with message (1). Contrary to the proposal, the *peerID* is added to the message (1) in order to correspond with the handshake message. All messages, except of the last, are synchronous messages. They have to be executed in the depicted order. It is assumed that both communicating parties have already completed the TP with the same tracker. Key format, algorithm and provider configuration are described in section 3.4.

Protocol 6 Extended Peer-Wire Protocol

0. Setup previous to protocol.

$$pA: \quad (AIK_{pub}^{pA}, AIK_{priv}^{pA}) \text{ , } data := \left(Address_{pB}, AIKCert_{ca}^{pB}, ticket\right)$$

$$pB: \quad (AIK_{pub}^{pB}, AIK_{priv}^{pB}) \text{ , } K^{pB,t}, \; info\_hash$$

1. Peer pA sends the initial request (handshake).

$$pA \rightarrow pB: \quad peerID^{pA} \text{ , } K_{pub}^{pA} \text{ , } ticket := enc\left\{AIKCert_{ca}^{pA}, resource, time\right\}_{K^{pB,t}} \tag{1}$$

2. Peer pB sends a response.

$$pB \rightarrow pA: \quad K_{pub}^{pB} \text{ , } quote\left\{hash(K_{pub}^{pA}||K_{pub}^{pB}), PCR_{none}\right\}_{AIK_{priv}^{pB}} \tag{2}$$

3. Peer pA sends a request.

$$pA \rightarrow pB: \quad quote\left\{hash(K_{pub}^{pB}||K_{pub}^{pA}), PCR_{none}\right\}_{AIK_{priv}^{pA}} \tag{3}$$

4. Peer pA computes the shared secret.

$$pA: \quad K^{pA,pB} = K_{priv}^{pA} \circ K_{pub}^{pB} \tag{4}$$

5. Peer pB computes the shared secret.

$$pB: \quad K^{pA,pB} = K_{priv}^{pB} \circ K_{pub}^{pA} \tag{4}$$

6. Peer pB sends the final encrypted response (handshake).

$$pB \rightarrow pA: \quad enc\{Content\}_{K^{pA,pB}} \tag{5}$$

(1a) pA creates message. Based on the tracker response of the extended tracker protocol, peer pA may have received a $data$ object. It implies, that peer pB is sharing the requested resource. Peer pA has to load the address of peer pB ($Address_{pB}$), the certificate of its TPM public key and the corresponding $ticket$ for further processing. Locally, DH parameters and the key pair$\left(K_{pub}^{pA}, K_{priv}^{pA}\right)$ are created. They are needed to establish a shared secret key according to the DH protocol. The $ticket$ and the public key K_{pub}^{pA} are transmitted to peer pB with message (1). Corresponding to the original PWP handshake, $peerID^{pA}$ is sent, too.

(1b) pB processes message. Peer pB receives message (1) containing a $ticket$, the public key K_{pub}^{pA} and the $peerID^{pA}$. This message serves as an initial handshake corresponding to the original protocol. At first, peer pB decrypts the ticket using the secret key $K^{pB,t}$ that is shared with tracker t. The key was already exchanged during tracker protocol execution, prior to this protocol. Peer pB verifies the requested $resource$ (info_hash) and the validity of the ticket with help of the provided $time$. The received certificate $AIKCert_{ca}^{pA}$ is stored but not verified, since this is expected to be done by the (trusted) tracker already. A local key pair $\left(K_{pub}^{pB}, K_{priv}^{pB}\right)$ is created according to the parameters of the received key K_{pub}^{pA}.

(2a) pB creates message. With message (2), a quote object and the public key K_{pub}^{pB} of peer pB are transmitted to peer pA. The quote object is obtained from the TPM and thereby signed with its private key AIK^{pB}_{priv}. Two concatenated and hashed keys (K^{pA}_{pub},K^{pB}_{pub}) are provided as nonce. PCRs are not included (PCR_{none}) since an attestation is not defined for the PWP. Instead, the ticket from the trusted tracker vouches for the integrity of both peers. With the quote object, peer pB can be authenticated, the receipt of K^{pA}_{pub} is acknowledged and the authenticity of K^{pB}_{pub} can be verified.

(2b) pA processes message. Peer pA receives message (2) containing key K^{pB}_{pub} and a quote object. The key is stored and the quote object is verified. Since no PCR values are included in the quote object, an attestation cannot be performed. This is expected to be done by the (trusted) tracker, in advance. But the public key AIK^{pB}_{pub}, included in the certificate $AIKCert^{pB}_{ca}$, is applied during verification of the quote object that is signed by the key AIK^{pB}_{priv}. The verification proves the authenticity of the message. Additionally, peer pA verifies the nonce. It is computed as hash of the two concatenated keys (K^{pA}_{pub},K^{pB}_{pub}). Herewith, it is ensured that peer *pB* has received the authentic key K^{pA}_{pub} and that peer pA has received the authentic key K^{pB}_{pub}.

(3a) pA creates message. With message (3) a single quote object is transmitted to peer pB. This quote object is signed using the private key AIK^{pA}_{priv} of pA. It contains the hash of two concatenated keys (K^{pB}_{pub},K^{pA}_{pub}). PCR values do not have to be included in this quote object. It is expected that the integrity of peer pA has already been verified by the (trusted) tracker.

(3b) pB processes message. Peer pB receives message (3) containing a single quote object that is signed by key AIK^{pA}_{priv}. The signature is verified by the public key AIK^{pA}_{pub} of pA. This key is obtained from the certificate $AIKCert^{pA}_{ca}$ that was received within the $ticket$ of message (1). It is not necessary to verify the signature of certificate $AIKCert^{pA}_{ca}$, since this is expected to be done by the (trusted) tracker.

The quoted hash value however, has to be verified in order to assure that peer pA has received the right public key K^{pB}_{pub} and that the public key K^{pA}_{pub} received in message (1) is authentic and not altered. For that purpose, the local keys K^{pB}_{pub} and K^{pA}_{pub} are concatenated and hashed. The resulting value must match the hash within the quote object. Since the complete quote was previously verified, the authenticity and integrity of the hash is given, already.

(4) pA and pB compute shared secret. Now, both parties compute the shared secret and complete the Diffie-Hellman protocol. Peer pA computes $K^{pA,pB} = K^{pA}_{priv} \circ K^{pB}_{pub}$ and peer pB computes $K^{pA,pB} = K^{pB}_{priv} \circ K^{pA}_{pub}$. All following communication is encrypted by the symmetric key $K^{pA,pB}$.

(5a) pB creates message. With transmission of message (5), peer pB completes the protocol extension. This message serves as response to message (1). It carries a content that is encrypted using the shared secret key $K^{pA,pB}$. The content just contains a `handshake` message according to the original PWP.

(5b) pA processes message. Peer pA receives message (5) and decrypts it using the shared secret key $K^{pA,pB}$. From now on, both parties may exchange asynchronous messages according to the original PWP. Each of those messages is completely encrypted as described in message (5).

3.3 Communication Framework

3.3.1 Tracker Protocol

Peer and tracker communicate on the application layer of the internet model [RFC-1122]. They realize a client/server topology where the tracker is a server and peers are clients. Messages are transmitted via the *Hypertext Transfer Protocol* (HTTP, [RFC-2616]).

The tracker integrates the Simple[GAL-2002] web framework. It listens to specified ports of the *Transmission Control Protocol* (TCP) and passes incoming requests to message handling tracker classes. A tracker provides upload services and, so called tracker services. The `UploadService` just receives metafiles and the `TrackerServiceTrusted` processes requests of the TP (see class diagram 7 on the next page). Requests of the TP are processed by an instance of `PeerConnection` for each connected peer.

On the peer site, HTTP requests are transmitted by the `ClientHttpRequest` and an `URLConnection` that provides input and output streams of the network communication. Requests are created and processed by the `PeerUpdaterTrusted` thread. It creates a message object and a URL string including the current event. With the extended tracker protocol new events and the corresponding messages `Encryption` (1), `SignQuote` (3), `Quote` (4), `DataQuote` (6), are introduced. Unlike the original protocol, the extended version uses the HTTP `POST` method in order to transmit the required messages. Previously, the `GET` method without attached content was sufficient.

Message Format

Peer and tracker communicate over the HTTP protocol [RFC-2616]. HTTP-Request messages and HTTP-Response messages are exchanged. Throughout the thesis it is assumed, that HTTP-messages are transported over TCP/IP packets. Each tracker protocol message is represented by an object. Message elements are defined in section 3.2. Within the message object, they are represented as byte arrays. Message objects are serialized to a string that can be carried within HTTP. The string is given to the Simple web framework that is handling further transportation steps to its destination. Some messages are of variable length. E.g. the content length might

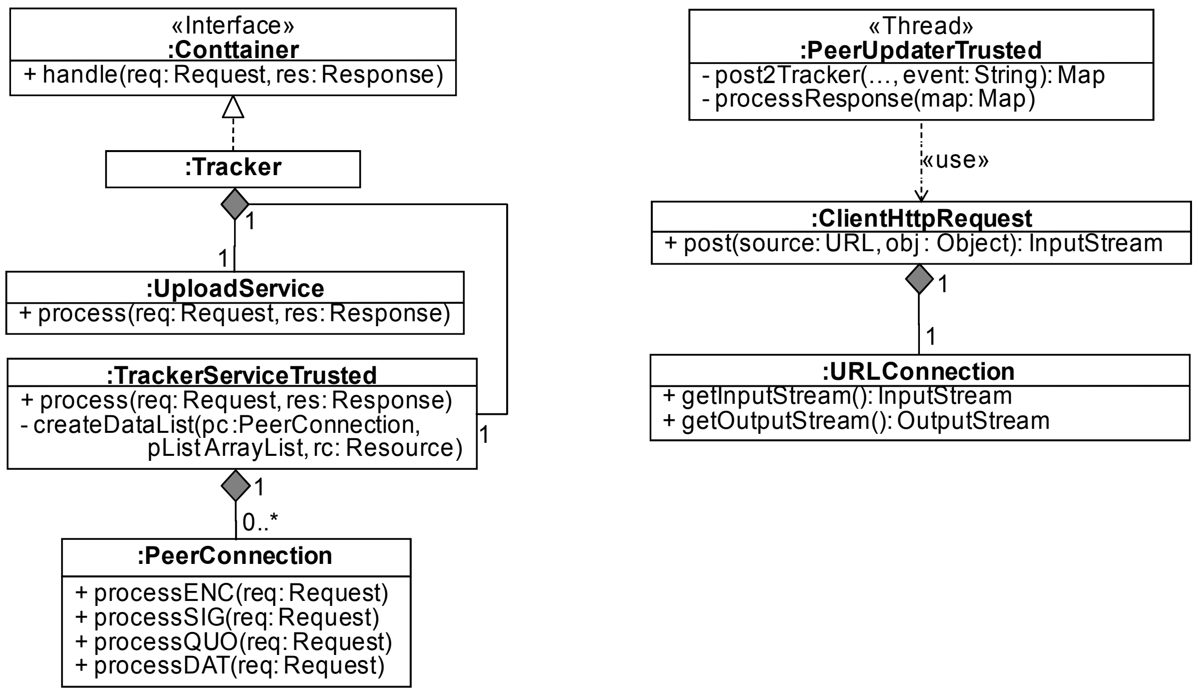

«Interface»
:Conttainer
+ handle(req: Request, res: Response)
:Tracker
1
1
:UploadService
+ process(req: Request, res: Response)
:TrackerServiceTrusted
+ process(req: Request, res: Response)
- createDataList(pc :PeerConnection, pList ArrayList, rc: Resource)
1
1
0..*
:PeerConnection
+ processENC(req: Request)
+ processSIG(req: Request)
+ processQUO(req: Request)
+ processDAT(req: Request)
«Thread»
:PeerUpdaterTrusted
- post2Tracker(..., event: String): Map
- processResponse(map: Map)
«use»
:ClientHttpRequest
+ post(source: URL, obj : Object): InputStream
1
1
:URLConnection
+ getInputStream(): InputStream
+ getOutputStream(): OutputStream

Figure 7: Communication Framework of the Tracker Protocol.

vary for formats of certificates and keys, SMLs, amount of peers and others. Exemplary content lengths could be extracted from a network analysis tool (wireshark[13]). In application revision 55, these are: 60,269 bytes for message (1), 876 bytes for message (3), 1,169 bytes for message (4) and 3,752 bytes for message (6). The complete captured network traffic of revision 47 and revision 55 can be found on the thesis DVD as described in section C.2 on page XX.

An example of a HTTP POST request is shown in format 1 on the next page and an example of an HTTP response is shown in format 2 on the following page. The POST request contains the `/announce` string and the content length of the following serialized message object. In the current version, still the values info_hash, peer_id, port and event are transmitted as plaintext. That corresponds with the original protocol. Depending on a security analysis of a concrete business model, concepts for hiding them might become necessary. No other TP fields are transmitted anymore. A successful response contains `200 OK`, the server type and a serialized message object as content.

[13]www.wireshark.org

Format 1 Example of an HTTP POST Request

```
POST /announce?peer_id=VALUE&port=VALUE&event=VALUE HTTP/1.1
Content-Type: multipart/form-data; boundary=THIS_STRING_SEPARATES
User-Agent: Java/1.6.0_0
Host: 127.0.0.1:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: LENGTH_SERIALIZED_OBJECT
```

Format 2 Example of an HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/plain
Connection: keep-alive
Transfer-Encoding: chunked
Server: Trusted BitTorrent Tracker
Date: Thu, 12 Aug 2010 08:12:14 GMT
...
EVENTVALUE ... CONTENT_SERIALIZED_OBJECT
```

3.3.2 Peer-Wire Protocol

In jBitTorrent a *message sender* (MS) thread and a *message receiver* (MR) thread realize the communication to another peer. For each peer relationship, separate sender and receiver are started. Such a relationship is designed as *download task* (DT). One or more of these tasks are managed by a *download manager* (DM). Sender and receiver communicate over input and output streams of a TCP/IP socket connection from the standard Java `net` API. A *Connection-Listener* (CL) is listening to TCP ports for incoming connections and creates sockets to remote peers. The threads of a peer application (MS, MR, DT, DM, CL) mainly communicate asynchronously. Java `EventListener` are applied, together with a `LinkedBlockingQueue` between DT and MR. In the `LinkedBlockingQueue`, message objects are linked and ordered according to the *first-in-first-out* (FIFO) principle. Concurrent access and modification may be performed, since it is thread-safe. The `EventListener` enables threads to exchange events. Listening threads have to implement listener interfaces and provide their reference to the threads that are creating events.

A static overview of the communication framework is depicted as class diagram in figure 8 on the next page. Several listeners interfaces extend from `EventListener` and declare public methods that are implemented in concrete classes. All depicted concrete classes inherit thread functionality. The `DownloadManager` is composed of a `ConnectionListener`, a

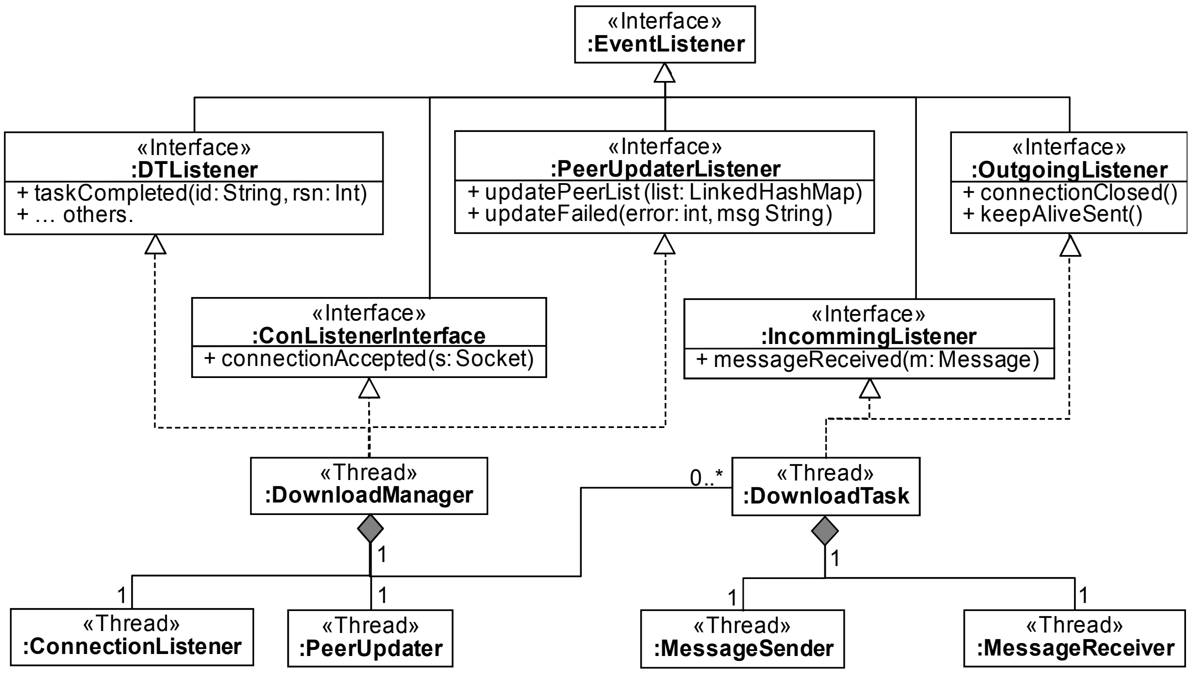


Figure 8: Communication Framework of the Peer-Wire Protocol.

`PeerUpdater` and as many `DownloadTasks` as remote peers are connected. Each `DownloadTask` is composed of a `MessageSender` and a `MessageReceiver`. They encapsulate the communication to a TCP Socket. All depicted messages return `void`.

The communication between peers is defined by messages. The original peer-wire protocol describes two messages types, *handshake* (HS) and *peer protocol* (PP). An additional type, called *trusted protocol* (TP), is introduced to cover the messages described by the extended PWP. With completion of the extended protocol, the complete content stream that is carried over TCP, is encrypted. Sender and receiver are extended with encrypting and decrypting functionality. Over this secured channel PP messages are exchanged, as well as, a HS message in the last message (6) of the protocol extension.

Messages in jBitTorrent are polymorph. The communication components operate on an abstract message, providing common interfaces and properties. Messages HS, PP and TP implement its interfaces `generate` and `encrypt`. With `generate`, the content of each message is concatenated to a byte array and with `encrypt` the byte array is encrypted with the provided secret key. TP messages are not encrypted with the shared secret, since the secret key is not available at that time. Decryption is done by the message receiver. The extension follows the provided design of jBitTorrent. In future development, the design should be amended to become symmetrical. That simplifies maintainability and understandability. E.g. messages may provide reverse functions to generate and encrypt alike.

Message Format

A *handshake* (HS) message consists of four fixed length values and the protocol name with a variable length. The first byte is the `length` field indicating the length of the `protocol` name field (maximum $2^8 - 1$ bytes). jBitTorrent defines "BitTorrent protocol" as protocol, the extended version is named "tBitTorrent v0.1". The next 8 bytes are `reserved` for future extensions followed by `fileID` and `peerID` each with 20 bytes.

length	**protocol**	**reserved**	**fileID**	**peerID**
1	...	8	20	20

Figure 9: *Handshake* (HS) Message Format.

With the *peer protocol* (PP) message an asynchronous cooperative communication between peers is defined. PP messages are of variable length. All PP messages start with four bytes indicating the overall `length` of the message in bytes excluding the length field. The length field is followed by a fixed one byte `id` of the message and the variable `payload` (maximum $2^{32} - 1 - 1$ bytes). Specified id's in BitTorrent are `0=Keep_Alive, 1=Choke, 2=Unchoke, 3=Interested, 4=Not_Interested, 5=Have, 6=Bitfield, 7=Request, 8=Piece, 9=Cancel and 10=Port.`

length	**id**	**payload**
4	1	...

Figure 10: *Peer Protocol* (PP) Message Format.

The *Trust Protocol* (TP) messages are introduced according to the extended PWP definition. Unlike PP and HS, the TP messages start with one byte `id` field. Except of message (6) each message of the extended PWP is indicated by an own `id`. Implemented extended id's are:

- `11=Trust_REQ1` for PWP message (1)
- `12=Trust_RES1` for PWP message (2)
- `13=Trust_REQ2` for PWP message (3)

The last message (6) is an encrypted HS message and not a TP message. The payload of these TP messages contains fixed pairs of length and value (`key, ticket, quote, peerID`) fields. If a value is not provided with the indicated message, the corresponding length field (`keyLength, ticketLength, quoteLength, peerIDLength`) is set to zero. The maximum size of all the value fields is $2^{32} - 1$. A complete capture of the network traffic of revision 47 and revision 55 can be found on the thesis DVD as described in section C.2 on page XX.

id	keyLength	key	ticketLength	ticket	quoteLength	quote	peerIDLength	peerID
1	4	...	4	...	4	...	4	...

Figure 11: *Trust Protocol* (TP) Message Format.

3.4 Cryptography Integration

Previously, the implementation of the specified cryptographic protocol was described. It is based on cryptographic keys and algorithms. All of the introduced cryptographic algorithms are integrated from certain providers. Their functionality, called services, is accessed using the framework of *Java Cryptography Extension* (JCE)[ORA-2010a] respectively *Java Cryptographic Architecture* (JCA)[ORA-2010a]. Additionally, the IAIK TSS (iaik.tc.tss [IAI-2010b]) library and the TPM [INF-2010] provide cryptographic keys and algorithms.

3.4.1 Architecture

The framework defines a general architecture for common cryptographic functionality. Several providers may implement their own solutions, but offering the same interfaces respectively services. Programmers may choose the service of a provider according to specific requirements. Within this thesis, services of the providers IAIK JCA/JCE[IAI-2010a], SunJSSE, SunJCE, Sun and SunRSASign are applied. Each service is provided by a so called engine class:

- **Cipher**: Provide algorithms for encryption and decryption of data. It is initialized with keys and configured with an algorithm type: e.g. several types are available for symmetric and asymmetric systems.
- **KeyGenerator**: Creates a secret key according to the specified key size and algorithm.
- **KeyAgreement**: Computes a shared secret between two communication parties. It is based on previously exchanged public keys and parameters.
- **KeyPairGenerator:** This service is used to create key pairs (public and private) according to a specified asymmetric crypto system.
- **AlgorithmParameterGenerator**: Creates a set of parameters according to the specified algorithm (e.g. DH).

- **MessageDigest**: Provides algorithms to calculate hashes of the given data.
- **Signature**: Offers signing and verification. They are initialized with a private or a public key.

3.4.2 Application

Within the developed application, several engines and keys are integrated. The table 3 provides implementation details to the keys that are specified within both protocols. For each kind of key, the size, format and engine are shown. A summary of all applied services is shown in table 4 on the following page. For each service the applied algorithm details and the used engine is depicted. The provider configuration for each key and service definition depends on the respective revision. It can be found in appendix B. It has to be noted, that default key lengths are used as far as possible. They are configurable throughout the application. The key-length has a strong influence on the protection level of the data. The longer the key, the harder it is to break (e.g. by brute force) it. On the other hand, performance decreases with longer keys. The integrated keys and algorithms represent reference values. For each operational environment it must be decided, whether this configuration is adequate. This might depend on many factors e.g. how valuable the content is or how long the content must be protected.

Keys	Size(bit)	Type	Format	Engine
S^t_{pub}, S^t_{priv}	512	Asymmetric	RSA	KeyPairGenerator
K^i_{pub}, K^i_{priv}	1024	Asymmetric	DH	KeyPairGenerator
K^i_{pub}, K^i_{priv}	-	Parameter	DH	AlgorithmParamGen
$K^{i,j}$	56	Symmetric	DES	KeyAgreement
$R^{i,j}$	128	Symmetric	AES	KeyGenerator
$AIKCert^i_{ca}$	-	Certificate	X509	AIKCertificate
$AIK^i_{pub}, AIK^i_{priv}$	2048	Asymmetric	RSA	TPM
$C^{ca}_{pub}, C^{ca}_{pub}$	512	Asym.	RSA	KeyPairGenerator

Table 3: Applied Cryptographic Keys.

Services	Format, Algorithm, Mode	Engine
$quote\{\}$	RSA	TPM
$ver\{quote\}$	RSA	TSS - TcCrypto
$hash$	SHA1	MessageDigest
$enc/dec\{\}_R$	AES/ECB/PKCS5Padding	Cipher
$enc/dec\{\}_{S_{pub}}$	RSA	Cipher
$enc/dec\{\}_K$	DES/ECB/PKCS5Padding	Cipher
$sig/ver\{\}$	SHA1withRSA	Signature
$ver\{AIKCert^i_{ca}\}$	X509	X509Certificate

Table 4: Applied Cryptography Services.

3.4.3 TPM & TSS

Trusted Computing Systems implement a hardware based security anchor, the TPM. Several applications and interfaces are available to access its services. During this thesis, platform dependent and independent versions are applied. In a development environment, where no hardware TPM is available, the emulator of the ETH Zurich [STR-2010] can be used. The access to a TPM is described in the specification of a *Trusted Software Stack* (TSS) by TCG. Between application and TPM, it defines the four layers Crypto Service Provider, *TSS Service Provider* (TSP), *TSS Core Service* (TCS) and TSS Device Driver Library. Crypto service providers implement the stack according to the specified APIs. Software for each layer can be obtained from different provider. For the tBitTorrent application, the IAIK TSS [IAI-2010b] is used during development. For measurement in the laboratory (see next chapter), the IAIK TSS Wrapper v.0.4 beta [IAI-2010b] provides access to the TSP layer. The wrapper accesses the platform dependent TCS layer. This is part of the operating system installation.

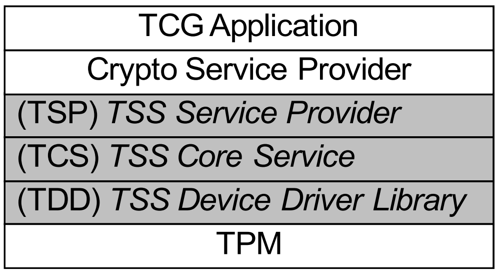


Figure 12: *Trusted Software Stack* (TSS).

3.5 Conclusion

An application could be realized that demonstrates usability of the extended protocols. It approves the concept of trustworthy resource exchange in a BitTorrent based network. Peers can communicate with other peers and trackers using trustworthy protocols and a TPM. Communication between enhanced entities and original entities is not possible anymore. The enhancements describe new protocols. Trackers do not yet provide management functionality that would meet operational requirements. It is limited to a laboratory scale environment as described in the next chapter. Due to the complexity of the BitTorrent application, only a very limited level of quality can be assured within this thesis. An extensive testing phase, including scenarios of many participating peers in different states is required. However, this is neither intended nor done within this thesis. The solution integrates cryptographic functionality according to the java cryptographic architecture. As a consequence, algorithms and formats can be configured according to the requirements that are expected from different operational environment.

4 Performance Evaluation

With the implemented protocol extensions, cryptographic techniques are applied to P2P communication. It is expected, that this will lead to a broader acceptance and application of the P2P data distribution in commercial environments. The applicability of such a trustworthy P2P technology depends upon other requirements, on its performance. It is of importance that relevant use cases like the establishment of a connection, download, update, etc. are still performed within an acceptable time. The goal of this evaluation is to gain performance characteristics of the extended protocols by measuring its processing time. Time consuming areas are identified to provide a basis for further development and optimization. Results are compared with measures of the original jBitTorrent. Since protocols are under test, dedicated application evaluation like stress- or load testing is not performed.

All experiments are conducted in a new physical (laboratory) environment that was not used for measurement before. No simulators or emulators are involved. Therefore, the results are expected to be comparable to a production like environment. Within this chapter the components of the laboratory, including their current configuration, measurement technique and experiment scenarios, are described in detail. Measurement results are presented and discussed.

4.1 Experiment Process

Experiments are created in the development environment by an experimenter. Thereafter, they are transferred to the experiment environment. The experimenter initiates the automatic execution. Results are collected and transferred back to the development environment. Now they can be computed and analysed.

In order to develop an experiment (scenario), the experimenter has to do the following:

- Define measurement points within the program code in the development environment.
- Compile the applications to run in the experiment environment.
- Configure/prepare the applications according to the experiment scenario.
- Create *experiment description* (ED): interpretable by measurement framework.
- Transfer all files to the *experiment controller* (EC).
- Instruct the EC to execute the experiment.
- Collect results from EC and transfer them to development environment.
- Analyse results and compute measures from measurement points.

4.2 Measurement Configuration

Experiments are conducted in a physical laboratory. In order to interpret the experiment results, the configuration of all relevant (strongly influencing) components within the laboratory has to be taken into account. Furthermore, the configuration description should be sufficient to reproduce and verify the measured results up to an acceptable degree.

4.2.1 Network

Experiments are conducted in a separated TCP/IP based network as depicted in figure 13. Ten nodes are physically connected to a router. They are logically grouped to the IP network 151. The workstation in network 150 is connected to the router as well. It serves as an experiment controller according to the framework description below. Development workstations are connected to the router via the intranet. Access to the experiment controller and each node is available via the *Secure Shell* (SSH) [RFC-4250] protocol. In that way the experiment controller may be instructed to initiate an experiment. Experiment traffic and control traffic are currently transported over the same network. But measures of the applied measurement technique (see section 4.3) are independent from the network.

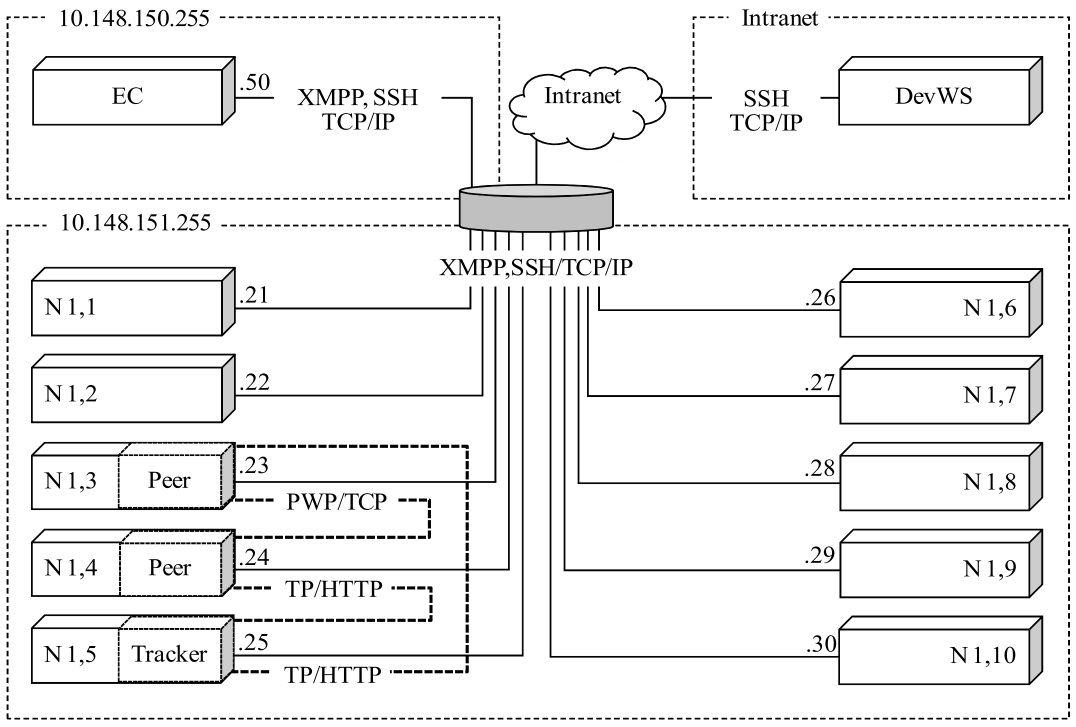


Figure 13: Network setup.

4.2.2 Framework

In order to create reproducible and comprehensible experiments an external *control, management and measurement framework* (OMF [RAK-2010]) is applied. It provides a set of tools that have to be installed on each participating network node. They are used to instrument experiments, execute them and collect the results. Network nodes are connected by OMF applications using the *extensible messaging and presence protocol* (XMPP [RFC-3920 - RFC-3923]). The framework is developed and maintained by a cooperation of NICTA [NIC-2010] and Winlab [RUT-2010].

Experimenters create *experiment descriptions* (ED) in a language, called *OMF Experiment Description Language* (OEDL), that is based on the scripting language Ruby [RUB-2010]. Descriptions contain definitions of the nodes that are participating in the experiment, applications to be loaded and executed on each node and the execution procedure. Within context of OMF, nodes are regarded as resources. Measurement points may be defined using the OMF measurement library. The experiment description is submitted to the *experiment controller* (EC). It interprets the description and initiates the execution of the experiment. Applications, e.g. BitTorrent binaries, are transferred to the *resource controller* (RC) of the specified nodes. According to the execution procedure, measurement results are collected at each node and transferred back to the experiment controller. Results may be accessed and analysed during or after completion of the experiment.

4.2.3 Nodes

Hardware

Each node of the experimental network consists of the same physical hardware components. These are: an Intel Atom Z530 (CPU) 1.60 GHz (Clock rate) with a single core / two threads and 32-bit instruction set (Word Size), 2GB DDR2 RAM (Memory), Infineon IFX Chip Version 1.2.1.2 (TPM)[INF-2010] operating from a 33 MHz clock, Intel 82574L Gigabit Network Connection (Network) and a SAMSUNG M6 Series HM320JI hard disk drive with 5400 revolutions per minute.

Software

On all nodes the operating system Ubuntu 4.4.1 with kernel version 2.6.31.22 is installed. Additional software applications are: IAIK/OpenTC jTSS Wrapper version 0.4beta (TSP), Trousers version 0.3.1-7ubuntu3 (TSS) and OMF Resource Controller 5.2.324 (OMF). The node 1,1 additionally serves as HTTP gateway, but it will not take part in the following experiments. The application under test will be executed within a *Java Virtual Machine* (JVM): Java version

1.6.0_0 (OpenJDK Runtime environment (IcedTea6 1.6.1) (6b16-1.6.1-3ubuntu1), OpenJDK Server VM (build 14.0-b16, mixed mode). The OMF resource controller application is installed and active on each node. For each experiment a list of processes per node is inquired from the operating system and stored in the corresponding logfile.

Java virtual machine

The JVM has a major influence to the processing time of applications. The current settings of each experiment is inquired at startup on each node and stored in the corresponding logfile. A typical configuration is shown in appendix A.3 on page VI. All experiments are conducted according to this configuration. Especially, the *Just-in-Time compiler* (JIT) and garbage collection is active for all experiments.

4.3 Measurement Technique

The processing time of two extended protocols, the PWP and theTP, has to be measured. These protocols are implemented as described in chapter 3. It is intended to reveal areas of low performance. Therefore, a detailed measurement of each protocol step is necessary. Some well known open source network measurement tools could be used for this measurement. However, they seem not to be applicable if more detailed measurement of the algorithms (e.g. SML verification, TPM operations, key creation) is required. Hence, measurement points within the application itself are introduced.

4.3.1 Method

At each measurement point a time stamp is obtained and printed to the standard out stream which is read by OMF and written to an experiment log file. The time stamp represents the time at which an event is recorded by the measurement system, not the time when the event occurred. However, this deviation is expected to be acceptable since accuracy in milliseconds is required.

The OMF framework provides time stamps in *seconds* (s). Initial experiments showed that this precision is not sufficient, since the processing time of protocol steps varies from *milliseconds*[14] (ms) to seconds. Instead, the timestamps provided by the *java virtual machine* (JVM) are used. Java offers timestamps represented as milliseconds (function `currentTimeMillis` of `System` class) elapsed from the 1.1.1970 and represented as nanoseconds (function `nanoTime` of `System` class) elapsed from an arbitrary point of time. Since the precision of *nanoseconds*[15]

[14]A millisecond is a thousandth of a second ($10^{-3}s$). One second will be represented in milliseconds as $1,000\,ms$. Thousands are separated by a colon.

[15]A nanosecond is a billionth (US) of a second ($10^{-9}s$). One second will be represented in nanoseconds as $1,000,000,000\,ns$. Thousands are separated by a colon.

(ns) is higher and the current date which can be derived from the milliseconds is not relevant in this context, measuring is done in nanoseconds. However, the accuracy in nanoseconds is not guaranteed[16]. The provider just claims that values of the most precise available system timer are provided. This may vary for each platform. Based on the description of the measured platform (section 4.2.3) it is assumed, that the most precise time values are derived from the CPU instruction clock. The current clock rate of 1.6GHz, is expected to provide 1.6 tick counts per nanosecond.

An example measurement is shown in algorithm 1. Time stamps are obtained and sent to the standard out stream where they are captured by the OMF. The time elapsed between two measurement points (e.g. MP00TTPSEND and MP00TTPRCVD) is computed simply by subtracting a value from another, which is taken thereafter. But this is done by an evaluation process after the experiment is finished. Hence, it will not affect measuring. During measurement, the logging framework log4j is disabled. It was observed, that it affects the performance significantly.

Algorithm 1 Example measurement points

```
System.out.println(System.nanoTime() + " " + Measure.MP00TTPSEND)
// ... the code to be measured ...
System.out.println(System.nanoTime() + " " + Measure.MP00TTPRCVD)
```

Distortion

Each measurement technique consumes processing time and, therefore, extends the measured processing time itself. In order to quantify that influence, the experiment A.4.1 (see appendix) was conducted. With that experiment 1,002 measurement points were created at the target system. Afterwards the elapsed time between the first and the last measurement point was computed. The result is a processing time of 0,167ms (MEAN) for 1,000 respectively 0.167ms per measurement point. Additionally, it has to be taken into account that values equal and greater than 0.5ms are rounded up to 1ms. Therefore, a measured processing time with more than ≈ 3 measurement points in between would lead to different results for a resolution in ms. But that assumption is rather a worst case assumption. The results show that the mean per measurement point decreases with a decreasing number of measurement points taken. In a typical experiment (see A.4.3) the maximum amount of measurement points per node is 20.

[16]"Returns the current value of the most precise available system timer, in nanoseconds. ... This method provides nanosecond precision, but not necessarily nanosecond accuracy. No guarantees are made about how frequently values change. Differences in successive calls that span greater than approximately 292 years (263 nanoseconds) will not accurately compute elapsed time due to numerical overflow."[SUN-2010]

4.3.2 Measure Definition

Within this evaluation several types of measures have to be distinguished. First of all, basic measures are to be computed by subtraction of timestamps. They indicate the elapsed processing time of an algorithm within an application (e.g. generating a key, measuring TPM quote, creating a response to a request, etc.). Where applicable basic measures are composed to aggregated measures (e.g. processing time for protocol steps are summarized). Statistical measures are computed on the basis of measurement sets of basic measures (procTime in figure 14). With the *algorithm processing time* (AlgPT) two measurement points are computed, that are defined within a step of the protocol (e.g. $procTime_0$ in figure 14, right).

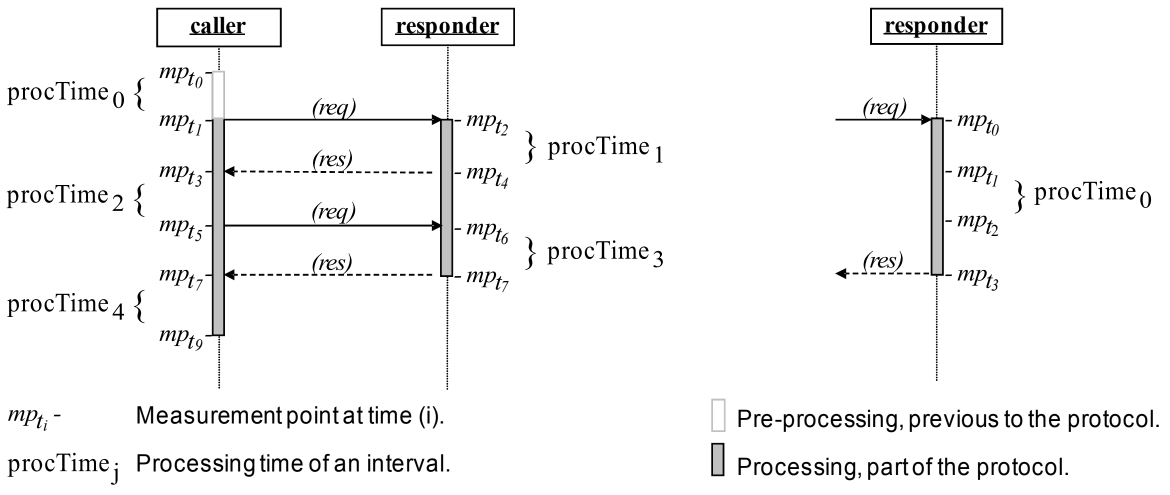


Figure 14: Measurement Points and Processing Time.

The experiments are conducted in a not completely controlled environment. Many soft and hardware components as well as their physical environment are influencing the experiments. In practice, such experiments can never be repeated exactly. An idealized environment must be assumed. As a consequence there will always be a deviation in the results of equally conducted experiments. The deviation is very important for the interpretation of the results. In order to measure it, experiments are repeated $\{exp_1, exp_2, ..., exp_n\}$ up to an acceptable sample size n. Then statistical measures can be computed. For all measures a sample size of 100 is defined. With the given sample size, the experiments of the main scenario take about 800 minutes. The execution of experiments is automated and scheduled to run out of the usual working houres in order to minimize external influences.

A list of timestamps, at certain measurement points $\{mp_{t_0}, mp_{t_1}, ..., mp_{t_k}\}$, is evaluated during each experiment (figure 14). A measurement point mp represents a timestamp as integer value. By subtraction of appropriate measurement points the measure $m = mp_{t+1} - mp_t$ is computed. According to experiment definition a list of measures $\{m_1, m_2, ..., m_j\}$ is the result of each experiment. Since experiments are repeated n-times, a sample set of values for each measure $set_i = \{m_{i_1}, m_{i_2}, ..., m_{i_n}\}$ is collected. Based on these sets, several statistical measures are calculated.

4.3.3 Statistical Measures

For a general set of values $X = \{x_1, x_2, ..., x_n\}$ (where $m_i = x$) statistical measures are defined as follows:

- The minimum $MIN = min(x_1, x_2, ..., x_n)$ is the smallest value within the sample set.
- The maximum $MAX = max(x_1, x_2, ..., x_n)$ is the largest value within the sample set.
- The median $MED = \begin{cases} x_{\frac{n+1}{2}} & n\,uneven \\ \frac{1}{2}\left(x_{\frac{n}{2}} + x_{\frac{n}{2}+1}\right) & n\,even \end{cases}$ is a numeric value, separating lower and higher half of an ordered sample set.
- The arithmetic average $MEAN = \frac{x_1+x_2+...+x_n}{n} = \mu$ is the sum of all sample values divided by the number of samples.
- The standard deviation is defined as $STD = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu^2)}$. It indicates that round about 68 percent of the values within the set can be found in the interval $[\mu - \sigma, \mu + \sigma]$. A small deviation indicates stable measures whereas a large deviation indicates unpredictable (random) behavior.
- A percentile $PCT(p) = \begin{cases} X_i + d\left(X_{i+1} - X_i\right) & for\, 0 < i < n \\ X_1 & for\, i = 0 \\ X_n & for\, i = n \end{cases}$ is an estimated value, that separates an ascending ordered set of measurements in a set of smaller and a set of greater values. The pth percentile is a value, $PCT(p)$, such that at most $(100p)\%$ of the measurements are less than this value and at most $100(1 - p)\%$ are greater. The 50th (P50, $p = 0.5$) percentile is called the median. Percentiles are estimated according to the description in [NIS-2003, Section 7.2.5.2. (EXCEL)]. At first $i + d = 1 + p(n - 1)$ with $0 < p < 100$ is computed. The result is a value, divided in an integer part i and a decimal part d. Now, PCT(p) can be computed as shown above.

Statistical measures are calculated after completion of the experiment. This is supported by implemented scripts and java classes. The implementation of the statistical measures $MEAN$ and STD can be found in file StandardDeviation.java (package trustedBittorrent.example.test).

Measures MIN, MAX, MED and $PCT(p)$ are calculated in OpenOffice [17] version 3.1. Floating point values can be computed with accuracy up to 15 digits.

Aggregated measures

An aggregated measure is the *service processing time* (SrvPT). It represents the time, the serving instance is occupied by processing the protocol (e.g. $procTime_1 + procTime_2 + procTime_3$ in figure 14 on page 64). Even the "waiting" times are included where the counter party is actively processing. Sometimes a response is sent and thereafter (during the waiting time) related tasks are computed until the next message is received. This measure excludes the network transmission time of messages. Statistical measures of aggregated measures are calculated as described above except that each value of the set is the sum of the aggregated processing times $(x_i = \sum_{j=1}^{z}(procTime_j)$.

4.4 Experiments

The experiments reflect basic scenarios derived from the BitTorrent protocols and the extensions. Scenarios are excluded from measurement if no security relevant information is exchanged or processed. That applies for all *Peer Protocol* (PP) messages of the PWP and following events (completed, stopped, empty) of the TP. Their encryption is the only difference to the original protocols. If necessary, it seems to be sufficient to estimate these delays based on reference values. Experiments within this chapter are based on the application described in chapter 3. Some experiments are repeated in chapter 5 to evaluate the results of an optimized application or protocol. Experiments are conducted completely automated and the result evaluation is performed mostly automated. All scripts and detailed results can be found as described in the appendix A.

4.4.1 Application Scenario

The extended protocol will be applied in a commercial platform for content distribution. In context of this thesis a business model with three involved parties is assumed, an *internet service provider* (ISP), many *content providers* (CP) and many customers.

- The ISP is responsible for the production, distribution and maintenance of platforms. Additionally the ISP may serve as the *certification authority* (CA) providing digital signatures for the TPM based platforms of the network participants.

[17] A reference for the definition of the calculation of these measures could not be found. However, manual calculation according to the definition above showed equal results.

- The CPs introduce content to their distribution network based on the underlying network of platforms. They operate the trackers. In case of a central tracker approach, the trackers are located at the CP site. For each content, described by the *metafile*, a more or less dynamic *overlaying network* (swarm) can be established. The CPs operate a file server (e.g. an HTTP web server) providing the metafiles to customers. Additionally, the CPs operate a customer's platform. On that platform the complete content is stored. Customers can connect to and download from that platform using the extended peer-wire protocol.

- Customers can participate in this content distribution. With the provided Peer-to-Peer based platform they can consume content, store it or even share it with other customers. But in this business model, it is assumed that active sharing and storing is managed by the content provider.

Strong mechanisms are required to ensure authenticity, authorization and integrity of the participating platforms, since the platforms are located at the customer's site. Obviously, a higher vulnerability can be assumed in such an "uncontrolled" environment. Hardware based security mechanisms and the extended protocols are introduced as countermeasure. But introducing a higher level of security will typically lead to decreased performance, since additional tasks have to be performed for the same use case. The main question is: Will the loss of performance be in an acceptable range? In order to answer this question, it has to be evaluated where and how often the extended tasks have to be performed. A representative scenario is required.

Download Use Case

As already stated, the ISP is responsible for the maintenance of all platforms. A maintenance use case could be an upgrade of a software component. In this use case the ISP behaves as a CP for update images as well. However, the same use case would apply for a CP of e.g. multimedia content, except that the download would be initiated by the customer. In the current scenario, the ISP management instructs a platform to download an update image using the BitTorrent application. Therefore the ISP provides a *unified resource locator* (URL) referring to a *metafile* at the CP's file server. This metafile contains the address of the tracker platform and information referring to the update image.

As shown in figure 15 on the following page the CP operates the *peer* (pB), a central tracker and a file server. The complete update image is stored at pB and is offered for downloading. The tracker keeps tracking swarms and the file server offers the metafiles. Peer pB must already be connected to the tracker by the tracker protocol. Furthermore the ISP has ensured that tracker and peer platforms are equipped with proper public-key certificates. This may be done during the creation or delivery process of platforms.

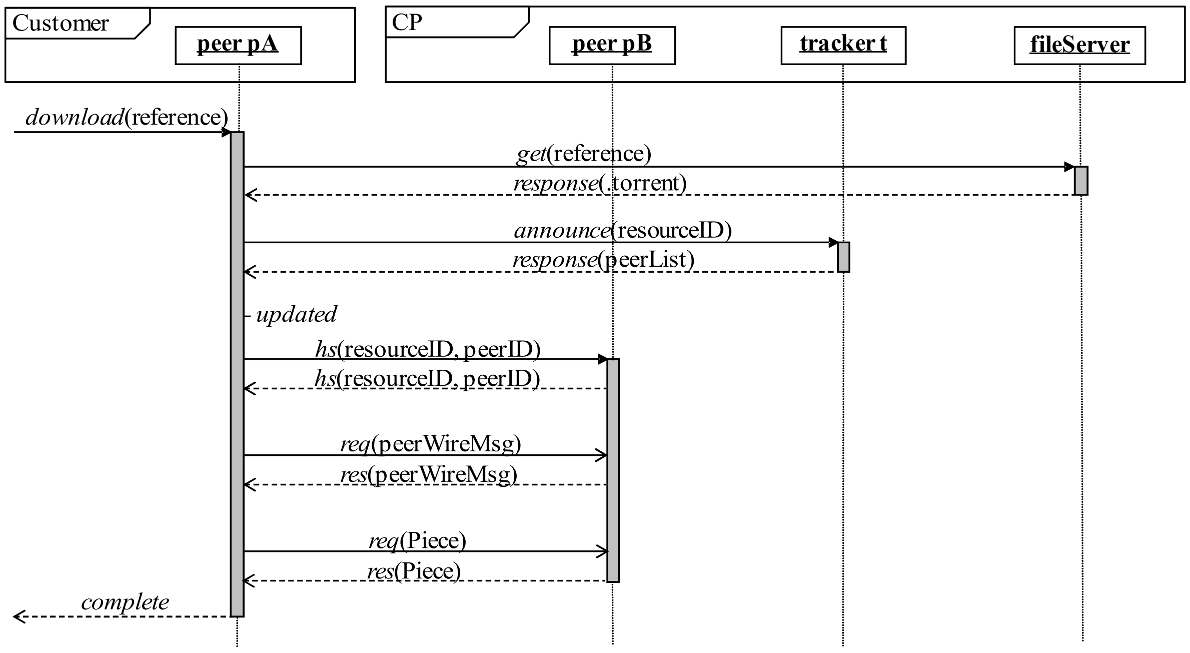


Figure 15: Download use case (functional view without extension).

In order to download the update image via BitTorrent, the customer's platform *peer* (pA) has to inquire the metafile from the CP's file server. This metafile contains the address of the tracker platform and information referring to the update image (e.g. resourceID). Peer pA transmits the responseID of the update image to the tracker via the TP. The tracker responds with a list of peers sharing the requested image. In this scenario the list contains pB only.

After pA has updated the local list of peers, pB is contacted via the PWP. With the initial handshake it is confirmed, that the intended resource from the intended peer is requested. According to the PWP, both peers coordinate the download by asynchronous messages. One or more pieces of the update image are exchanged until pA has received the complete image.

Occurrence of Extensions

For the communication between peer and tracker as well as between peers, extended protocols are applied. They are subject to measurement. Communication to other instances, like the file server, is not in focus of measurement. Both protocol extensions are embedded in the first two steps of the original protocols. As a result, the extended protocols accomplish original and extended tasks at once.

Tracker Protocol. With the tracker protocol, a list of peers is transmitted to pA. After completion of the extended tracker protocol the following tasks are accomplished additionally:

- Mutual authentication of both parties.
- Integrity (software) of the peer platform is validated.
- An encrypted communication is established (shared secret exchanged).
- A download authorization token (called ticket) is exchanged.

The extended-tracker protocol has to be performed, at least, once for each peer/tracker relationship. That would be once for each peer and for the tracker as often as the number of participating peers. When such a relationship was initially established, not all of the tasks have to be performed in subsequent requests for the same or another resource. In fact, only the creation of the tickets will cause additional processing time. However, the validity of each accomplished task should be limited to proper intervals.

Peer-Wire Protocol. When peer pA has received a list of peers sharing the same image, a connection to peer (pB) can be established via the PWP is performed. After completion of the extended PWP the following tasks are accomplished additionally:

- Mutual authentication of both parties.
- The download authorization token (called ticket) is verified.
- An encrypted communication is established (shared secret exchanged).
- Download of the image.

The extended PWP has to be performed, at least, once for each peer/peer relationship. For peer pA, that would be the number of provided peers from the tracker. For the contacted peers, it would be only once in this scenario. It might be many times, if the update applies to many peers of the distribution. When such a relationship was initially established, not all of the tasks have to be performed in subsequent requests again. For authentication and encrypted communication the shared secret can be reused. Only the verification of the ticket will cause additional processing time.

4.4.2 Download Experiment Scenario

Purpose. The performance of the extended protocols and the original protocols shall be measured and compared. The conducted experiment scenario mainly represents the download use case. It is assumed to be representative for an operative application.

Preconditions. Within an operative environment two peers (pA, pB), one tracker (t), one file server and one CA are required. However, the web server and the CA are not subject to measurement and therefore not set up as separate instances respectively network nodes (see appendix A.1). Instead, a metafile `nada_customer_02_app.torrent` is already created and distributed to pA and pB before experiment execution. It refers to the tracker and the image `nada_customer_02_app.img` stored at pB.

Configuration. Apart from the definition of the experiment scenario the configuration of the application has a major influence to the results of the experiments. If not stated different, the following configuration is applied:

- pA SML=4 (for details see the respective experiment referred in appendix)
- pA KHL=3 (for details see the respective experiment referred in appendix)
- Cryptographic algorithms/schemes, (see description in chapter 3)

Description. During the experiment, peer pB (CP) performs the extended TP with the tracker. Before pA performs the TP, pA and the tracker generate and exchange the keys S^t_{pub} and $AIKCert^p_{ca}$ (see protocol 4 on page 43). In practice, this might be done by the CA of the ISP. These steps are not part of the protocol itself, but necessary preprocessing for each platform. Within experiment logfiles and reports, the steps are referred to as step 0 and step -1 (see appendix A.2).

Actions. There is no interaction defined for this experiment. All activities are defined in the experiment description and the configuration of the applications. The experimenter only has to start the experiment. It will automatically be finished after the scenario is completed.

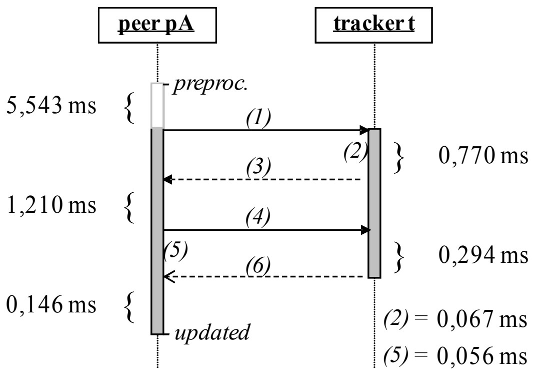
4.5 Results

On the following pages the results of two experiment sets A.4.2(appendix) and A.4.3(appendix) are presented. Each is based on 100 experiments. However, these are not the final results. They rather provide a basis or benchmark for further analysis and optimization, described in the next chapter. For all experiments the corresponding software version (revision) is provided. All experiment details (data sets, logs, calculations, scripts, reports, etc.) can be found as described in appendix C.2. For both protocols, an initial protocol execution and a subsequent protocol execution from the same entity is considered. Finally, some measures are analysed further and some parameters that significantly influence performance measures are discussed.

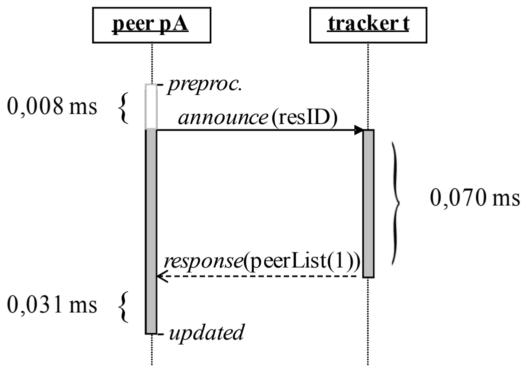
4.5.1 Initial `announce-started` for a resource (Tracker Protocol)

With the initial `announce` (event `started`) a peer pA announces to the tracker, that he wants to become member of a swarm. Therefore the resourceID is transmitted. In response he receives a list of swarm members. After completion of the trustworthy protocol the following tasks are accomplished: authentication, attestation, encrypted connection and authorization (by token).

In figure 16 the results of two experiments shown. Figure (a) depicts the result of the extended BitTorrent application whereas figure (b) depicts the results of the original BitTorrent application. The measures are computed as previously described. They are displayed in *milliseconds* (ms)[18].



(a) tBittorrent Tracker-Protocol



(b) jBittorrent Tracker-Protocol

Figure 16: Comparing initial `announce-started` for a resource (MEAN, $1s = 1,000ms$).

(a) tBitTorrent

The *extended tracker protocol* (tBitTorrent) is performed between pA and the tracker. It requires (see protocol 4 on page 43) four transmissions $\{(1), (3), (4), (6)\}$ and two processing steps $\{(2), (5)\}$. In context of this scenario, pA completes the protocol at measurement point *updated.* In table 5 on the following page the results for each measure are shown.

[18]A millisecond is a thousandth of a second ($10^{-3}s$). One second will be represented in milliseconds as $1,000\,ms$. Thousands are separated by a colon.

mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(1)*	1,646	5,543	17,786	4,517	3,558	10,372
RECV(1)	*SEND(3)*	0,596	0,770	0,935	0,780	0,073	0,853
RECV(3)	*SEND(4)*	1,042	1,210	2,282	1,189	0,154	1,321
RECV(4)	*SEND(6)*	0,198	0,294	0,441	0,287	0,050	0,366
RECV(6)	*updated*	0,131	0,146	0,226	0,137	0,019	0,174
BEG(2)	*END(2)*	0,038	0,067	0,112	0,074	0,020	0,090
BEG(5)	*END(5)*	0,049	0,056	0,104	0,056	0,007	0,063

Table 5: (a) Tracker protocol results of tBitTorrent ($1s = 1,000ms$).

(b) jBitTorrent

The *original tracker protocol* (jBitTorrent) is performed between pA and the tracker. It requires two transmissions (*announce* and *response*). In context of this scenario, pA completes the protocol at measurement point *updated.* In table 6 the results for each measure are shown.

mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(ann)*	0,007	0,008	0,031	0,007	0,003	0,008
RECV(ann)	*SEND(res)*	0,057	0,070	0,108	0,070	0,007	0,081
RECV(res)	*updated*	0,026	0,031	0,039	0,032	0,002	0,035

Table 6: (b) Tracker protocol results of jBitTorrent ($1s = 1,000ms$).

Comparing (a) and (b)

Both experiments can be compared, if the measures for peer pA and tracker are aggregated as described in section 4.3.2.Table 7 shows the aggregated measures.

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	4,055	7,965	20,138	6,956	3,524	12,769
SrvPT *t*	tBitTorrent	1,992	2,275	3,315	2,261	0,155	2,427
SrvPT *pA*	jBitTorrent	0,091	0,110	0,143	0,109	0,009	0,123
SrvPT *t*	jBitTorrent	0,057	0,070	0,108	0,070	0,007	0,081

Table 7: Comparing (a) and (b) tracker protocol performance ($1s = 1,000ms$).

Interpretation

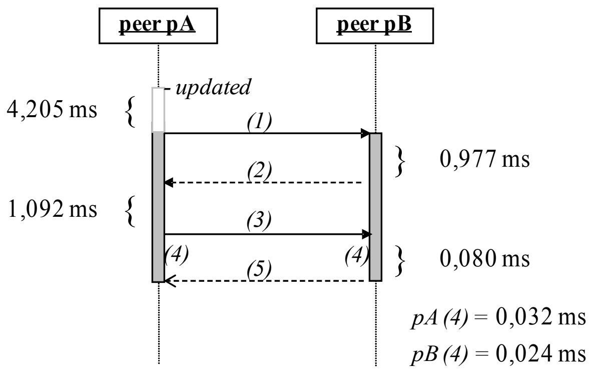
The comparison of jBitTorrent and tBitTorrent clearly shows a significant rise of processing time that is caused by the extension of both protocols. Peer pA (tBitTorrent) completes the protocol after nearly 8 seconds (MEAN) in which the tracker (tBitTorrent) takes more than 2 seconds (MEAN). Most of the processing time of *SrvPT pA* occurs during preprocessing of the protocol $(preproc, SEND(1))$. For this measure a strong deviation of more than 3 seconds (STD) is observed. It is caused by a cryptographic library (see further analysis). However, preprocessing time doesn't seem to be a critical measure, since it may be done at any time in advance of the protocol execution. Other measures provide an acceptable deviation, even if some exceptional values are measured (e.g. MAX of $(RECV(3), SEND(4))$). The key measure of the tracker protocol is *SrvPT t,* since many peers are expected to contact a single tracker and therefore it has to be executed often.

4.5.2 Subsequent `announce-started` for a resource (Tracker Protocol)

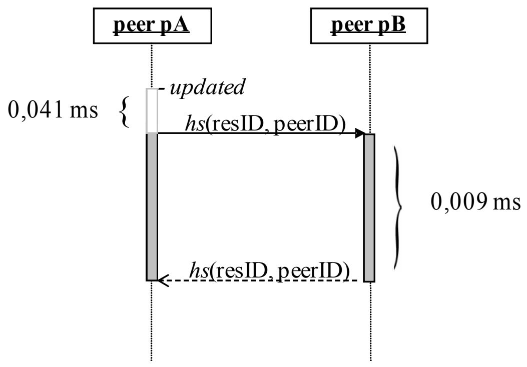
With the subsequent `announce` (event `started`) a peer announces to the tracker, that he wants to become member of an additional swarm. Since the trustworthy protocol was already performed once before (initial for another resource), it is not necessary to perform the related protocol steps again. The request should be encrypted with the shared key. This request is neither specified, nor implemented or evaluated. Since only tickets have to be created it can be assumed, that the processing time will correlate to the corresponding time of the initial request 4.5.1.

4.5.3 Initial `handshake` for a resource (Peer-Wire Protocol)

With the initial `handshake` a peer requests to share a resource with a certain peer. The extension accomplishes the tasks: authentication, authorization, secure connection.



(a) tBittorrent Peer-Wire-Protocol



(b) jBittorrent Peer-Wire-Protocol

Figure 17: Comparing initial `handshake` for a resource (MEAN, $1s = 1,000ms$).

(a) tBitTorrent

The *extended peer-wire protocol* (tBitTorrent) is performed between pA and pB. It requires (see protocol 4 on page 43) four transmissions $\{(1),(2),(3),(5)\}$ and two processing steps $\{pA\,(5)\,,pB\,(5)\}$. In context of this scenario, pA completes the protocol when the handshake is received. Thereafter both parties communicate asynchronous. The decryption of the last message is excluded from measuring. In table 8 the results for each measure are shown.

mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(1)*	0,419	4,205	23,834	3,086	3,726	7,408
RECV(1)	*SEND(2)*	0,946	0,977	1,020	0,975	0,014	0,997
RECV(2)	*SEND(3)*	0,974	1,092	1,897	1,059	0,157	1,142
RECV(3)	*SEND(5)*	0,070	0,080	0,130	0,077	0,010	0,091
BEGpA(4)	*ENDpA(4)*	0,020	0,032	0,063	0,032	0,004	0,037
BEGpB(4)	*ENDpB(4)*	0,023	0,024	0,047	0,024	0,003	0,025

Table 8: (a) Peer-Wire protocol results of tBitTorrent ($1s = 1,000ms$).

(b) jBitTorrent

The *original tracker protocol* (jBitTorrent) is performed between pA and pB. It requires two transmissions (*hs-request* and *hs-response*). In context of this scenario, pA completes the protocol when the handshake is received. In table 9 the results for each measure are shown.

mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(hs)*	0,021	0,041	0,080	0,040	0,009	0,050
RECV(hs)	*SEND(hs)*	0,005	0,009	0,025	0,007	0,004	0,015

Table 9: (b) Peer-Wire protocol results of jBitTorrent ($1s = 1,000ms$).

Comparing (a) and (b)

Both experiments can be compared, if the measures for peer pA and tracker are aggregated as described in section 4.3.2. Table 10 on the following page shows the aggregated measures.

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,525	6,356	25,900	5,257	3,727	9,640
SrvPT *pB*	tBitTorrent	2,015	2,150	2,942	2,118	0,154	2,209
SrvPT *pA*	jBitTorrent	0,031	0,050	0,097	0,050	0,011	0,062
SrvPT *pB*	jBitTorrent	0,005	0,009	0,025	0,007	0,004	0,015

Table 10: Comparing (a) and (b) peer-wire protocol performance ($1s = 1,000ms$).

Interpretation

All measures of the extended application (tBitTorrent) are with less performance than the original application (jBitTorrent). This is caused by the security extensions of the PWP. A strong deviation (STD) of more than 3 seconds can be observed for the measure SrvPT *pA*. It occurs during preprocessing of the protocol $(updated, SEND(1))$ and it is caused by the same cryptographic library that is used during tracker protocol execution (see further analysis). Some exceptional values occur (e.g. MAX of $(RECV(2), SEND(3))$), however the standard deviation (STD) and the 90% percentile (P90) indicate stable resp. predictable measures. The SrvPT of peer *pB* is considered as the key measure for the PWP since the calling peer can't complete its tasks until *pB* is finished. However, it can't be stated that one peer *pB* must be frequented very often. That is characteristically for the Peer-to-Peer model and contrary to the client-server model.

4.5.4 Subsequent Communication for a resource (Peer-Wire Protocol)

Once the initial handshake of the extended PWP is performed, two peers have established a protected communication channel. In the original and the extended version the same messages are exchanged, then. Information about available pieces of a resource (bitfields) is synchronized and pieces are exchanged. This is still the typical BitTorrent processing as specified in [COH-2008] and implemented in jBitTorrent. Additional workload can be expected just for encryption and decryption. These operations vary with the applied algorithms of each provider and key lengths. The configuration is described in table 4 on page 57 and appendix B. The processing time is expected to be similar to other well known and measured cryptographic applications. Hence, no additional effort is spent on measuring these operations. An example performance analysis of data encryption algorithms can be found in [TAM-2008].

4.5.5 Further Analysis

Diffie-Hellman parameter generation

Measures of both protocols indicate a strong deviation during preprocessing. It is assumed that a single cryptographic library causes them. During preprocessing of both protocols DH parameters are generated in order to exchange freshly generated keys. Parameter generation is realized by the generator `DHParameterGenerator` from the library `iaik.security.dh` (version 3.16) of the provider IAIK. The processing time of this generator is measured as well. Therefore, the previously described experiment (A.4.3) already includes the two additional measurement points $DHGen(B)$ and $DHGen(E)$. The measured code is a single line:

```
AlgorithmParameters params = paramGen.generateParameters();
```

Both measurement points are located between *preproc.* and *SEND(1)* of the tracker protocol and between *updated* and *SEND(1)* of the peer-wire protocol. The results are shown in table 11.

Tracker Protocol:

mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(1)*	1,646	5,543	17,786	4,517	3,558	10,372
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
DHGen(B)	*DHGen(E)*	0,344	4,223	16,428	3,238	3,564	9,052

Peer-Wire Protocol:

mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(1)*	0,419	4,205	23,834	3,086	3,726	7,408
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
DHGen(B)	*DHGen(E)*	0,342	4,127	23,769	3,018	3,731	7,338

Table 11: Measuring the Diffie-Hellman parameter generator ($1s = 1,000ms$).

Now the results of both measures can be compared for each protocol. Similar values of the MEAN and the STD confirm that the observed deviation in both protocols is caused by the measured generator.

Swarm size

Previously described experiments are based on a scenario with only two participating peers. But in scenarios, close to operational environments, much more peers are expected to share one resource. Such a large swarm would affect the processing time of the tracker protocol. Especially, the processing time SrvPT of the tracker *t* would increase during creation of message

(6). This is due to the fact that, according to the swarm size, many tickets have to be created. As a result, the peer must wait longer until the tracker has completed its task. During protocol post processing, the peer may contact many peers what obviously consumes more time. But the amount of peers, that have to be contacted at once, is subject to the peer configuration.

In order to quantify the elapsed time during ticket creation two additional measurement points are introduced. $TCrea(B)$ indicates the beginning of the ticket creation for a given list of peers (currently 1) and $TCrea(E)$ indicates its end. Conducting the previously described experiment (see A.4.3) leads to the following results:

mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *t*	tBitTorrent	1,992	2,275	3,315	2,261	0,155	2,427
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
TCrea(B)	*TCrea(E)*	0,025	0,043	0,094	0,043	0,012	0,061

Table 12: Measuring the ticket creation time ($1s = 1,000ms$).

The values shown in table 12 describe the processing time for a list of peers with size one (1 ticket). If a linear development is assumed for an increasing list of peers, a significant influence to the processing time can be expected even for small sizes. A list of peers with a size of 10 would be expected to be processed in more than 400ms (MEAN). That becomes significant in relation to the SrvPT *t* with a MEAN of 2,275ms for a peer list of one. Therefore, a deeper analysis of the current code and additional effort for optimization is recommended.

SML&KHL size

The previously described experiment scenario is based on a certain SML (size 4) and KHL (size 3) configuration. It is expected, that the size of both lists significantly influences the processing time of both participants of the tracker protocol. Both lists are used during attestation of a peer. Therefore a peer transmits its SML to a tracker where it is validated. Each entry of the SML (name and hash) is validated against a list of known (valid) values. Additionally, an expected PCR value is calculated that can be compared to the PCR of the peer, thereafter. Following measures are affected:

- Peer measure $(preproc, SEND(1))$, loading and serialization.
- Tracker measure $(RECV(1), SEND(3))$, deserialization (and storing).
- Tracker measure $(RECV(4), SEND(6))$, validation.

The size and content of an SML varies for each system as well as the content and size of the KHL. As an approach to an operational environment it is expected, that the KHL necessarily contains, at least, as much entries as the SML. Otherwise, the SML would always contain unknown entries (assuming that each SML value is unique). Example SMLs from the IBM *Integrity Measurement Architecture* (IMA) [SAI-2006](ssd_ima.measurements.html) are used to estimate an SML of an operational environment. The referred example installation of this IMA system produces 68 SML entries previous to the kernel and 349 entries during startup of the operating system. Together, an SML of about 400 entries and necessarily a minimal KHL of 400 entries are expected to come close to an operational environment.

With this configuration an additional experiment was conducted (version R53, see A.4.6). In order to measure the processing time of the SML validation, two additional measures were introduced $(SMLVal(B), SMLVal(E))$ between the measurement points $RECV(4)$ and $SEND(6)$ of the TP. The measured value of 5,810ms (MEAN) is greater than the previously measured SrvTP of the tracker. Therefore, additional effort is to be spent in optimization. Further analysis and optimization is described in the next chapter.

mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SMLVal(B)	*SMLVal(E)*	5,674	5,810	6,051	5,809	0,074	5,913

Table 13: Measuring the SML validation time with KHL&SML size of 400 ($1s = 1,000ms$).

4.6 Conclusion

Within this chapter an evaluation of the previously enhanced application is presented with the main intension to quantify its performance. Therefore, relevant scenarios are identified and discussed. A controlled physical environment (laboratory) for measurements is described, as well as an appropriate measurement technique. The process of measuring is completely automated and the process of data evaluation is automated except of some report formatting tasks. Hence, each experiment is reproducible and traceable. Measures are defined for each protocol step, each communication party, selected algorithms and external components. Statistical measures are introduced to increase the informative value and the quality of the presented results. From these results it can be concluded that:

- The enhancements cause a significant increase of processing time (from milliseconds to few seconds) compared to the original protocol implementation.
- Most of the processing time is located in uncritical (preprocessing) areas.
- Areas of low performance exist. They are subject to further improvement.

- An external library is causing strong deviation, however in areas that are expected to be uncritical (preprocessing).
- Apart from the environment and the integrated software components (e.g. crypto algorithms), the results depend on the variables swarm size, KHL and SML.

5 Performance Optimization

"The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet - that is, not until you have a perfectly clear and unoptimized solution." - Michael A. Jackson [JAC-1975, Preface vii]

This chapter focuses on perfomance optimization of the implemented and evaluated *Peer-Wire Protocol* (PWP) and *Tracker Protocol*. If not already done in the previous chapter, concepts for optimization are introduced, implemented and measured as far as it is possible within this thesis. The resulting application will not be optimal, but it should serve as an indicator that allows to approximate an "optimal" operative application. However, in future development different requirements may be evaluated for different business models resulting in different "optimal" applications. Furthermore, it should confirm or disprove the protocol design decisions. As suggested by Jackson [JAC-1975, Preface vii], the optimization will be limited and based on an unoptimized and even evaluated version from the previous chapters.

5.1 Approach and Focus

Each optimization intension must focus on at least one key aspect that is subject to optimization. Typical aspects are resource (memory, diskspace, bandwidth, power, etc.) consumption, resource usage, processing time, security or others. This chapter focuses on performance respectively the processing time of the protocol implementation. It has to be noted, that usually the optimization of one aspect will require a trade-off with another aspect in return. Therefore, an application that is optimal in only one aspect is rather unrealistic. Typical trade-offs for an optimized performance might be: increasing code, loss of code- and design comprehensibility, increasing resource consumption, decreasing maintainability, and others. On the contrary this thesis provides evidence for the performance loss in exchange to an increased security level.

Once the focus is fixed on one aspect like processing time and trade-offs to other aspects are accepted, it has to be decided where to optimize. In general, the processing time depends more or less on every involved hard and software component as well as on involved remote (network communication) components. Furthermore, for software components the compiling process and the runtime environment (e.g. JVM) have to be taken into account. Therefore, a valid approach would be to systematically analyse each involved component and suggest appropriate optimizations. However, this could not be accomplished within this thesis. Therefore, the decision towards performance optimization are based on peculiar measurement results from the previous chapter. Futhermore, the focus is narrowed to "new", extended functionality.

Optimization Process

The evaluation technique presented in the previous chapter is reused for the optimization of the application and the protocols. It serves as a definition of experiments and benchmark for further optimization. Figure 18 depicts the applied process cycle.

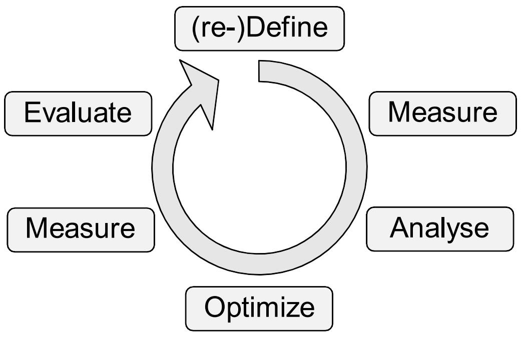


Figure 18: Optimization Process.

- (re)-Define: Initially some definitions have to be created. Measures have to be defined as well as the application configuration, one or more scenarios and a detailed description of the environment. This is initially done within the previous chapter.
- Measure: Based on the definitions, experiments are conducted. The results of the current version serve as benchmarks for an enhanced version of the application.
- Analyse: The collected measures may now be analysed with respect to the intended optimization (here processing time). Areas of low performance (e.g. hot spots/bottle necks) may be identified.
- Optimization: The insight from the analysis is used to develop concepts and designs for optimization. They are implemented and a new version of the application is created.
- Measure: Followed by the optimization, the amended application resp. protocol is measured again.
- Evaluation: Results of the second measurement can be compared with the initially measured benchmarks. It might present a gain or loss in performance and reveil trade-offs. Further decisions are based upon that comparison. Modifications can be accepted or rejected.

With the evaluation one cycle is completed. Now, it might be necessary to redefine some assumptions, measures, configuration etc. In that case an additional measurement has to follow. If definitions are kept, the following steps are conducted again. That is repeated until an acceptable result is achieved.

5.2 Application Optimization

5.2.1 Serialization

As described in section 3.3.1, messages of the tracker protocol are serialized and thereafter, transmitted over HTTP. During the evaluation of the communication framework it was observed, that the introduced (previous to the thesis) serialization mechanism has a rather poor performance. Further analysis showed, that each message is serialized to a large XML file. During message creation of each step, the serialized message is written to the file system. Even worse, before sending it must be load from file system. It is expected that optimization of these routines would siginificantly increase the performance of peer and tracker.

Solution

Messages are represented as serializable objects. Before transmission they are serialized to a byte array using the `java.io ObjectOutputStream` and, thereafter, the byte array is converted to a hex string. Thereby, each byte is converted to two characters (e.g. byte '1' → String '01' and byte '-1' → String 'ff'). Only the values *0-1* and *a-f* are valid. The hex string representation prevents interpretation failures in lower layers. These strings are not written to the filesystem anymore. Consequently, they don't have to be loaded from filesystem. Instead they are kept in memory until the message is sent. The recipient of the transmission converts the hex string back to a byte array. As a counterpart to the `ObjectOutputStream`, the `java.io ObjectOutputStream` converts the byte array back to an object that is casted to the original message class.

Evaluation

Another experiment, equal to the one described in chapter 4.4, was conducted in order to measure the effect of the optimization (see A.4.3). The results of that application (revision 50), containing optimized code, is compared to the results of experiment A.4.3 with application (revision 48). Nearly all measures, depicted in table 14 on the next page, indicate a siginificant performance gain. Just the MAX and STD of ServTP *pA* show less performance. However, this seems to be due to few exceptional experiments.

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	R48(TP)	4,055	7,965	20,138	6,956	3,524	12,769
SrvPT *t*	R48(TP)	1,992	2,275	3,315	2,261	0,155	2,427
SrvPT *pA*	R50(TP)	2,534	6,563	33,763	5,401	4,695	10,263
SrvPT *t*	R50(TP)	1,484	1,764	2,237	1,755	0,132	1,949

Table 14: Comparing original with optimized serialization ($1s = 1,000ms$).

5.2.2 SML Validation

As described in section 4.5.5, SMLs and KHLs with sizes that can be expected in operational environments, cause an unacceptable processing time of the tracker protocol. It is assumed that an optimization of the SML validation would significantly decrease processing time of the tracker measure $(RECV(4), SEND(6))$ for large SML/KHL sizes. Furthermore the validation may be computed in parallel to the protocol processing. A similar approach is described in section 5.3.2 “Concurrent Computing of the Shared Secret”.

Algorithm optimization

The implemented SML validation algorithm is derived from the project Ethemba [BRE-2008]. Mainly it validates that the hash and the name of each value in the SML matches the hash and the name of a KHL entry. Furthermore, a PCR is calculated on the basis of the hashes within the SML. That value is compared to the PCR received in message (4) after SML validation. During analysis of the relevant code it could be identified that unnecessary operations are performed for the validation of each SML entry:

- KHL is loaded from file system.
- 2nd search operation over KHL is performed (1st for hash, 2nd for name).

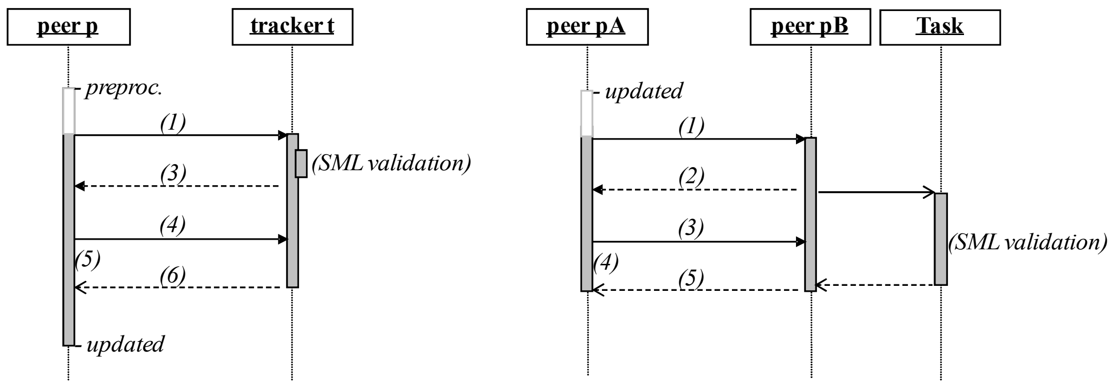
As an optimization to the validation, these unnecessary operations were removed. In order to measure the difference, two measurement points $(SMLVal(B), SMLVal(E))$ are introduced and the experiment described in the previous chapter was executed again. The SML is configured to 400 values and the KHL is configured to 400 values, too. Now the previous version R53 (A.4.6) and the optimized version R54 (A.4.7) can be compared. Results are shown in table 15 on the following page. They clearly point out, that the optimized SML validation is with higher performance. Same performance gain can be observed for the corresponding SrvPT measures.

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SMLVal	R53(TP)	5,674	5,810	6,051	5,809	0,074	5,913
SMLVal	R54(TP)	0,095	0,124	0,153	0,126	0,010	0,133

Table 15: Measuring the SML validation time ($1s = 1,000ms$).

Concurrent computing

The enhanced tracker protocol defines to transmit the SML with the first message (1) to the tracker. Subsequently the SML must be validated. However, the results of this validation are not needed until processing of message (4). This is similar to the situation described in section 5.3.2. That implemented optimization is based on concurrent computing of the corresponding task. According to that solution, concurrent computing of the SML validation is applied as an optimization here. The task is submitted at the end of message (1) processing and the result is accessed (synchronization) at the end of message (3) processing (see figure 19). It is performed by the callable `SMLValidator` class, now.



(a) Previous processing (b) Optimized processing

Figure 19: Sequence diagram concurrent SML validation.

This solution is realized in application version R55. In order to measure its performance, a set of experiments is executed according to the description in chapter 4. It can be compared to the previous version as shown in table 16 on the following page. Both experiments are configured with a SML of size 400 and a KHL of size 400. As expected, the SrvPT *t* is performed faster because SML validation is processed during the time the tracker is waiting for message (4). However, this gain of performance can be expected to decrease with an increasing amount of simultaneous requests from different peers. If the operative system is equipped with multiple CPU's, the validation could already be executed with the recipience of message (1). Message processing and validation would be executed in paralled ("ideally") without competition.

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT t	R54(TP)	1,752	2,126	3,030	2,116	0,222	2,323
SrvPT t	R55(TP)	0,678	1,881	3,204	1,876	0,248	2,047

Table 16: Comparing concurrent SML validation ($1s = 1,000ms$).

Dependency from SML/KHL

As already stated, the processing time of the tracker protocol depends on the size of SML and KHL. In general, it depends on the SML and the validation mechanism applied. This protocol might be applied in an area where the SML is validated against a list of known malicious applications. For that reason each differing mechanism should be evaluated separately. Still, for the current mechanism it is of interest how the processing time will behave for an increasing size of the SML and for an increasing size of the KHL.

- SML: Initially the KHL is loaded. For each SML entry an amount of single operations on hash and program name strings (e.g. comparing, conversion, concatenation, ...) is performed. For each SML entry the appropriate entry in the KHL has also to be found. Independent of the applied search algorithm, it would result in a processing time that is linearly increasing with the size of the SML (fix KHL). Apart from validation the same applies for the serialization mechanism, that will take more processing time for an increasing SML.

- KHL: An increasing KHL would increase the processing time for each SML entry. Assuming that an efficient search algorithm is used (e.g. binary search) with a KHL that is sorted in advance, the complexity would be not more (worst case) than $O(log\, n)$ where n is the size of the KHL. The current application is based on search algorithms derived from `java.util Hashtable`.

Finally, according to the operative environment it might be possible to substitute the transmission and validation of the SML. That would be the case if the ISP can define a limited set of valid SMLs, since all of the delivered platforms can be expected to run similar or same software. Then it would be sufficient to transmit a hash value of the SML in message (1). If it matches a hash of a predefined SML, no validation and calculation of the PCR might be necessary. It can be done in advance for each defined SML.

5.3 Protocol Optimization

5.3.1 Replacing public key encryption

The Tracker Protocol defines the initial message as values that are encrypted by a symmetric key. The asymmetric public key encryption of the applied RSA crypto system is known to provide lower performance than typical symmetric key encryption systems. After the first decade of Public-Key cryptography research, Diffie [DIF-1988] stated, that the RSA system is running with lower performance (DES is thousend times faster) than DES and uses keys that are much larger (ten times larger) than DES. This is still valid until today, even if the performance gap has become smaller. The RSA lab states [RSA-2010, Chapter 3.1.2 How fast is the RSA algorithm?] that DES is generally 100 times faster than RSA, if it is implemented in software. Modern algorithms like AES are even faster than DES.

Solution

Diffie [DIF-1988] suggested to use hybrid crypto systems, where RSA is applied for key management and symmetric cryptography for encryption. This is nowadays a common approach which is applied for the Tracker Protocol, too. It is already implemented in chapter 3. The peer encrypts the message by a freshly generated symmetric key R. This symmetric key is encrypted by the tracker's public key S^t_{pub}. The resulting cyphertexts of both operations are transmitted to the tracker. At first the tracker must recover the encrypted symmetric key R using its private key S^t_{priv}. Now, the recovered symmetric key can be utilized to decrypt the message.

- $enc\left\{message\right\}_{S^t_{pub}}$ is replaced by $enc\left\{message\right\}_R$,$enc\left\{R\right\}_{S^t_{pub}}$ and subsequently
- $dec\left\{C_m\right\}_{S^t_{priv}}$ is replaced by $dec\left\{C_R\right\}_{S^t_{priv}}$, $dec\left\{C_m\right\}_R$.

Evaluation

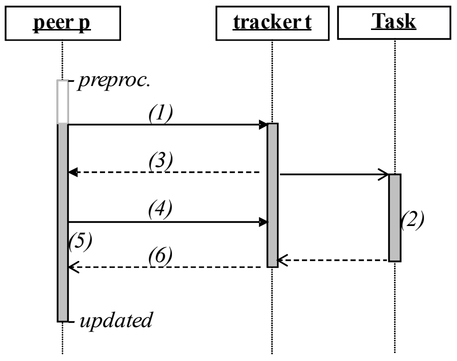
Measures are provided in all experiments starting from revision 48 (see appendix A.4.3). However, both variants are not compared. In general it can be stated that the advantage of the applied solution increases with the length of the message. Currently it is assumed, that the message at least includes one large value of variable length, the SML. This solution ensures computing with good performance even for increasing sizes of the SML.

5.3.2 Concurrent Computing (Shared Secret)

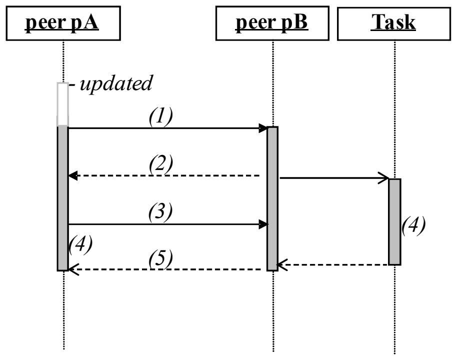
The extensions of the PWP, as well as the Tracker Protocol include tasks for the creation of a shared secret between both communication parties. These secret keys are not needed until the end of the protocol. Currently they are computed during processing of the response to a recent request. But it would bring more performance to compute them, in parallel , during the waiting time until the next incoming message arrives. Especially platforms with more than one core processors would profit from that optimization.

Concept

Such an optimization is applied to task (2) of the TP and peer pB's task (4) of the PWP (see sequence diagram 20). Task two is concurrently computed after the tracker completed sending of message (3). Tracker t and `Task` must be synchronized with the encryption of message (6) at the latest. But the `Task` might be completed earlier. The same concept applies for peer pB's `Task` (4) of the PWP. The tasks (5) of the TP and peer pA's `Task` (4) of PWP are computed after message sending anyway. Therefore a modification is not necessary for the current platform with one core processor.



(a) tBittorrent Tracker-Protocol



(b) tBittorrent Peer-Wire-Protocol

Figure 20: Sequence Diagram: Concurrent computing of a shared secret.

Design

The thread, representing the asynchronous task and the main thread are configured to run with the same priority, called normal priority (`Thread.NORM_PRIORITY`), which is a default of this Java environment. It implies that both threads compete for processing time of the underlying CPUs. Thus, it might result in better performance if the asynchronous task is prioritized lower. However, the meaning and consequences of the priority setting in Java is often referred to as vague and not reliable [ULL-2009, Chap. 11.3.10] . That is why no additional effort is spent in prioritization and its measurement so far. Instead, an asynchronous call is placed at the end of the current requests processing. It is assumed, that the asynchronous task will be performed during the time waiting for the next call or response.

The realized solution is depicted in the class diagram 21 on the next page. Basically two threads compete for processing time, a `FutureTask` and an `Object` as a representative of the appropriate tracker and peer class. With the `DHSecretKeyGenerator`, a callable class is introduced that provides the functionality of secret key generation in a `call` function. This function is required by the parametrized `Callable` interface that is bound to the type `SecretKey`. Each `Object` submits an instance of the `DHSecretKeyGenerator` generator and receives a `FutureTask` in response. Now the task is executed until completion. The result can be accessed by its parametrized `get` function that is bound to `SharedSecret`, too. If the result is not available yet, the calling thread is blocked until completion of the task.

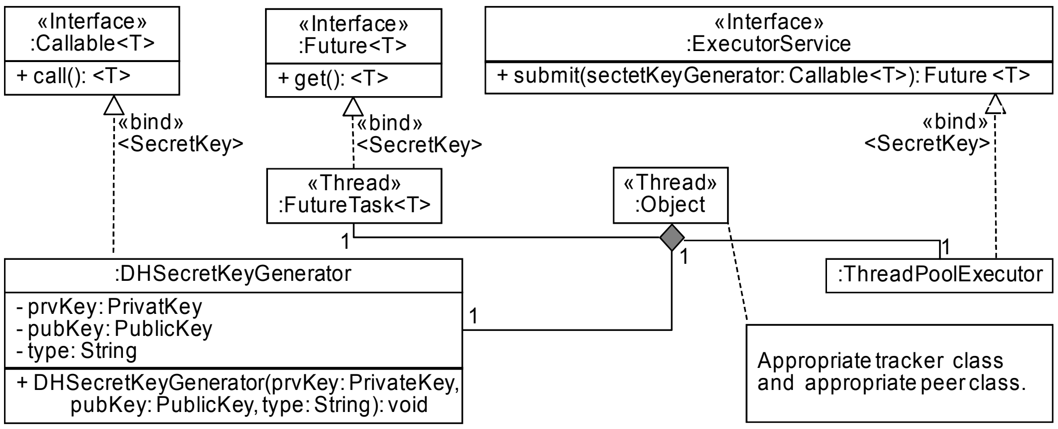


Figure 21: Class Diagram: Concurrent computing of a shared secret.

Evaluation

Complete measurement results of both protocols can be found in R51 (A.4.5). It can be compared to the experiment set of the previous application version R50 (A.4.4). R50 does not contain this optimization. The processing time of the shared secret computing is small, compared to the overall processing time. In revision 50 it took 0,069 ms (MEAN) to compute task (2) and 0,032 ms to compute peer pA's task (4). Hence, it is not surprising that the SrvPT of *t* an *pB* does not show much less processing time. The processing time of the tasks is within the standard deviation (STD) of the SrvPT.

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *t*	R50(TP)	1,484	1,764	2,237	1,755	0,132	1,949
SrvPT *pB*	R50(PWP)	2,014	2,144	2,898	2,125	0,130	2,230
SrvPT *t*	R51(TP)	1,481	1,737	2,581	1,719	0,185	1,863
SrvPT *pB*	R51(PWP)	2,009	2,167	2,978	2,126	0,164	2,243

Table 17: Comparing optimized secret creation to previous version ($1s = 1,000ms$).

Finally, it has to be mentioned that concurrent computing does not have to decrease the processing time if more than one peer is calling at a time. If there are many peers requesting to perform the protocol at once, then there might be no more waiting time.

5.3.3 Multiple Hash Attestation

According to Strumpf [STU-2008] and confirmed by the performance evaluation, the quote operation consumes a lot of the protocols processing time. As depicted in table 18, a quote operation takes about 400 ms in the mean within the current laboratory environment. Unfortunately, quotes are computed sequentially by the TPM. That would be crucial if many simultanous quotes arrived at a trusted computing system. Consequently, the processing time for a quote request becomes a multiple, according to the number of previously received and not completed quote requests. The described problem might occur in the considered field of business, too.

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
Quote(pA)	R55(TP)	0,386	0,453	0,551	0,414	0,064	0,544
Quote(pA)	R55(PWP)	0,368	0,384	0,414	0,384	0,009	0,394
Quote(pB)	R55(PWP)	0,370	0,413	0,558	0,394	0,049	0,536

Table 18: Measures of TPM quotes ($1s = 1,000ms$).

One of Stumpf's suggested solutions is the multiple hash attestation. The main idea is, that nonces of simultaneously received quote-requests are stored in a buffer. Then they are collectively computed by the TPM. That approach requires some extensions to the current protocols.

Tracker Protocol Design

One quote operation is defined by the tracker protocol. It is performed by the calling peer. Unless the peer doesn't need to request many resources at once, this quote operation seems to be uncritical. For the current business model, a peer is expected to perform the tracker protocol only once or twice at a time.

Peer-Wire Protocol Design

It is very likely, that in some scenarios many peers connnect to a single peer at once. E.g. if the single peer shares a rare resource. This protocol has to be performed for each resource, since each permission (ticket) must be verified in a new session. If multiple peers request one resource from a single peer, the single peer has to perform the protocol multiple times at once. Each calling peer has to process the protocol only once.

A solution is the extension of the PWP (see protocol 6 on page 48). Instead of a single nonce, multiple nonces are quoted. Protocol 7 shows the suggested extension. Both quotes of the protocol are treated in the same way. Though the described problem might not occur for the second quote, a consistent processing is preferable. Messages (2) and (3) of the PWP have to be amended and multiple nonces have to be collected in additional steps 2 and 4.

Protocol 7 PWP with Multiple Hash Attestation (extension only)

2. PWP: Peer pB stores nonces.

$$pB: \quad nList_{pB} := \left\{ hash(K_{pub}^{pA} || K_{pub}^{pB}) \,||\, hash(K_{pub}^{pI} || K_{pub}^{pB}) \,||\, hash(K_{pub}^{pN} || K_{pub}^{pB}) \right\}$$

3. PWP: Peer pB sends a response.

$$pB \rightarrow pA: \quad K_{pub}^{pB}, quote\,\{hash(nList_{pB}), PCR_{none}\}_{AIK_{priv}^{pB}}, nList_{pB} \tag{2}$$

4. PWP: Peer pA stores nonces.

$$pA: \quad nList_{pA} := \left\{ hash(K_{pub}^{pB} || K_{pub}^{pA}) \,||\, hash(K_{pub}^{pI} || K_{pub}^{pA}) \,||\, hash(K_{pub}^{pN} || K_{pub}^{pA}) \right\}$$

5. PWP: Peer pA sends a request.

$$pA \rightarrow pB: \quad quote\,\{hash(nList_{pA}), PCR_{none}\}_{AIK_{priv}^{pA}}, nList_{pA} \tag{3}$$

(2) Previously to this step, peer pB has received requests from several peers $\{pA, pI, pN\}$ for one resource at once. For each peer, a nonce is computed according to the PWP. Nonces are stored in a local nonce list $nList_{pB}$ before they are quoted. An appropriate storage interval has to be provided. It might either be realized as a tri-state ring buffer as described by Stumpf [STU-2008] or by the TPM itself (e.g. synchronized by a semaphor). The nonce list is a variable, concatenated string of nonces.

(3a) In this step, peer pB computes the quote. The hash value of the nonce list $nList_{pB}$ is provided as a nonce to the quoting operation of the TPM. Peer pB sends its public key K_{pub}^{pB}, the quote and the nonce list to peer pA.

(3b) Peer pA receives the response. The quote has to be verified as follows: At first peer pA calculates the expected nonce $hash(K_{pub}^{pA}||K_{pub}^{pB})$. Then peer pA has to verify, that the expected nonce is included in the nonce list $nList_{pB}$. If that is true, peer pA performes a quote validation according to the PWP.

(4&5) Steps 4 and 5 are processed in the same manner as previously described. Now, peer pA keeps a list of nonces and peer pB performs the verification.

Evaluation

An evaluation could not be performed, yet. It is expected, that the processing time for a single execution of the protocol is slightly increased. This is due to the extensions. On the other hand, the described bottleneck is removed. Results are expected to be similar to the evaluation of Strumpf [STU-2008].

5.3.4 Passive Attestation

The Multiple Hash Attestation (see section 5.3.3 on page 89) is an adequate solution for the performance bottleneck that is caused by the sequential processing of TPM quote operations. Strumpf [STU-2008] suggests two additional solutions, “Timestamped Hash-Chain Attestation” and “Tickstamp attestation”. Contrary to the multiple hash attestation, they can be classified as passive attestation mechanisms. Main idea is that, independent from current attestation requests, the attester frequently creates nonces and subsequently computes quotes.

Timestamped Hash-Chain Attestation

Within the Timestamped Hash-Chain Attestation a trusted third party provides an initial nonce to the attester. The attesting service performes frequently quotes and provides them to appraisers. For the initial quote, the nonce of the trusted third party is applied. For all further quotes the hash of the respective previous nonce is appied (hash chain). Some additional mechanisms ensure the integrity of the process (see [STU-2008]). An appraiser could validate the nonces if it can resolve the initial nonce. However, Strumpf suggests an implicit method where the attesting service is measured by the TPM. That can be validated by the appraiser according to the typical remote attestation mechanisms.

Tickstamp attestation

The Tickstamp Attestation is similar to the previous mechanism. Quotes are frequently performed using locally created nonces. This approach however, relies on the *TPM_Current_Ticks* operation. TPM ticks are utilized to confirm the time of creation. Initially, at any point of time, the appraiser verifies an attestation token. That may be obtained from the attester or a third party. In following attestation requests, the current token can be compared to the initial (or previous) token. Modifications of the attested TCS can be recognized.

Approach

Though these solutions promise faster performance than the Multiple Hash Attestation, the are incompatible to the current PWP and Tracker Protocol design. This is due to the fact that an additional function is bound to nonces of the current protocols. They are used to confirm the authenticity of public keys. More precisely, nonces are represented by public keys and the keys are singed during the quote operation. In order to take advantage of the suggested solutions, a new mechanism is required to ensure authenticity of public keys. That seems to have a large impact to the protocol design, since a fundamental concept is changed. Such a modification is expected to lead to new protocols which is not in scope of this thesis.

5.3.5 Peer-Level Shared Secrets

The current version of the PWP defines that the protocol has to be performed for each resource and each peer. Even if the same peer is requested to share several resources, for each resource the protocol has to be performed. But it seems to be unnecessary and inefficient to exchange many secrets between two peers. A single shared secret could be reused for the exchange of all resources. However, there are some reasonable counter-arguments. They imply, that an advantage might not be expected in the considered field of business.

Counter-Arguments

The recognized lack of performance doesn't have to be expected in context of the considered field of business. Instead, it must be assumed, that resources are properly distributed in the network. This is an elementary property of BitTorrent content delivery (contrary to e.g. broadcasting) in practice. Two peers that are sharing more than one resource becomes increasingly improbable, with an increasing network size. Some mechanisms enforce this behavior. An example is the creation of a peer list by the tracker. Typically, that list is limited to an applicable size. E.g. for a swarm of 1000 peers, only 50 may randomly be included in the response list. The receiver of the list may not connect all received peers. Another limited amount of e.g. 20 randomly selected peers will be connected. Out of these, some may refuse a connection if they have already reached their limit of connections. As long as this common BitTorrent approach is applied, it becomes unlikely that two peers exchange many resources within a reasonable period of time (e.g. days).

Relevant advandages could only be expected, if several resources are exchanged within a small period of time. If e.g. one peer would like to receive ten different resources from a single other peer, the protocol must be executed ten times. According to experiment A.4.9 on page XVII, the mean SrvPT for the calling peer is 6,601 seconds. Hence, an overall time of more than one minute can be expected for ten sequentially processed connections. In this case an optimization could be 10 times faster. In general, as many times as different resources are exchanged with the same peer.

Approach

After an initial execution of the PWP between two peers, a shared secret key is established. A subsequent request between the same peers for another resource could be performed in a different way (see protocol 8). As a consequence, the tracker must be able to manage shared secrets per peer and not per swarm. It might become necessary to establish a socket between the peers, in order to prevent confusion of messages for different resources.

Protocol 8 Limited Peer-Wire Protocol

0. Setup previous to protocol.

$$pA: \quad (AIK^{pA}_{pub}, AIK^{pA}_{priv}) ,$$
$$data_{i+1} := \big(Address_{pB}, AIKCert^{pB}_{ca}, ticket_{i+1}\big), K^{pA,pB}$$
$$pB: \quad (AIK^{pB}_{pub}, AIK^{pB}_{priv}) , K^{pB,t}, K^{pA,pB}$$

1. Peer pA sends the initial request (handshake).

$$pA \rightarrow pB: \quad peerID^{pA} , K^{none}_{pub} , \qquad (1)$$
$$ticket_{i+1} := enc\,\big\{AIKCert^{pA}_{ca}, resource_{i+1}, time\big\}_{K^{pB,t}}$$

2. Peer pB sends the final encrypted response (handshake).

$$pB \rightarrow pA: \quad enc\,\{Content_{i+1}\}_{K^{pA,pB}} \qquad (2)$$

1a) Peer pA initiates a limited PWP execution with peer pB. It includes the received ticket for another resource. Transmission of a key is not necessary. However, the key field is kept to provide a common interface.

1b) Peer pB can recognize the limited execution, if it compares the delivered peerID with a list of connected (shared secret available) peerID's. During the limited PWP execution only the first and the last message of the PWP have to be exchanged.

2a) Peer pB answers with an encrypted content according to the usual processing. The shared secret from the previous protocol execution is applied.

2b) Peer pA (and only peer pA) is able to decrypt the message. The shared secret from the previous protocol execution is applied.

5.4 Conclusion

Within this chapter, concepts for the optimization of performance are presented. As a result, an application with better performance and appropriate measures are provided. Some concepts apply to the currently implemented application and others to the introduced protocols and therewith, to similar applications in general. In the current state of the software, most of the gained performance could be achieved by elimination of low quality algorithms of the application. That is essentially the result of quality assurance which could be expected from an initial software version. Since the current version is intended to be only a demonstrator, more effort on quality assurance is not reasonable. Hence, more performance issues on application level can be expected, when the application is intensively used in more dynamic scenarios.

Some optimizations apply to the protocols, but not all of them could be implemented. Some are just discussed and recommended not to be implemented, because of a bad performance gain to trade-off ratio. Nevertheless, there is currently no observed bottleneck left that would seriously prevent an introduction of these protocols in practice. Latest experiments (see appendix A.4.9) indicate a mean $SrvPT\,t$ (Tracker Protocol) and a mean $SrvPT\,pB$ (Peer-Wire Protocol) of about 2 seconds each.

As initially stated, further optimization in one direction (e.g. performance) will in general be possible. Furthermore, the introduced protocols are still subject to scientific research. It can be expected that further development and application in different business scenarios will lead to modifications. Especially optimations must be put into question if requirements are changing.

6 Conclusion and Outlook

6.1 Results

This thesis describes an implementation, evaluation and optimization of a trustworthy P2P communication system that allows controlled and protected distribution of data. Cryptographic protocols using hardware based security functionality (TPM) are applied to a certain exemplary P2P system (BitTorrent).

Implementation. The application demonstrates the fundamental use case of a trustworthy data distribution between peers that are located at the customer's site. The thesis describes a practicable way to extend a broadly used P2P communication system in order to allow control of the distribution of content. Cryptographic protocols are implemented that rely on a trusted platform module and integrated libraries. Other cryptographic engines and formats can be configured. Scientific concepts and practically approved technologies are integrated.

Evaluation. An empirical study is provided that evaluates the performance of the cryptographic protocols. A measurement technique could be defined and applied in a new laboratory that was not used for measurement before. The laboratory setup, experiment setup and scenarios are defined in detail. Hence, experiments are easily reproducible.

Optimization. Some concepts could be developed to improve the performance behavior of the application and the protocols. Several could be implemented and approved. A solution for the problem of piling up TPM requests could be adapted to the suggested protocols.

6.2 Conclusion

The implemented demonstrator approves, that the proposed protocols can be applied on the basis of state of the art technologies. These protocols provide a solution to security concerns that scientists address to the commercial application of P2P networks. It has to be noted however, that the demonstrator cannot be applied in the given business field without the context of a NaDa like system. The NaDa system is now equipped with one of its main components that allows to study basic scenarios.

The protocols can be expected to be performed in about 2 seconds each, assuming the parameters described in this thesis. Further optimization is possible, but the TPM performance seems to set a minimum for the overall performance. For example the peer, involved in the TP processing, has to wait a minimum of about 450 ms (*Quote(pA)*) of the overall *SrvPT t* of about 2 seconds to complete the protocol. During the PWP, two quotes have to be performed. Hence, the minimum processing time is about twice as much as the TP.

6.3 Outlook

Development of a prototype. Next step in direction to an operative system would be a fully functional prototype. Concepts that are only described in the previous chapters should be implemented and the quality of the whole application should be assured by an extensive functional- and stress testing phase. Additionally, a prototype would require evaluating interfaces to other components of the NaDa system (management, virtualization, storage, user interfaces, monitoring, billing, policies and others). They have to be specified in detail or implemented if possible. The available high level architecture is not sufficient [KUN-2009] for this purpose. A proper interface to the management component would have to provide more than just BitTorrent functionality as it is currently implemented. Further insight can be expected from ongoing integration activities with a project partner. A graphical management- and monitoring interface that reflects the state and ongoing processing of each network component, would be a valuable component.

Other Fields of Business. Now that a trustworthy P2P protocol is successfully implemented, it might be studied, if the results can be applied in other areas. Similar integration of the trustworthy protocols may be considered for areas where P2P communication is already applied (e.g. networks like VoIP, Car-to-Car, Car-to-X, Energy and others). It has to be evaluated whether the achieved security goals are desirable in these areas.

Further research. Though the standalone P2P communication demonstrator provides acceptable measures, the whole NaDa system has to be taken into account for the commercial application. Measures of that system, containing all relevant components, would be valuable. An initial integration with management components of a project partner is ongoing. Furthermore, the applied protocols are not formally verified and analyzed. That is recommended in order to prevent certain security issues in practice. The current work does not address advanced management of a P2P network. Currently, communication to single centralized trackers is evaluated. In the last years however, systems have been successfully deployed that apply a distributed P2P management. An initial concept for a trustworthy distributed tracker is proposed in [KUN-2010] and further effort in this field of research is spent during an ongoing thesis of Lincke [LIN-2010].

Appendix A Measurement

The most important artefacts of the measurement process are the experiment description and the experiment report. They are provided for each set of experiments. Examples are depicted on the following pages.

Appendix A.1 Experiment Description

The experiment description can be interpreted by the OMF framework. It describes the invocation and termination of each participating component.

```
# --------------------------------------------------------------------------
# File:     expRunExtendedBittorrent.rb
# Author:   korn
# Date:     100419
# Desc.:    Three applications (tracker, client1, client2) are started by OMF.
#           At first, client2 communicates with the tracker over the
#           tracker protocol. Thereafter client 1 communicates with the tracker
#           over the tracker protocol. Client1 receives the address of client2
#           from the tracker. Then client1 initiates the communication via the
#           PWP with client2. At the end of the experiment client1 should
#           have downloaded an image from client2.
# Modif.:   100419 korn  creation
# --------------------------------------------------------------------------

# --------------------------------------------------------------------------
# - Define Variables
# --------------------------------------------------------------------------
trackerIP="10.148.151.25"
env="EXP"
path_torrentFile="/workspace/TrustedBitTorrent/bin/trustedBittorrent/example/client2/
    nada_customer_02_app.torrent"
path_downloadFolder="/workspace/TrustedBitTorrent/bin/trustedBittorrent/example/client2/"
path_logFile="/workspace/TrustedBitTorrent/bin/trustedBittorrent/example/client2/logfile.log"

# --------------------------------------------------------------------------
# - Define Applications
# --------------------------------------------------------------------------
defApplication('app_tracker', 'tracker') {|app|
 app.shortDescription = "BitTorrent Tracker"
 app.binaryRepository = "TrustedBitTorrentBIN.tar"
 app.path = ("/workspace/TrustedBitTorrent/bin/trustedBittorrent/example/trusted_bittorrent.sh
      #{env} tracker")
}
defApplication('app_client1', 'client1') {|app|
 app.shortDescription = "BitTorrent Peer (Seeder)"
 app.binaryRepository = "TrustedBitTorrentBIN.tar"
 app.path = ("/workspace/TrustedBitTorrent/bin/trustedBittorrent/example/trusted_bittorrent.sh
      #{env} client1 #{trackerIP}")
}
defApplication('app_client2', 'client2') {|app|
 app.shortDescription = "BitTorrent Peer (Downloader)"
 app.binaryRepository = "TrustedBitTorrentBIN.tar"
 app.path = ("/workspace/TrustedBitTorrent/bin/trustedBittorrent/example/trusted_bittorrent.sh
      #{env} client2 #{trackerIP}")
}

# ------------------------------------------------------------------------------------------
# - Define Groups
# ------------------------------------------------------------------------------------------
defGroup('grp_tracker', [ [1,5] ]) {|node|
 node.addApplication('app_tracker')
}
defGroup('grp_client1', [ [1,4] ]) {|node|
 node.addApplication('app_client1')
}
defGroup('grp_client2', [ [1,3] ]) {|node|
 node.addApplication('app_client2')
}

# ------------------------------------------------------------------------------------------
# - Run Experiments
# ------------------------------------------------------------------------------------------
whenAllUp() {|node|

 # - prepare
 puts "[EXPERIMENT] preparing experiment"
 puts "[EXPERIMENT] REVISION (see svn_revision_information.txt)"
 puts "[EXPERIMENT] clean before start"

 # - tracker
 wait 5
 puts "[EXPERIMENT] starting grp_tracker"
 group('grp_tracker').startApplications

 # - client_1
 wait 5
 puts "[EXPERIMENT] starting grp_client1"
 group('grp_client1').startApplications

 # - client_2 (! torrent is expected in .tar)
 puts "[EXPERIMENT] waiting for grp_client1"
 wait 140
 puts "[EXPERIMENT] starting grp_client2"
 group('grp_client2').startApplications

 # - end
 puts "[EXPERIMENT] waiting for grp_client2"
 wait 250
 puts "[EXPERIMENT] inquire cpuload"
 allGroups.exec('ps', ['-eo', 'pcpu,pid,user,args'])
 wait 5
 puts "[EXPERIMENT] stopping all applications"
 allGroups.stopApplications
 wait 3
 puts "[EXPERIMENT] reading SML (measurement parameter)"
 allGroups.exec('cat', ['/var/log/nada_sml'])

 wait 3
```

```
  puts "[EXPERIMENT] removing files for clean next start"
  allGroups.exec('mv', ['/workspace/TrustedBitTorrent/bin/trustedBittorrent/example/client2/
      nada_customer_02_app.img', '/tmp/nada_customer_02_app.img'])
  allGroups.exec('locate', ['peers.xml'])
  allGroups.exec('mv', ['/workspace/TrustedBitTorrent/bin/trustedBittorrent/example/tracker/www
      /files/peers.xml', '/tmp/peers.xml'])
  wait 3
  Experiment.done
}
```

Appendix A.2 Experiment Report

Within the experiment report (.xls), data of all single experiments is collected. Basic calculations are processed within the sheet, advanced calculations are processed in java classes.

EXPERIMENT REPORT

ExperiemntSet #:	100814_ExpSet100_TBT_Standard_R58.xls
Description:	100 experiments of the standard scenario for application revision 58.

SrvPT (values in ms)

Node	MP – BEGIN	Step	Step	MP – END	MIN	MEAN	MAX	MEDIAN	STD	90.00%
n_1_3	MP00TTP-RCVD'	0	UPD	MP00TTP-UPDA'	3,688	7,174	21,955	6,337	3,194	11,145
n_1_5	MP00TTP-RCVD'	1	6	MP00TTP-SEND	1,671	1,943	2,763	1,940	0,147	2,061
n_1_3	MP00PWP-RECV	UPD	5	MP00PWP-RECV	2,523	6,601	18,619	5,605	3,693	11,286
n_1_4	MP00TTP-UPDA'	1	5	MP00PWP-SEND	2,059	2,197	2,997	2,157	0,149	2,314

Thesis Format (values in ms)

Node	MP – BEGIN	Step	Step	MP – END	MIN	MEAN	MAX	MEDIAN	STD	90.00%
n_1_3	MP00TTP-RCVD'	0	1	MP00TTP-SEND	1,669	5,110	19,988	4,339	3,192	9,149
n_1_5	MP00TTP-RCVD'	1	3	MP00TTP-SEND	0,413	0,536	0,658	0,541	0,054	0,603
n_1_3	MP00TTP-RCVD'	3	4	MP00TTP-SEND	1,044	1,224	1,997	1,205	0,133	1,336
n_1_5	MP00TTP-RCVD'	4	6	MP00TTP-SEND	0,145	0,182	0,232	0,185	0,015	0,201
n_1_3	MP00TTP-RCVD'	6	UPD	MP00TTP-UPDA'	0,095	0,120	0,259	0,111	0,029	0,143
n_1_5	MP20TTP-DHBE	2	2	MP20TTP-DHBE	0,074	0,105	0,179	0,077	0,036	0,161
n_1_3	MP50TTP-DHBE	5	5	MP50TTP-DHBE	0,080	0,093	0,168	0,090	0,015	0,099
n_1_3	MP00TTP-UPDA'	UPD	1	MP00PWP-SEND	0,426	4,403	16,491	3,454	3,691	9,216
n_1_4	MP00PWP-RECV	1	2	MP00PWP-SEND	1,003	1,028	1,129	1,024	0,020	1,052
n_1_3	MP00PWP-RECV	2	3	MP00PWP-SEND	0,995	1,112	1,889	1,075	0,148	1,235
n_1_4	MP00PWP-RECV	3	5	MP00PWP-SEND	0,047	0,057	0,099	0,056	0,008	0,066
n_1_3	MP40PWP-DHBE	4	4	MP40PWP-DHBE	0,073	0,085	0,162	0,082	0,013	0,090
n_1_4	MP40PWP-DHBE	4	4	MP40PWP-DHBE	0,072	0,079	0,140	0,078	0,007	0,084

SCENARIO DATA

Node	MP – BEGIN	Step	Step	MP – END	MIN	MEAN	MAX	MEDIAN	STD	90.00%
										(Percentil)
n_1_5	MP00TTP-RCVD'	-1	0	MP00TTP-SEND	1,004 ms	1,415 ms	2,127 ms	1,427 ms	0,193 ms	1,649 ms
n_1_5	MP00TTP-SEND	0	1	MP00TTP-RCVD'						
n_1_5	MP00TTP-RCVD'	1	3	MP00TTP-SEND	1,336 ms	1,419 ms	1,642 ms	1,416 ms	0,044 ms	1,474 ms
n_1_5	MP00TTP-SEND	3	4	MP00TTP-RCVD'						
n_1_5	MP00TTP-RCVD'	4	6	MP00TTP-SEND	0,304 ms	0,331 ms	0,373 ms	0,329 ms	0,013 ms	0,346 ms
n_1_5	MP00TTP-SEND	6	0	MP00TTP-RCVD'						
n_1_5	MP00TTP-RCVD'	-1	0	MP00TTP-SEND	0,136 ms	0,256 ms	0,580 ms	0,231 ms	0,090 ms	0,350 ms
n_1_5	MP00TTP-SEND	0	1	MP00TTP-RCVD'						
n_1_5	MP00TTP-RCVD'	1	3	MP00TTP-SEND	0,413 ms	0,536 ms	0,658 ms	0,541 ms	0,054 ms	0,603 ms
n_1_5	MP00TTP-SEND	3	4	MP00TTP-RCVD'						
n_1_5	MP00TTP-RCVD'	4	6	MP00TTP-SEND	0,145 ms	0,182 ms	0,232 ms	0,185 ms	0,015 ms	0,201 ms
n.d.	MP00TTP-SEND	6	n.d.	n.d.						
n_1_4	MP00TTP-SEND	-1	0	MP00TTP-RCVD'						
n_1_4	MP00TTP-RCVD'	0	1	MP00TTP-SEND	1,431 ms	4,906 ms	15,166 ms	4,118 ms	2,870 ms	8,936 ms
n_1_4	MP00TTP-SEND	1	3	MP00TTP-RCVD'						
n_1_4	MP00TTP-RCVD'	3	4	MP00TTP-SEND	0,987 ms	1,187 ms	2,297 ms	1,166 ms	0,165 ms	1,373 ms
n_1_4	MP00TTP-SEND	4	6	MP00TTP-RCVD'						
n_1_4	MP00TTP-RCVD'	6	UPD	MP00TTP-UPDA'	0,052 ms	0,062 ms	0,133 ms	0,059 ms	0,012 ms	0,068 ms
n_1_4	MP00TTP-UPDA'	UPD	1	MP00PWP-RECV						
n_1_4	MP00PWP-RECV	1	2	MP00PWP-SEND	1,003 ms	1,028 ms	1,129 ms	1,024 ms	0,020 ms	1,052 ms
n_1_4	MP00PWP-SEND	2	3	MP00PWP-RECV						
n_1_4	MP00PWP-RECV	3	5	MP00PWP-SEND	0,047 ms	0,057 ms	0,099 ms	0,056 ms	0,008 ms	0,066 ms
n_1_4	MP00PWP-SEND	5	PP	MP00PWP-SEND						
n.d.	MP00PWP-SEND	PP	n.d.	n.d.						
n_1_3	MP00TTP-SEND	-1	0	MP00TTP-RCVD'						
n_1_3	MP00TTP-RCVD'	0	1	MP00TTP-SEND	1,669 ms	5,110 ms	19,988 ms	4,339 ms	3,192 ms	9,149 ms
n_1_3	MP00TTP-SEND	1	3	MP00TTP-RCVD'						
n_1_3	MP00TTP-RCVD'	3	4	MP00TTP-SEND	1,044 ms	1,224 ms	1,997 ms	1,205 ms	0,133 ms	1,336 ms
n_1_3	MP00TTP-SEND	4	6	MP00TTP-RCVD'						
n_1_3	MP00TTP-RCVD'	6	UPD	MP00TTP-UPDA'	0,095 ms	0,120 ms	0,259 ms	0,111 ms	0,029 ms	0,143 ms
n_1_3	MP00TTP-UPDA'	UPD	1	MP00PWP-SEND	0,426 ms	4,403 ms	16,491 ms	3,454 ms	3,691 ms	9,216 ms
n_1_3	MP00PWP-SEND	1	2	MP00PWP-RECV						
n_1_3	MP00PWP-RECV	2	3	MP00PWP-SEND	0,995 ms	1,112 ms	1,889 ms	1,075 ms	0,148 ms	1,235 ms

n_1_3	MP00PWP-SEND	3	5	MP00PWP-RECV						
n_1_3	MP00PWP-RECV	5	PP	MP00PWP-SEND	0,004 ms	0,008 ms	0,036 ms	0,007 ms	0,003 ms	0,012 ms
n.d.	MP00PWP-SEND	PP	n.d.	n.d.						
n_1_5	MP20TTP-DHBE	2	2	MP20TTP-DHBE	0,083 ms	0,089 ms	0,180 ms	0,087 ms	0,010 ms	0,093 ms
n.d.	MP20TTP-DHBE	2	n.d.	n.d.						
n_1_5	MP20TTP-DHBE	2	2	MP20TTP-DHBE	0,074 ms	0,105 ms	0,179 ms	0,077 ms	0,036 ms	0,161 ms
n.d.	MP20TTP-DHBE	2	n.d.	n.d.						
n_1_4	MP50TTP-DHBE	5	5	MP50TTP-DHBE	0,079 ms	0,088 ms	0,177 ms	0,087 ms	0,012 ms	0,092 ms
n.d.	MP50TTP-DHBE	5	n.d.	n.d.						
n_1_3	MP50TTP-DHBE	5	5	MP50TTP-DHBE	0,080 ms	0,093 ms	0,168 ms	0,090 ms	0,015 ms	0,099 ms
n.d.	MP50TTP-DHBE	5	n.d.	n.d.						
n_1_4	MP40PWP-DHBE	4	4	MP40PWP-DHBE	0,072 ms	0,079 ms	0,140 ms	0,078 ms	0,007 ms	0,084 ms
n.d.	MP40PWP-DHBE	4	n.d.	n.d.						
n_1_3	MP40PWP-DHBE	4	4	MP40PWP-DHBE	0,073 ms	0,085 ms	0,162 ms	0,082 ms	0,013 ms	0,090 ms
n.d.	MP40PWP-DHBE	4	n.d.	n.d.						
n_1_5	MP09TTP-TicketGen			MP09TTP-TicketGe	0,000 ms	0,000 ms	0,001 ms	0,000 ms	0,000 ms	0,000 ms
n.d.	MP09TTP-TicketGen		n.d.	n.d.						
n_1_5	MP09TTP-TicketGen			MP09TTP-TicketGe	0,019 ms	0,032 ms	0,063 ms	0,030 ms	0,009 ms	0,046 ms
n.d.	MP09TTP-TicketGen		n.d.	n.d.						
n_1_4	MP01LIB-DHGen			MP01LIB-DHGen	0,271 ms	3,643 ms	13,833 ms	2,926 ms	2,871 ms	7,684 ms
n.d.	MP01LIB-DHGen		n.d.	n.d.						
n_1_3	MP01LIB-DHGen			MP01LIB-DHGen	0,317 ms	3,810 ms	18,709 ms	2,987 ms	3,192 ms	7,813 ms
n.d.	MP01LIB-DHGen		n.d.	n.d.						
n_1_3	MP01LIB-DHGen			MP01LIB-DHGen	0,328 ms	4,289 ms	16,401 ms	3,351 ms	3,695 ms	9,114 ms
n.d.	MP01LIB-DHGen		n.d.	n.d.						
n_1_5	MP41TTP-MLValidation			MP41TTP-MLValida	0,138 ms	0,203 ms	0,255 ms	0,206 ms	0,027 ms	0,239 ms
n.d.	MP41TTP-MLValidation		n.d.	n.d.						
n_1_5	MP41TTP-MLValidation			MP41TTP-MLValida	0,121 ms	0,157 ms	0,292 ms	0,149 ms	0,032 ms	0,216 ms
n.d.	MP41TTP-MLValidation		n.d.	n.d.						
n_1_4	MP07TPM-Quote			MP07TPM-Quote	0,365 ms	0,435 ms	0,536 ms	0,402 ms	0,059 ms	0,527 ms
n.d.	MP07TPM-Quote		n.d.	n.d.						
n_1_4	MP07TPM-Quote			MP07TPM-Quote	0,369 ms	0,385 ms	0,412 ms	0,382 ms	0,009 ms	0,400 ms
n.d.	MP07TPM-Quote		n.d.	n.d.						
n_1_3	MP07TPM-Quote			MP07TPM-Quote	0,377 ms	0,459 ms	0,568 ms	0,418 ms	0,066 ms	0,548 ms
n.d.	MP07TPM-Quote		n.d.	n.d.						
n_1_3	MP07TPM-Quote			MP07TPM-Quote	0,381 ms	0,407 ms	0,552 ms	0,395 ms	0,041 ms	0,418 ms
n.d.	MP07TPM-Quote		n.d.	n.d.						
n_1_4	MP08AIK-Load			MP08AIK-Load	0,489 ms	0,586 ms	1,722 ms	0,564 ms	0,133 ms	0,666 ms
n.d.	MP08AIK-Load		n.d.	n.d.						
n_1_4	MP08AIK-Load			MP08AIK-Load	0,490 ms	0,509 ms	0,527 ms	0,509 ms	0,007 ms	0,520 ms
n.d.	MP08AIK-Load		n.d.	n.d.						
n_1_3	MP08AIK-Load			MP08AIK-Load	0,506 ms	0,603 ms	1,464 ms	0,581 ms	0,125 ms	0,684 ms
n.d.	MP08AIK-Load		n.d.	n.d.						
n_1_3	MP08AIK-Load			MP08AIK-Load	0,503 ms	0,596 ms	1,393 ms	0,563 ms	0,143 ms	0,660 ms
n.d.	MP08AIK-Load		n.d.	n.d.						

Appendix A.3 JVM Configuration

Excerpt of the logfile RESULT_sorted.txt of experiment 100814_ExpSet100_TBT_Standard_R58.

```
# ---------------------------------------------------------------------------
# - MEASUREMENT - JAVA SYSTEM PROPERTIES:
# ---------------------------------------------------------------------------
'java.runtime.name: OpenJDK Runtime Environment'
'sun.boot.library.path: /usr/lib/jvm/java-6-openjdk/jre/lib/i386'
'java.vm.version: 16.0-b13'
'java.vm.vendor: Sun Microsystems Inc.'
'java.vendor.url: http://java.sun.com/'
'path.separator: :'
'java.vm.name: OpenJDK Server VM'
'file.encoding.pkg: sun.io'
'sun.java.launcher: SUN_STANDARD'
'user.country: US'
'sun.os.patch.level: unknown'
'java.vm.specification.name: Java Virtual Machine Specification'
'user.dir: /workspace/TrustedBitTorrent'
'java.runtime.version: 1.6.0_18-b18'
'java.awt.graphicsenv: sun.awt.X11GraphicsEnvironment'
'java.endorsed.dirs: /usr/lib/jvm/java-6-openjdk/jre/lib/endorsed'
'os.arch: i386'
'java.io.tmpdir: /tmp'
'line.separator:'
''
'java.vm.specification.vendor: Sun Microsystems Inc.'
'os.name: Linux'
'sun.jnu.encoding: ANSI_X3.4-1968'
'java.library.path: /usr/lib/jvm/java-6-openjdk/jre/lib/i386/server:/usr/lib/jvm/java-6-
     openjdk/jre/lib/i386:/usr/lib/jvm/java-6-openjdk/jre/../lib/i386:/usr/local/jTssWrapper/
     output/lib::/usr/java/packages/lib/i386:/usr/lib/jni:/lib:/usr/lib'
'java.specification.name: Java Platform API Specification'
'java.class.version: 50.0'
'sun.management.compiler: HotSpot Tiered Compilers'
'os.version: 2.6.31-22-generic'
'user.home: /root'
'user.timezone:'
'java.awt.printerjob: sun.print.PSPrinterJob'
'file.encoding: ANSI_X3.4-1968'
'java.specification.version: 1.6'
'java.class.path: /usr/local/jTssWrapper/ext_libs/iaik_jtss_tsp.jar:/usr/local/jTssWrapper/
     ext_libs/iaik_jtpmtools.jar:/usr/local/jTssWrapper/output/jars/iaik_jtss_wrapper_swig.jar
     :/usr/local/jTssWrapper/output/jars/iaik_jtss_wrapper.jar:/usr/local/jTssWrapper/output/
     jars/:bin/trustedBittorrent/ext/groovy.jar:bin/trustedBittorrent/ext/tpmj.jar:bin/
     trustedBittorrent/ext/iaik_jtss_tcs_soap.jar:bin/trustedBittorrent/ext/iaik_jce.jar:bin/
     trustedBittorrent/ext/kxml.jar:bin/trustedBittorrent/ext/iaik_xsect.jar:bin/
     trustedBittorrent/ext/iaik_tccert.jar:bin/trustedBittorrent/ext/xalan.jar:bin/
     trustedBittorrent/ext/xerces.jar:bin/trustedBittorrent/ext/iaik_jtss_tsp.jar:bin/
     trustedBittorrent/ext/simple_upload-4.1.15.jar:bin/trustedBittorrent/ext/
     iaik_jtss_tsp_soap.jar:bin/trustedBittorrent/ext/xml-apis.jar:bin/trustedBittorrent/ext/
     jaxen-jdom.jar:bin/trustedBittorrent/ext/jdom.jar:bin/trustedBittorrent/ext/iaik_xkms.jar:
     bin/trustedBittorrent/ext/saxpath.jar:bin/trustedBittorrent/ext/iaik_jce_full.jar:bin/
     trustedBittorrent/ext/ant.jar:bin/trustedBittorrent/ext/jaxen-core.jar:bin/
     trustedBittorrent/ext/simple-xml-2.1.7.jar:bin/trustedBittorrent/ext/freemarker.jar:bin/
     trustedBittorrent/ext/iaik_jtpmtools.jar:bin/trustedBittorrent/ext/log4j-1.2.15.jar:bin/
```

```
trustedBittorrent/ext/velocity.jar:bin/trustedBittorrent/ext/routing.jar:bin/trustedBittorrent/ext/iaik_jtss_tcs.jar:bin:'
'user.name: root'
'java.vm.specification.version: 1.0'
'java.home: /usr/lib/jvm/java-6-openjdk/jre'
'sun.arch.data.model: 32'
'user.language: en'
'java.specification.vendor: Sun Microsystems Inc.'
'java.vm.info: mixed mode'
'java.version: 1.6.0_18'
'java.ext.dirs: /usr/lib/jvm/java-6-openjdk/jre/lib/ext:/usr/java/packages/lib/ext'
'sun.boot.class.path: /usr/lib/jvm/java-6-openjdk/jre/lib/resources.jar:/usr/lib/jvm/java-6-openjdk/jre/lib/rt.jar:/usr/lib/jvm/java-6-openjdk/jre/lib/sunrsasign.jar:/usr/lib/jvm/java-6-openjdk/jre/lib/jsse.jar:/usr/lib/jvm/java-6-openjdk/jre/lib/jce.jar:/usr/lib/jvm/java-6-openjdk/jre/lib/charsets.jar:/usr/lib/jvm/java-6-openjdk/jre/lib/rhino.jar:/usr/lib/jvm/java-6-openjdk/jre/classes'
'java.vendor: Sun Microsystems Inc.'
'file.separator: /'
'java.vendor.url.bug: http://java.sun.com/cgi-bin/bugreport.cgi'
'sun.io.unicode.encoding: UnicodeLittle'
'sun.cpu.endian: little'
'sun.cpu.isalist:'
```

Appendix A.4 Experiment Results

Within this section a summary of each set of experiments is given. They are referred and explained in context of the thesis. A high level overview of the test results is given below. All experiment sets consist of 100 single experiments.

Software					Configuration											Optimization Implemented				SrvPT MEAN(ms)	
					Application & Scenario							Environment (fix)									
JBT	TBT	Rev	Month	Day	Swarms	Swarm-size	SML-size	KHL-size	Simultaneous calls	Key-size	Cryptographic Algorithms	JIT	Logging	HW	OS & Env. Applications	Serialization without XML&FS	Concurrent DH	Optimized SML validation	Concurrent SML validation	SrvPT t (Tracker Protocol)	SrvPT pB (Peer Wire Prot.)
Y	N	19	6	26	1	0	-	-	1	-	-	ON	ON	fix	fix	N	N	N	N	0,070	0,009
N	Y	45	6	27	1	1	4	3	1	fix	fix	ON	OFF	fix	fix	N	N	N	N	2,395	2,170
N	Y	48	7	24	1	1	4	3	1	fix	fix	ON	OFF	fix	fix	N	N	N	N	2,275	2,150
N	Y	50	7	30	1	1	4	3	1	fix	fix	ON	OFF	fix	fix	Y	N	N	N	1,764	2,144
N	Y	51	7	31	1	1	4	3	1	fix	fix	ON	OFF	fix	fix	Y	Y	N	N	1,737	2,167
N	Y	53	8	5	1	1	400	400	1	fix	fix	ON	OFF	fix	fix	Y	Y	N	N	7,685	2,148
N	Y	54	8	6	1	1	400	400	1	fix	fix	ON	OFF	fix	fix	Y	Y	Y	N	2,126	2,152
N	Y	55	8	7	1	1	400	400	1	fix	fix	ON	OFF	fix	fix	Y	Y	Y	Y	1,881	2,139
N	Y	58	8	14	1	1	400	400	1	Sun	Sun	ON	OFF	fix	fix	Y	Y	Y	Y	1,943	2,197

Appendix A.4.1 Experiment: Experiment_nada_2010_07_28_18_59_34.tar.gz

- Results are based on a set of 100 experiments.
- Measuring the distortion that is caused by the measuring technique ($1s = 1,000ms$).

mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
MT_0	MT_{101}	0,012	0,023	0,054	0,015	0,012	0,044
MT_0	MT_{501}	0,061	0,093	0,136	0,094	0,016	0,113
MT_0	MT_{1001}	0,126	0,167	0,211	0,169	0,018	0,188

Appendix A.4.2 Experiment: 100626_ExpSet100_JBT_Standard_R19.tar.gz

- Results are based on a set of 100 experiments.
- Measuring the original java BitTorrent application.

Original Tracker Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	jBitTorrent	0,091	0,110	0,143	0,109	0,009	0,123
SrvPT *t*	jBitTorrent	0,057	0,070	0,108	0,070	0,007	0,081
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(ann)*	0,007	0,008	0,031	0,007	0,003	0,008
RECV(ann)	*SEND(res)*	0,057	0,070	0,108	0,070	0,007	0,081
RECV(res)	*updated*	0,026	0,031	0,039	0,032	0,002	0,035

Original Peer-Wire Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	jBitTorrent	0,031	0,050	0,097	0,050	0,011	0,062
SrvPT *pB*	jBitTorrent	0,005	0,009	0,025	0,007	0,004	0,015
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(hs)*	0,021	0,041	0,080	0,040	0,009	0,050
RECV(hs)	*SEND(hs)*	0,005	0,009	0,025	0,007	0,004	0,015

Appendix A.4.3 Experiment: 100724_ExpSet100_TBT_Standard_R48

- Results are based on a set of 100 experiments.
- SML includes 4 and KHL include 3 values.
- Initial (evaluated) version without optimization.

Extended Tracker Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	4,055	7,965	20,138	6,956	3,524	12,769
SrvPT *t*	tBitTorrent	1,992	2,275	3,315	2,261	0,155	2,427
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(1)*	1,646	5,543	17,786	4,517	3,558	10,372
RECV(1)	*SEND(3)*	0,596	0,770	0,935	0,780	0,073	0,853
RECV(3)	*SEND(4)*	1,042	1,210	2,282	1,189	0,154	1,321
RECV(4)	*SEND(6)*	0,198	0,294	0,441	0,287	0,050	0,366
RECV(6)	*updated*	0,131	0,146	0,226	0,137	0,019	0,174
BEG(2)	*END(2)*	0,038	0,067	0,112	0,074	0,020	0,090
BEG(5)	*END(5)*	0,049	0,056	0,104	0,056	0,007	0,063
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
DHGen(B)	*DHGen(E)*	0,344	4,223	16,428	3,238	3,564	9,052
TCrea(B)	*TCrea(E)*	0,025	0,043	0,094	0,043	0,012	0,061
SMLVal(B)	*SMLVal(E)*	0,005	0,011	0,040	0,008	0,009	0,028

Extended Peer-Wire Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,525	6,356	25,900	5,257	3,727	9,640
SrvPT *pB*	tBitTorrent	2,015	2,150	2,942	2,118	0,154	2,209
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(1)*	0,419	4,205	23,834	3,086	3,726	7,408
RECV(1)	*SEND(2)*	0,946	0,977	1,020	0,975	0,014	0,997
RECV(2)	*SEND(3)*	0,974	1,092	1,897	1,059	0,157	1,142
RECV(3)	*SEND(5)*	0,070	0,080	0,130	0,077	0,010	0,091
BEGpA(4)	*ENDpA(4)*	0,020	0,032	0,063	0,032	0,004	0,037
BEGpB(4)	*ENDpB(4)*	0,023	0,024	0,047	0,024	0,003	0,025
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
DHGen(B)	*DHGen(E)*	0,342	4,127	23,769	3,018	3,731	7,338
TCrea(B)	*TCrea(E)*	0,025	0,043	0,094	0,043	0,012	0,061

Appendix A.4.4 Experiment: 100730_ExpSet100_TBT_Standard_R50

- Results are based on a set of 100 experiments.
- SML includes 4 and KHL include 3 values.
- Serialization and deserialization is optimized.

Extended Tracker Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,534	6,563	33,763	5,401	4,695	10,263
SrvPT *t*	tBitTorrent	1,484	1,764	2,237	1,755	0,132	1,949
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(1)*	0,648	4,654	32,014	3,478	4,705	8,471
RECV(1)	*SEND(3)*	0,179	0,326	0,431	0,337	0,059	0,393
RECV(3)	*SEND(4)*	0,985	1,161	1,661	1,142	0,118	1,302
RECV(4)	*SEND(6)*	0,153	0,276	0,365	0,278	0,044	0,338
RECV(6)	*updated*	0,129	0,144	0,207	0,139	0,014	0,165
BEG(2)	*END(2)*	0,037	0,069	0,124	0,075	0,021	0,095
BEG(5)	*END(5)*	0,044	0,056	0,097	0,055	0,006	0,061

Extended Peer-Wire Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,728	6,385	15,720	5,604	2,862	9,984
SrvPT *pB*	tBitTorrent	2,014	2,144	2,898	2,125	0,130	2,230
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(1)*	0,604	4,241	13,677	3,503	2,893	7,892
RECV(1)	*SEND(2)*	0,943	0,971	1,022	0,969	0,014	0,990
RECV(2)	*SEND(3)*	0,980	1,089	1,863	1,062	0,129	1,147
RECV(3)	*SEND(5)*	0,070	0,082	0,127	0,080	0,010	0,094
BEGpA(4)	*ENDpA(4)*	0,025	0,032	0,044	0,032	0,003	0,036
BEGpB(4)	*ENDpB(4)*	0,023	0,025	0,062	0,024	0,006	0,025

Appendix A.4.5 Experiment: 100731_ExpSet100_TBT_Standard_R51

- Results are based on a set of 100 experiments.
- SML includes 4 and KHL include 3 values.
- Computing of the shared secret is performed concurrently.

Extended Tracker Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,553	6,941	26,051	5,776	4,356	12,331
SrvPT *t*	tBitTorrent	1,481	1,737	2,581	1,719	0,185	1,863
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(1)*	0,632	5,040	24,342	3,962	4,383	10,475
RECV(1)	*SEND(3)*	0,135	0,260	0,415	0,268	0,061	0,324
RECV(3)	*SEND(4)*	1,007	1,203	2,075	1,177	0,183	1,321
RECV(4)	*SEND(6)*	0,171	0,273	0,385	0,269	0,043	0,327
RECV(6)	*updated*	0,148	0,164	0,220	0,160	0,012	0,176
BEG(2)	*END(2)*	0,037	0,065	0,133	0,067	0,021	0,092
BEG(5)	*END(5)*	0,041	0,049	0,070	0,049	0,003	0,054

Extended Peer-Wire Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,658	6,042	16,201	5,122	3,121	10,439
SrvPT *pB*	tBitTorrent	2,009	2,167	2,978	2,126	0,164	2,243
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(1)*	0,500	3,875	14,066	3,050	3,118	7,796
RECV(1)	*SEND(2)*	0,964	0,990	1,026	0,989	0,013	1,011
RECV(2)	*SEND(3)*	0,984	1,116	1,945	1,070	0,162	1,171
RECV(3)	*SEND(5)*	0,047	0,060	0,121	0,056	0,011	0,073
BEGpA(4)	*ENDpA(4)*	0,022	0,029	0,054	0,028	0,005	0,037
BEGpB(4)	*ENDpB(4)*	0,023	0,027	0,065	0,026	0,005	0,031

Appendix A.4.6 Experiment: 100805_ExpSet100_TBT_Standard_R53

- Results are based on a set of 100 experiments.
- SML and KHL include 400 values each.
- The validation algorithm is derived from Ethemba project[BRE-2008].

Extended Tracker Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	8,877	12,875	24,964	12,061	3,209	17,485
SrvPT *t*	tBitTorrent	7,324	7,685	8,739	7,677	0,186	7,875
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(1)*	1,211	5,024	17,333	4,144	3,189	9,356
RECV(1)	*SEND(3)*	0,312	0,469	0,628	0,466	0,064	0,553
RECV(3)	*SEND(4)*	1,010	1,217	2,180	1,209	0,152	1,348
RECV(4)	*SEND(6)*	5,832	5,998	6,206	6,003	0,078	6,103
RECV(6)	*updated*	0,147	0,165	0,215	0,159	0,014	0,184
BEG(2)	*END(2)*	0,035	0,053	0,112	0,041	0,018	0,077
BEG(5)	*END(5)*	0,043	0,049	0,071	0,049	0,004	0,055
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SMLVal(B)	*SMLVal(E)*	5,674	5,810	6,051	5,809	0,074	5,913

Extended Peer-Wire Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,681	6,486	18,947	5,671	3,406	11,693
SrvPT *pB*	tBitTorrent	2,026	2,148	3,219	2,108	0,160	2,237
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(1)*	0,468	4,338	16,889	3,566	3,400	9,355
RECV(1)	*SEND(2)*	0,961	0,991	1,041	0,989	0,016	1,015
RECV(2)	*SEND(3)*	0,986	1,096	2,134	1,062	0,158	1,170
RECV(3)	*SEND(5)*	0,046	0,060	0,101	0,058	0,010	0,076
BEGpA(4)	*ENDpA(4)*	0,021	0,029	0,040	0,028	0,004	0,036
BEGpB(4)	*ENDpB(4)*	0,023	0,026	0,036	0,026	0,002	0,029

Appendix A.4.7 Experiment: 100806_ExpSet100_TBT_Standard_R54

- Results are based on a set of 100 experiments.
- SML and KHL include 400 values each.
- The validation algorithm is optimized.

Extended Tracker Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	3,416	7,096	26,290	6,068	3,480	11,512
SrvPT *t*	tBitTorrent	1,752	2,126	3,030	2,116	0,222	2,323
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(1)*	1,137	4,801	24,132	3,670	3,496	9,232
RECV(1)	*SEND(3)*	0,325	0,482	0,616	0,484	0,058	0,557
RECV(3)	*SEND(4)*	1,006	1,240	2,171	1,203	0,196	1,401
RECV(4)	*SEND(6)*	0,257	0,402	0,509	0,405	0,058	0,482
RECV(6)	*updated*	0,150	0,167	0,216	0,160	0,016	0,200
BEG(2)	*END(2)*	0,034	0,049	0,127	0,040	0,017	0,075
BEG(5)	*END(5)*	0,044	0,049	0,057	0,049	0,003	0,054
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SMLVal(B)	*SMLVal(E)*	0,095	0,124	0,153	0,126	0,010	0,133

Extended Peer-Wire Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,501	6,630	25,300	5,505	3,981	11,090
SrvPT *pB*	tBitTorrent	2,020	2,152	2,784	2,117	0,119	2,303
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(1)*	0,417	4,477	22,998	3,326	3,972	9,037
RECV(1)	*SEND(2)*	0,950	0,992	1,062	0,992	0,017	1,014
RECV(2)	*SEND(3)*	0,984	1,100	1,758	1,068	0,117	1,230
RECV(3)	*SEND(5)*	0,045	0,059	0,094	0,059	0,009	0,068
BEGpA(4)	*ENDpA(4)*	0,022	0,030	0,060	0,028	0,005	0,036
BEGpB(4)	*ENDpB(4)*	0,020	0,026	0,052	0,026	0,004	0,031

Appendix A.4.8 Experiment: 100807_ExpSet100_TBT_Standard_R55

- Results are based on a set of 100 experiments.
- SML and KHL include 400 values each.
- The validation algorithm is performed concurrently.

Extended Tracker Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,877	7,237	28,157	5,989	4,059	12,347
SrvPT *t*	tBitTorrent	0,678	1,881	3,204	1,876	0,248	2,047
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(1)*	0,507	5,190	26,293	3,964	4,057	10,164
RECV(1)	*SEND(3)*	0,369	0,493	1,412	0,492	0,110	0,557
RECV(3)	*SEND(4)*	1,008	1,202	2,256	1,200	0,169	1,395
RECV(4)	*SEND(6)*	0,145	0,197	0,350	0,196	0,026	0,226
RECV(6)	*updated*	0,150	0,167	0,239	0,159	0,017	0,195
BEG(2)	*END(2)*	0,037	0,054	0,092	0,046	0,017	0,078
BEG(5)	*END(5)*	0,044	0,050	0,067	0,051	0,003	0,055
measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SMLVal(B)	*SMLVa(E)*	0,103	0,154	0,270	0,139	0,040	0,215

Extended Peer-Wire Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	1,035	5,658	13,883	5,482	2,349	8,761
SrvPT *pB*	tBitTorrent	1,035	2,139	3,022	2,120	0,182	2,268
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(1)*	0,385	3,553	11,776	3,293	2,321	6,723
RECV(1)	*SEND(2)*	0,556	0,982	1,036	0,984	0,045	1,003
RECV(2)	*SEND(3)*	0,984	1,108	1,982	1,074	0,143	1,235
RECV(3)	*SEND(5)*	0,046	0,060	0,188	0,056	0,016	0,074
BEGpA(4)	*ENDpA(4)*	0,022	0,030	0,049	0,028	0,005	0,037
BEGpB(4)	*ENDpB(4)*	0,020	0,026	0,048	0,026	0,003	0,031

Appendix A.4.9 Experiment: 100814_ExpSet100_TBT_Standard_R58

- Results are based on a set of 100 experiments.
- Services and keys are used from one provider (all from Sun, see section B).

Extended Tracker Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	3,688	7,174	21,955	6,337	3,194	11,145
SrvPT *t*	tBitTorrent	1,671	1,943	2,763	1,940	0,147	2,061
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
preproc.	*SEND(1)*	1,669	5,110	19,988	4,339	3,192	9,149
RECV(1)	*SEND(3)*	0,413	0,536	0,658	0,541	0,054	0,603
RECV(3)	*SEND(4)*	1,044	1,224	1,997	1,205	0,133	1,336
RECV(4)	*SEND(6)*	0,145	0,182	0,232	0,185	0,015	0,201
RECV(6)	*updated*	0,095	0,120	0,259	0,111	0,029	0,143
BEG(2)	*END(2)*	0,074	0,105	0,179	0,077	0,036	0,161
BEG(5)	*END(5)*	0,080	0,093	0,168	0,090	0,015	0,099

Extended Peer-Wire Protocol ($1s = 1,000ms$)

measure	application	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
SrvPT *pA*	tBitTorrent	2,523	6,601	18,619	5,605	3,693	11,286
SrvPT *pB*	tBitTorrent	2,059	2,197	2,997	2,157	0,149	2,314
mp_t	mp_{t+i}	MIN(ms)	MEAN(ms)	MAX(ms)	MED(ms)	STD(ms)	P90(ms)
updated	*SEND(1)*	0,426	4,403	16,491	3,454	3,691	9,216
RECV(1)	*SEND(2)*	1,003	1,028	1,129	1,024	0,020	1,052
RECV(2)	*SEND(3)*	0,995	1,112	1,889	1,075	0,148	1,235
RECV(3)	*SEND(5)*	0,047	0,057	0,099	0,056	0,008	0,066
BEGpA(4)	*ENDpA(4)*	0,073	0,085	0,162	0,082	0,013	0,090
BEGpB(4)	*ENDpB(4)*	0,072	0,079	0,140	0,078	0,007	0,084

Appendix B Cryptography Integration

Appendix B.1 Applied Configuration

Up to revision 57, the default (chosen by runtime environment) configuration of engines for each provider is used. It is not evaluated in detail. With revision 58 the provider for each engine is set to a fixed value as shown in the following tables (keys and services). Corresponding measurement results can be found in section A.4. The applied provider versions are: SunJSSE 1.7, SunJCE 1.7, SunRsaSign 1.7, Sun 1.6, IAIK/JCE 3.16 and Infineon 1.2. The key performance indicator show similar results.

Appendix B.1.1 Keys

Keys	Size(bit)	Type	Format	Engine	Provider
S^t_{pub}, S^t_{priv}	512	Asym.	RSA	KeyPairGenerator	SunJSSE
K^i_{pub}, K^i_{priv}	1024	Asym.	DH	KeyPairGenerator	SunJCE
K^i_{pub}, K^i_{priv}	-	Param	DH	AlgParameterGen	SunJCE
$K^{i,j}$	56	Sym.	DES	KeyAgreement	SunJCE
$R^{i,j}$	128	Sym.	AES	KeyGenerator	SunJCE
$AIKCert^i_{ca}$	-	Cert	X509	AIKCertificate	IAIK
$AIK^i_{pub}, AIK^i_{priv}$	2048	Asym.	RSA	TPM	Infineon
$C^{ca}_{pub}, C^{ca}_{pub}$	512	Asym.	RSA	KeyPairGenerator	SunJSSE

Appendix B.1.2 Services

Services	Format,Algorithm,Mode	Engine	Provider
$quote\{\}$	RSA	TPM	Infineon
$ver\{quote\}$	RSA	TSS TcCrypto	IAIK
$hash$	SHA1	MessageDigest	SUN 1.6
$enc/dec\{\}_R$	AES/ECB/PKCS5Padding	Cipher	SunJCE
$enc/dec\{\}_{S_{pub}}$	RSA	Cipher	SunJCE
$enc/dec\{\}_K$	DES/ECB/PKCS5Padding	Cipher	SunJCE
$sig/ver\{\}$	SHA1withRSA	Signature	SunRsaSign
$ver\{AIKCert^i_{ca}\}$	X509	X509Certificate	IAIK

Appendix B.2 Certificate Example

This example certificate is used as a basis for the implemented protocols. It is created by the integrated Ethemba application[BRE-2008].

```
Serial number: 20100814123714103
Signature algorithm: sha1WithRSAEncryption
    (1.2.840.113549.1.1.5)
Issuer: CN=Fraunhofer SIT,OU=NaDa,O=Fraunhofer SIT,C=DE
Valid not before: Sat Aug 14 12:36:14 CEST 2010
not after: Sun Aug 14 12:36:14 CEST 2011
Subject:
RSAES-OAEP public key (2048 bits):
public exponent: 10001
modulus: a8dba942a8f3b80685907693adf774ec3fd33d9de82eff15ed0ece
5f9392ebd1962b72188179129d9c40d71a21da5f56e0c94831dd96dcbb45c68
ead5823cbbebb132d6b86c557f5dd48c13dcd4dda81c44317aa054033620a59
db28cdb50831bb06f5f771ae21a8f22f0e17805d9cdfaae9890954652b46fb9
db20070630d9a6d3d5e11786590e626ee77be08ff07605accf10abd44926bca
b6ce66f99340aef33e53023ca681b3bead6e6ca6f0ebdfe9a283360e520d641
7d9ffa1747c2bbc6acce54eb452d9ec43bd266a2b19196e97b81d9f7be7322d
dd7c51c8e4f302d47c9044a033728175a916275c001d0781d4f7accbfed6600
36f7acc00d1c48537
parameters:
Hash algorithm: SHA (1.3.14.3.2.26)
Mask generation algorithm:
MGF1 (1.2.840.113549.1.1.8) with parameter P
Source algorithm: p
Specified (1.2.840.113549.1.1.9) with parameter
Certificate Fingerprint (MD5) :
AA:E1:A7:C6:4E:EF:CA:C4:1C:00:81:0F:A2:4B:F2:B7
Certificate Fingerprint (SHA-1):
B6:72:0A:E3:52:AD:F0:AC:A5:01:D2:57:16:EB:0E:DD:4C:DC:5C:1B
Extensions: 5
```

Appendix C Thesis DVD (back side of the cover)

Appendix C.1 Folder Structure

Not all practical work results of this thesis can be provided in this appendix. This is due to the large amount of files and the size of the files. All related files are stored on a DVD that is provided with this thesis. In the followging, the folder structure on the DVD is depicted and described up to a depth of three (3rd-Level).

File System Structure (1st-Level)	Description
• Experiments	All experiment results in different versions.
• ProjectFiles	All project files and sources in different versions.
• References	All Internet references within the Thesis.pdf file.
Masterarbeit_RomanKorn.pdf	Final thesis in Portable Document Format.

Appendix C.2 Experiments

The folder `Experiments` contains result files of all referenced experiments. Its content may differ for each revision. The final structure is depicted in the next section (folder `ProjectFiles/Revision_Final_tBittorrent/experiments`). The naming convention is: <YYMMDD>_ExpSet<SIZE>_<APPLICATION>_<SCENARIO>_<REVISION>. Each single experiment within a set of experiments is stored as (tar.gz) compressed folder. It can be extracted using the tar command in Unix or using several zip (e.g. http://www.7-zip.de/) tools in Windows based environments.

File System Structure (2nd-Level)	Description
• Experiments	
• 100626_ExpSet100_JBT_Standard_R19	Result files of particular experiment.
• 100724_ExpSet100_TBT_Standard_R48	Result files of particular experiment.
• 100730_ExpSet100_TBT_Standard_R50	Result files of particular experiment.
• 100731_ExpSet100_TBT_Standard_R51	Result files of particular experiment.
• 100805_ExpSet100_TBT_Standard_R53	Result files of particular experiment.
• 100806_ExpSet100_TBT_Standard_R54	Result files of particular experiment.
• 100807_ExpSet100_TBT_Standard_R55	Result files of particular experiment.
• 100809_ExpSet100_TBT_MTech_R55	Result files of particular experiment.
• 100814_ExpSet100_TBT_Standard_R58	Result files of particular experiment.
• CapturesOfNetworkTraffic	Captures of the wirewhark tool.

Appendix C.3 Project Files

The folder `ProjectFiles` contains all project related sources and other files including: application files, external files, experiment files. Several versions of the project are provided.

File System Structure (2nd-Level)	Description
• ProjectFiles	Files from different versions of the project.
• Revision_00_BaptisteDubuis	Original jbittorrent-v1.1 sources (not runnable).
• Revision_09_ImranKhalid	Initial project setup by Imran Khalid (not runnable).
• Revision_19_jBitTorrent	Runnable jBitTorrent version for measurement.
• Revision_47_tBitTorrent	Runnable tBitTorrent version for measurement.
• Revision_48_tBitTorrent	Runnable tBitTorrent version for measurement.
• Revision_50_tBitTorrent	Runnable tBitTorrent version for measurement.
• Revision_51_tBitTorrent	Runnable tBitTorrent version for measurement.
• Revision_53_tBitTorrent	Runnable tBitTorrent version for measurement.
• Revision_54_tBitTorrent	Runnable tBitTorrent version for measurement.
• Revision_55_tBitTorrent	Runnable tBitTorrent version for measurement.
• Revision_58_tBitTorrent	Runnable tBitTorrent version for measurement.
• Revision_Final_tBitTorrent	Final trustworthy BitTorrent version.
• OtherSources	PCA, WebServer, TPMEmulator, jTSSWrapper, ...
SVN_Changes.log	Complete project change-log from revision 1 to final.

File System Structure (3rd-Level)	**Description**
• Revision_Final_tBittorrent	
• bin	All binaries (structure according to folder `src`).
• experiments	All experiment related files and scripts.
• ExperimentSet	Files and scripts for measure calculation.
• calc	Results of calculations.
• Experiments	Experiment (.tar files) from OMF-Laboratory.
ExperimentReport.xls	Calculation sheet of current experiment.
calcMeanSigma_SrvPT.sh	Calculate mean and sigma of SrvPT.
calcMeanSigma.sh	Calculate mean and sigma of all measurement sets.
clean.sh	Script to clean the environment.
collectDataSets.sh	Collect measurement sets from experiments.
• Scenario_1	OMF scripts for current experiment scenario.
• scripts	
create.sh	Create Experiment (.tar) and send to EC.
exec-kill.rb	Kill all left over processes in OMF.
ext.sh	Extract Experiment (.tar) on EC.
infrastructure.sh	Prints environment information.
log.sh	Collecting experiment log files and formatting.
mps.sh	Collecting measurement points from log files.
run.sh	Run given number of experiments (sequential).
schedule.sh	Run given # of experiments in given # of seconds.
expRunExtendedBittorrent.rb	Runnable OMF experiment definition.
RESULT_*	Single experiment result/log files.
• src	Java source files.
• trustedBittorrent	Trusted bittorrent application.
• app	External application sources.
• example	Example application files and storage directory.
• ext	External libraries (OMF-Laborartory).
• ext-dev	External libraries (development environment).
• jBittorrentAPI	Peer functionality.
• jTPMtools	TPM related tools.
• messages	Tracker Protocol messages.
• net	Network definitions and entities.
• trackerBT	Tracker functionality.
• types	Commonly used types.
• utils	Commonly used utilities.
HowTo.txt	How to run each program.
Settings.java	Some global settings of all applications.

Abbreviations

AES	Advanced Encryption Standard
AIK	Attestation Identity Key
AlgPT	Algorithm Processing Time
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency Network
BIOS	Basic Input/Output System
CA	Certificate Authority
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CL	Connection Listener
CORDIS	Community Research and Development Information Service
CP	Content Provider
CPU	Central Processing Unit
CRTM	Core Root of Trust for Measurement
DDR	Double Data Rate
DES	Digital Encryption Standard
DH	Diffie-Hellman
DHT	Distributed Hash Table
DM	Download Manager
DNS	Domain Name System
DSA	Digital Signature Algorithm
DT	Download Task
EC	Experiment Controller
ECB	Electronic Code Book
ED	Experiment Description
EK	Endorsement Key
et al.	et alii
FIFO	First-In-First-Out
GHz	Gigahertz
HTTP	Hypertext Transfer Protocol
HS	Handshake
IAIK	Institute for Applied Information Processing and Communication
IMA	Integrity Measurement Architecture
IP	Internet Protocol
IPL	Initial Program Load
IPTV	Internet Protocol Television
ISP	Internet Service Provider
ITU	International Telecommunication Union
I/O	Input/Output
jBitTorrent	Java BitTorrent
JCA	Java Cryptographic Architecture
JCE	Java Cryptographic Extension
JDK	Java Development Kit

JIT	Just-in-time
JVM	Java Virtual Machine
KHL	Known Hash List
MAC	Message Authentication Code
MBR	Master Boot Record
MED	Median
Mgnt	Management
MHz	Megahertz
MR	Message Receiver
MS	Message Sender
ms	Milli Seconds
MTM	Mobile Trusted Module
NaDa	NanoDataCenters
nonce	Number Used Once
ns	Nano Seconds
OAEP	Optimal Asymmetric Encryption Padding
OEDL	OMF Experiment Description Language
OFB	Output Feedback
OMF	cOntrol and Management Framework
PC	Personal Computer
PCR	Platform Configuration Register
PCT	Percentile
RFC	Request for Comments
PKCS	Public Key Cryptography Standard
PKI	Public Key Infrastructure
PWP	Peer Wire Protocol
PP	Peer Protocol
P2P	Peer-to-Peer
P90	90th Percentile
RAM	Random Access Memory
RNG	Random Number Generator
ROM	Read Only Memory
RPC	Remote Procedure Call
RSA	Rivest, Shamir, Adleman
RTR	Root of Trust for Reporting
RTM	Root of Trust for Measurement
RTS	Root of Trust for Storage
SHA	Secure Hash Algorithm
SML	Stored Measurement Log
SRK	Storage Root Key
SrvPT	Service Processing Time
SSH	Secure Shell
STD	Standard Deviation
TBB	Trusted Building Block
tBitTorrent	Trustworthy BitTorrent

TC	Trusted Computing
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TCP	Trusted Computing Platform
TCP	Transmission Control Protocol
TCS	Trusted Computing System
TCS	TSS Core Service
TOS	Trusted Operating System
TP	Trustworthy Protocol
TPM	Trusted Platform Module
TP	Tracker Protocol
TSP	TSS Service Provider
TSS	Trusted Software Stack
TTP	Trusted Third Party
TV	Tele Vision
UGC	User Generated Content
URL	Unified Resource Locator
UUID	Universally Unique Identifier
VM	Virtual Machine
VoD	Video on Demand
VoIP	Voice over IP
WWW	World Wide Web
XMPP	Extensible Messaging and Presence Protocol

References

[AYY-2010] Ayyasamy, S.; Sivanandam, S.N.: *Trust Based content Distribution for Peer-To-Peer Overlay Networks*. In: International Journal of Network Security & Its Applications (IJNSA), Volume 2, Number2, April 2010.

[BAL-2005] Balfe, Shane; Lakhani, Amit D.; Paterson, Kenneth G.: *Securing Peer-to-Peer Networks Using Trusted Computing* . In: Trusted Computing (Edited by Chis Mitchell), IEE Press, 2005, pages 271-298.

[BRE-2008] Brett, Andreas; Leicher, Andreas: *Ethemba* : Trusted Host Environment Mainly Based on Attestation. http://ethemba.info/ethemba/ethemba.pdf, created at 24.12.2008, accessed at 23.07.2010

[BUC-2008] Buchmann, Johannes: *Einführung in die Cryptographie* . Vierte, erweiterte Auflage Heidelberg: Springer Verlag, 2008.

[CHA-2009] Chang, Hyunseok; Jamin, Sugih;Wang, Wenjie. *Live streaming performance of the zattoo network.* In: IMC '09: Proceedings of the Ninth ACM SIGCOMM Internet Measurement Conference 2009, pages 417–429, New York.

[COH-2008] Bram Cohen: *The BitTorrent Protocol Specification* : Version 11031 . http://bittorrent.org/beps/bep_0003.html, created at 28.02.2008, accessed at 13.04.2010.

[COK-2010] Coker, George; Guttman, Joshua: *Principles of Remote Attestation* : International Journal of Information Security, accepted 2010.

[DIF-1988] Diffie, Whitfield: The First Ten Years of Public-Key Cryptography. In: Proceedings of the IEEE, 1988, Pages 560-577.

[DUB-2007] Dubuis, Baptiste: *JBittorrent* . http://jbittorrent.sourceforge.net, crated at 19.09.2007, accessed at 22.07.2010.

[EUC-2009] Eurpean Commision Community Research and Development Information Service: *CORDIS : ICT : Projects : NANODATACENTERS : Nano data centers* . http://cordis.europa.eu/fetch?CALLER=PROJ_ICT&ACTION=D&DOC=17&CAT=PROJ&QUERY=012ad7528bbd:a7be:792ee0e0&RCN=86610, created at 03.03.2009, accessed at 03.09.2010.

[ECK-2008] Eckert, Claudia: *IT-Sicherheit* : Konzepte - Verfahren - Protokolle. 5. Auflage München: Oldenburg Verlag, 2008.

[GAL-2002] Gallagher, Niall: *Simple* . http://sourceforge.net/projects/simpleweb, created at 12.09.2002, accessed at 22.07.2010.

[HUA-2007] Huang, Cheng; Li, Jin; Ross, Keith W.: *Can Internet Video-on-Demand be Profitable?*. In: Proceedings of the 2007 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2007, pages 133-144.

[IAI-2010a] Institute for Applied Information Processing and Communication (IAIK): *JCA/JCE* . http://jce.iaik.tugraz.at/sic/Products/Core-Crypto-Toolkits/JCA-JCE, created in 2010, accessed at 23.07.2010.

[IAI-2010b] Institute for Applied Information Processing and Communication (IAIK): *Java Trusted Computing Software Stack (jTSS)*. http://trustedjava.sourceforge.net, created in 2010, accessed at 23.07.2010.

[INF-2010] Infineon Technologies AG: *Trusted Platform Module*. http://www.infineon.com/cms/en/product/channel.html?channel=ff80808112ab681d0112ab6921ae011f, created in 2010, accessed at 16.08.2010.

[JAC-1975] Jackson, Michael: *Principles of Program Design*. Academic Press, London and New York, 1975.

[KUN-2009] Kuntze, Nicolai; et al.: *Trust in the P2P Distribution of Virtual Goods*. In: Alapan Arnab and Jürgen Nützel, editors, Virtual Goods 2009, pages 109-123.

[KUN-2009a] Kuntze, Nicolai; Rudolph, Carsten; *Trust in Distributed small sized Data Centers*. In: Proceedings of the 2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing, 2009, pages 28-33.

[KUN-2010] Kuntze, Nicolai; Rudolph, Carsten; Fuchs, Andreas: *Trust in Peer-to-Peer content distribution protocols*. In: Lecture Notes in Computer Science 6033 2010, pages 76-89.

[LEG-2006] Legout, Arnaud; Urvoy-Keller G.; Michiardi, P.: *Rarest First and Choke Algorithms Are Enough*. In: 6th ACM SIGCOMM, 2006, pages 203-216.

[LEI-2009] Leicher, Andreas; Kuntze, Nicolai; Schmidt, Andreas U.: *Implementation of a Trusted Ticket System*. In: Proceedings of the IFIP Security, 2009.

[LIN-2010] Lincke, Nico: *Entwicklung eines Konzepts für ein sicheres Peer-to-Peer System mit dezentralem Management*. Ongoing thesis declared at the University of Applied Sciences Darmstadt, department of computer science, 2010.

[MAG-2007] [MAG-2007] Magharei, Nazarin; Rejaie, Reza: *Mesh or Multiple-Tree: A Comparative Study of Live P2P.* In: IEEE INFOCOM, May 2007.

[MAR-2008] Martin, Andrew: *The ten-page introduction to Trusted Computing* . www.comlab.ox.ac.uk/files/1873/RR-08-11.PDF, created in November 2008, accessed at 03.06.2010.

[MAH-2007] Mahlmann, Peter; Schindelhauer, Christian: *Peer-to-Peer-Netzwerke: Algorithmen und Methoden.* Berlin: Springer-Verlag, 2007.

[MAY-2002] Maymounkov, Peter; Mazières, David: *Kademlia: A Peer-to-peer Informaiton System Based on the XOR Metric.* In: 1st International Workshop on Peer-to-peer Systems (IPTPS), 2002.

[MUE-2008] Müller, Thomas: *Trusted Computing Systeme* : Konzepte und Anforderungen. Heidelberg: Springer-Verlag, 2008.

[NIC-2010] Australia's Information and Communications Technology Centre of Excellence: *NICTA Home* . http://nicta.com.au, created in 2010, accessed at 21.05.2010.

[NIS-2003] National Institute of Standards and Technology: *Engineering Statistics Handbook* . http://www.itl.nist.gov/div898/handbook/index.htm, created in 2003, accessed at 28.07.2010.

[OMG-2010] Object Management Group: *Unified Modeling Language (UML).* http://www.uml.org, created in 2010, accessed at 21.09.2010.

[ORA-2010a] Oracle Corporation: *Java Cryptography Architecture (JCA) Reference Guide* . http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html, created in 2010, accessed at 23.07.2010.

[RAK-2010] Rakotoarivelo, Thierry: *OMF Developer Portal* . http://mytestbed.net, created at 06.04.2010, accessed at 21.05.2010.

[RAT-2001] Ratnasamy, Sylvia; et al.: *A Scalable Content-Addressable Network.* In: Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM New York, 2001, pages 161-172.

[RSA-1999] RSA Laboratories: *PKCS #5 v2.0: Password-Based Cryptography Standard* . http://rsa.com/rsalabs/node.asp?id=2127, created at 25.03.1999, accessed at 04.10.2010.

[RSA-2002] RSA Laboratories: *PKCS #1 v2.1: RSA Cryptography Standard.* http://rsa.com/rsalabs/node.asp?id=2125, created at 14.06.2002, accessed at 04.10.2010.

[RSA-2010] RSA Laboratories: *Public-Key Cryptography Standards (PKCS)* . http://rsa.com/rsalabs/node.asp?id=2124, created in 2010, accessed at 19.08.2010.

[RUB-2010] Ruby Visual Identity Team: *Ruby Programming Language* . http://ruby-lang.org, created in 2010, accessed 21.05.2010.

[RUT-2010] Rutgers, The State University of New Jersey: *WINLAB : Home* . http://www.winlab.rutgers.edu, created in 2010, accessed at 21.05.2010.

[SAI-2006] Sailer, Reiner; et al IBM Research: *Integrity Measurement Architecture (IMA).* IBM Research, http://domino.research.ibm.com /comm/research_people.nsf/pages/sailer.ima.html, created in 2006, accessed at 04.08.2010.

[SAN-2005] Sandu, Ravi; Zhang, Xinwen: *Peer-to-peer access control architecture using trusted computing technology*. In: Proceedings of the tenth ACM symposium on Access control models and technologies, 2005, pages 147-158.

[SCH-1996] Schneider, Bruce: *Applied Cryptography* . Second Edition: John Wiley & Sons, 1996.

[SHE-2009] Sherman, Alex; et al.: *Adding Trust to P2P Distribuion of Paid Content*. In: Lecture Notes in Computer Science, 2009, Volume 5735/2009, 2009, pages 459-474.

[SIT-2010a] Fraunhofer-Institut für Sichere Informationstechnologie (SIT): *Welcome to the Nanodatacenters Project* . http://www.nanodatacenters.eu/, created in 2010, accessed at 24.06.2010.

[STE-2005] Steinmetz, Ralf; Wehrle, Klaus: *Peer-to-Peer Systems and Applications* . Heidelberg: Springer-Verlag, 2005

[STO-2001] Stoica, Ion; et al.: *Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications.* In: IEEE/ACM Transactions on Networking, Vol. 11, pages 17-32, Febuary 2003.

[STR-2010] Strasser, Mario; Stamer, Heiko; Molina, Jesus: *Software-based TPM Emulator* . http://tpm-emulator.berlios.de, created in 2010, accessed at 23.08.2010.

[STU-2008] Stumpf, Frederic; et al: *Improving the Scalability of Platform Attestation.* Proceedings of the 3rd ACM workshop on Scalable trusted computing, 2008, pages 1-10.

[SUN-2010] SUN Microsystems Inc. : *System (Java Platform SE 6)* . http://java.sun.com/javase/6docs/api/java/lang/System.html#nanoTime(), created at 18.12.2009, accessed at 29.07.2010.

[TAM-2008] Tamami, A. A.: *Performance Analysis of Data Encryption Algorithms.* http://www.cs.wustl.edu/~jain/cse567-06/ftp/encryption_perf/index.html, created in 2008, accessed at 01.10.2010.

[TCG-2010] Trusted Computing Group: *Trusted Computing Group* : Home. http://www.trustedcomputinggroup.org/, created in 2010, accessed at 13.04.2010.

[TCG-2005] Trusted Computing Group: *TCG PC Client Specific TPM Interface Specification (TIS)* : Version 1.2 FINAL. http://www.trustedcomputinggroup.org/files/resource_files/87BCE22B-1D09-3519-ADEBA772FBF02CBD/TCG_PCClientTPMSpecification_1-20_1-00_FINAL.pdf, created at 07.11.2005, accessed at 23.08.2010.

[TCG-2007] Trusted Computing Group: *TCG Software Stack (TSS)*: Specification Version 1.2. http://www.trustedcomputinggroup.org/resources/tcg_software_stack_specification_tss_12_faq, created at 07.03.2007, accessed at 25.09.2010.

[TEC-2010] Technicolor S.A.: *Technicolor - Creation, Management and Delivery of content.* www.thomson.net, created in 2010, accessed at 05.07.2010.

[THO-2010] Thompson (Project NanoDataCenters): *Deliverable D1.2 Evaluation of the energy efficiency of distributed vs. centralised content distribution.* http://www.nanodatacenters.eu/index.php?view=article&catid=48%3Aarchitecture&id=77%3Aevaluation-of-the-energy-efficiency-of-distributed-vs-centralised-content-distribution&format=pdf&option=com_content&Itemid=67, created 30.01.2010, accessed at 20.09.2010.

[THO-2010b] Thompson (Project NanoDataCenters): *Deliverable 2.1 Measurement-based characterisation of application and user behaviour.* http://nanodatacenters.eu/index.php?option=com_phocadownload&view=category&id=3:measurement&Itemid=66, created 30.01.2010, accessed 04.09.2010.

[ULL-2009] Ullenboom, Christian: *Java ist auch eine Insel :* Programmieren mit der Java Standard Edition Version 6. 8.aktualisierte Auflage, Bonn: Gallileo Press, 2009.

[VAR-2008] Varvello, Matteo; et al.: *Is There Life in Second Life?.* In: International Conference On Emerging Networking Experiments And Technologies, 2008, 1st article.

[VTT-2010] VTT Technical Research Centre of Finland: *P2P Next* . http://www.p2p-next.org/, created in 2010, accessed at 03.09.2010.

REQUEST FOR COMMENTS (IETF)

[RFC-1122] Braden, R.: *Requirements for Internet Hosts – Communication Layers.* RFC1122, 1989.

[RFC-1831] Srinivasan, R.: RPC: *Remote Procedure Call Protocol Specification Version 2.* RFC1831, 1995.

[RFC-2616] Fielding, R.; et al.: *Hypertext Transfer Protocol – HTTP/1.1.* RFC2616, 1999.

[RFC-2631] Rescorla, R.: *Diffie-Hellman Key Agreement Method.* RFC2631, 1999.

[RFC-3920] Saint-Andre, P.: *Extensible Messaging and Presence Protocol (XMPP): Core.* RFC3920, 2004.

[RFC-3921] Saint-Andre, P.: *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence.* RFC3921, 2004.

[RFC-3922] Saint-Andre, P.: *Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM).* RFC3922, 2004.

[RFC-3923] Saint-Andre, P.: *End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP).* RFC3923, 2004.

[RFC-4250] Lehtinen, S.; et al.: *The Secure Shell (SSH) Protocol Assigned Numbers.* RFC4250, 2006.

[RFC-5280] Cooper, D.: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC5280.