Performance evaluation in trust enhanced decentralised content distribution networks

Roman Korn
University of Applied Sciences Darmstadt
Darmstadt, Germany
roman.korn@web.de

Nicolai Kuntze and Jürgen Repp
Fraunhofer-Institute for Secure Information Technology SIT
Rheinstraße 75, 64295 Darmstadt, Germany
{nicolai.kuntze|juergen.repp}@sit.fraunhofer.de

***Abstract*—Content Delivery Networks (CDN) have become a key architecture in the provision of today's Internet based services like Video on Demand. A higher degree of the decentralisation in the implementation of CDNs is the next possible step to ensure quality levels but also to be more Internet Service Provider (ISP) friendly in the operation of CDNs. This paper discusses the impact of trust enhancements into peer-to-peer (P2P) based content distribution protocols using the BitTorrent protocol as an example.**

I. Introduction

Distribution of content is a growing area of service delivery. Traditional *Content Delivery Networks* (CDN) are based on servers located in the infrastructure of the various Internet service provider. New, upcoming areas like *video on demand* (VoD) present higher demands towards the CDNs but also to the bandwidth utilisation, especially in the backbone infrastructure.

We expect that decentralized CDNs would be more adequate for upcoming demands in this area. Such decentralized CDNs are deployed to the edge of the ISP network represented by the Internet access point and located at the users household. Such residential home gateways could form a decentralised CDN based on *peer-to-peer* (P2P) protocols.

Establishing infrastructure components at the vicinity of the end user, outside the direct control of the owner always rises the question on the security and reliability of such an approach. Advanced protection mechanisms like Trusted Computing can be used to protect each individual device but also to establish a network where each node is able to determine the trustworthiness of its communication partners. Adversary nodes would be immediately excluded from the network.

In [9] a first approach to integrate Trusted Computing concepts into the well known BitTorrent protocol was presented. Meanwhile the concepts could be implemented. Our solution applies a cryptographic processor providing security relevant operations on very low speed. This paper focuses on first findings regarding the practicability of such a scheme.

The paper is organised as follows. In II the basic principles of Trusted Computing are laid down. Section III presents the modified BitTorrent protocol [1] in more detail. The detailed analysis in presented in section V.

This work was partially supported by the EU FP7 NanoDataCenters project.

II. Trusted Computing Principles

Trusted Computing technology as defined by the *Trusted Computing Group* (TCG) [6] aims to establish consistently behaving computer systems. Such a consistent behavior is enforced by methods checking the integrity of a system and identifying anomalous (unwanted) characteristics. These methods depict a trusted system's base of trust and thus are implemented in hardware, as it is less susceptible for attacks than software pendants.

To successfully realize stringently reliable system components, several mechanisms of a cryptographic processor, namely *Trusted Platform Module* (TPM) [7], are applied. The TPM incorporates strong asymmetric key cryptography, cryptographic hash functions and a random number generator. Additionally each trusted system is equipped with a unique key pair whose private key is securely and irrevocably stored on the TPM. The TPM itself is the only entity able to read and apply this key for e.g. signing or encryption. This concept builds a powerful basement to securely sign and encrypt data of a trusted system. It is used to measure system integrity and to ensure a system is and remains in a predictable and trustworthy state that produces only accurate results.

Trust for Measurement The key concept of Trusted Computing is establishment and extension of trust from an initially trusted security anchor up to the application level. Starting from system boot, the *Secure Hash Algorithm* (SHA-1) digest of each loaded component (e.g. a binary) is computed (measuring). The first component of this cycle acts as security anchor. It has to be trusted, since it's integrity can not be measured in this context. This anchor is called *Core Root of Trust for Measurement* (CRTM).

Each component that is involved in the system boot process measures its successive components using `TPM_EXTEND`. The TPM stores obtained digest values in its local and protected *Platform Configuration Registers* (PCR). This type of system boot is also called *Trusted Boot Process*. Such a process of successively extending trust is known as building a *Chain of Trust*. To reproduce and verify a platform register's value in hindsight, each invocation of the `TPM_EXTEND` operation has to be tracked in a log. During runtime, the trusted operating system stores entries in a file called *Stored Measurement Log* (SML).

Trust for Reporting Another important concept of Trusted Computing is *Remote Attestation*. That process enables an external party to prove trustworthiness of a Trusted Computing System. To verify a system's integrity, a subset of PCRs together with the above-mentioned SML is sent to the external party. In order to ensure integrity of the submitted PCR subset, it is signed by a unique TPM key pair, the *Attestation Identity Key* (AIK).

AIK Certification Each TPM is globally unique and thus identifiable and traceable. Privacy issues arise with the attestation of system states to external parties. In order to avoid this security issue, TPMs can operate pseudonymous. They generate temporary keys for attestation. These *Attestation Identity Keys* (AIK) can be created at any time using the `TPM_MakeIdentity` operation and may be certified by a *Trusted Third Party*. Certificates allow external parties to verify, that an AIK belongs to a TCG conform platform.

0. Set-up previous to protocol.	
$peer \quad p:$	$metafile, (AIK^p_{pub}, AIK^p_{priv})\ , AIKCert^p_{ca}\ , S^t_{pub}$
$tracker\,t:$	$metafile, (S^t_{pub}, S^t_{priv})\ , C^{ca}_{pub}$
1. Peer sends an encrypted request.	
$p \rightarrow t:$	$enc\left\{request, K^p_{pub}, SML^p, AIKCert^p_{ca}\right\}_{S^t_{pub}}$
2. Tracker computes the shared secret.	
$t:$	$K^{p,t} = K^t_{priv} \circ K^p_{pub}$
3. Tracker sends a signature and a public key.	
$t \rightarrow p:$	$sig\left\{hash(K^p_{pub} \| K^t_{pub})\right\}_{S^t_{priv}}\ , K^t_{pub}$
4. Peer sends a quote.	
$p \rightarrow t:$	$quote\left\{hash(K^t_{pub} \| K^p_{pub}), PCR_{0..n}\right\}_{AIK^p_{priv}}$
5. Peer computes the shared secret.	
$p:$	$K^{p,t} = K^p_{priv} \circ K^t_{pub}$
6. Tracker sends the encrypted response.	
$t \rightarrow p:$	$enc\{Data\}_{K^{p,t}}$

TABLE I
EXTENDED TRACKER PROTOCOL (TP)

III. Protocol Implementation

Our implementation of a trustworthy *Tracker Protocol* (TP) and a trustworthy *Peer-Wire Protocol* (PWP) is an extension to the original BitTorrent protocols. The TP enables communication between peers and trackers whereas the PWP is applied between peers. The presented approach focuses on a centralized, trustworthy tracker that appraises the integrity of calling peers and provides download authorization tickets to peers. If permitted, trustworthy peers may share a resource with other trustworthy peers. A group of peers, sharing the same resource, is referred to as swarm. A unique metafile, typically named ".torrent", describes a certain resource (`info`) and refers to a responsible tracker (`announce`). Peers may obtain such a metafile independent from the discussed protocols.

A. Extended Tracker Protocol The extended protocol, as described by [9] [1] , is performed between a peer and a tracker. The protocol accomplishes all tasks of the original version and additionally: mutual authentication in context of the applied keys and the AIK certificate, attestation and appraisal of the peer platform's software integrity, an encrypted communication channel is established, and 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 I and described in detail below. Analogue to the original protocol, a peer sends the fields *peer_id*, *port* and *event* in each transmission. The tracker responds with an *event*. All of these original values are still used, but not considered further in the following description.

(1a) The first message includes values that are encrypted using a public key of the tracker. This key S^t_{pub} has to be obtained from a *Certification Authority* (CA) in advance. Furthermore, the peer system is expected to be equipped with the certificate $AIKCert^p_{ca}$. In field $request$ of message (1) the `info_hash` is transmitted. It is a hashed value of the `info` field within the $metafile$. The transmission of a public key K^p_{pub} is the initial part of the *Diffie-Hellman* (DH) shared secret exchange [3]. 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 $AIKCert^p_{ca}$ is read from a key storage and added to message (1). 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. 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)** 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} 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 set-up of the platform by the ISP. The $request$ is stored until creation of message (6).

(2) 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 to message (6) and subsequent communication steps.

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

(3a) A signature $sig\left\{hash(K^p_{pub}||K^t_{pub})\right\}_{S^t_{priv}}$ and the tracker public key K^t_{pub} have to be transmitted in message (3). 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)** When peer p receives message (3), it stores the tracker key 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. Verification is done using the already available tracker public key S^t_{pub}.

(4a) With message (4) a quote object is 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 TPM using AIK^p_{priv}. 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). This assumption is valid as long as the `TPM_QUOTE` operation is protected from unauthorized execution. Otherwise the key has to be bound to the platform integrity measurement by a `TPM_EXTEND` operation. 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)** The tracker receives the quote object, 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 section II. 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 attestation. 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 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 verification of the quote object, it can be assumed that only peer and tracker possess the shared secret.

(5). 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^p_{pub}. As described by the DH, the peer computes the shared secret key $K^{p,t} = K^p_{priv} \circ K^t_{pub}$, now.

(6a) The last communication step of the TP 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 TP in advance. A data object $data := \left(Address_{pB}, AIKCert^{pB}_{ca}, ticket\right)$ 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 $AIKCert^{pB}_{ca}$ is provided to verify the identity of pB during PWP communication. Moreover, a $ticket := enc\{AIKCert^p_{ca}, resource, time\}_{K^{pB,t}}$ is included in each data object. Such a ticket authorizes peer pB to share the specified $resource$ with the owner of the certificate $AIKCert^p_{ca}$ 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)** This is the final step in TP processing. The peer has to decrypt the message using the shared secret ($dec\{Data\}_{K^{p,t}}$). Elements of the data list are either stored or updated. Further processing is done using the PWP as described in the following sections.

B. Extended Peer-Wire Protocol The extension to the peer-wire protocol is based on a proposal of Kunzte et al. in [9] (see Table II). It accomplishes tasks of the original protocol and mutual authentication of both parties, indirect attestation by a third party token (ticket), an encrypted communication is established (shared secret exchanged), and a download authorization token (ticket) is exchanged and verified.

0. Set-up previous to protocol.

$peer \quad pA : (AIK^{pA}_{pub}, AIK^{pA}_{priv})\ ,$

$\qquad data := \left(Address_{pB}, AIKCert^{pB}_{ca}, ticket\right)$

$peer \quad pB : (AIK^{pB}_{pub}, AIK^{pB}_{priv})\ ,\ K^{pB,t},\ info\_hash$

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

$pA \rightarrow pB : peerID^{pA}\ ,\ K^{pA}_{pub}\ ,$

$\qquad ticket := enc\left\{AIKCert^{pA}_{ca}, resource, time\right\}_{K^{pB,t}}$

2. Peer pB sends a response.

$pB \rightarrow pA : K^{pB}_{pub}\ ,\ quote\left\{hash(K^{pA}_{pub}||K^{pB}_{pub}), PCR_{none}\right\}_{AIK^{pB}_{priv}}$

3. Peer pA sends a request.

$pA \rightarrow pB : quote\left\{hash(K^{pB}_{pub}||K^{pA}_{pub}), PCR_{none}\right\}_{AIK^{pA}_{priv}}$

4. Both peers compute the shared secret.

$pA : \qquad K^{pA,pB} = K^{pA}_{priv} \circ K^{pB}_{pub}$

$pB : \qquad K^{pA,pB} = K^{pB}_{priv} \circ K^{pA}_{pub}$

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

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

TABLE II
EXTENDED PEER-WIRE PROTOCOL

Our protocol implementation embeds the enhancement between handshake request and response of the original protocol. Accordingly, message (1) serves as initial handshake and message (5) 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.

(1a) Based on the tracker response of the extended TP, peer pA receives 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 DH. 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)** Peer pB receives the first message that 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 TP 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. Tickets serve as evidence, here. 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) 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_{priv}^{pB}. Two concatenated and hashed keys (K_{pub}^{pA},K_{pub}^{pB}) are provided as nonce. PCRs are not included (PCR_{none}) since attestation is not necessary as long as the unauthorized access to the `TPM_QUOTE` operation is assured. The ticket from the trusted tracker vouches for the integrity of both peers. **(2b)** When peer pA receives message (2) the key K_{pub}^{pB} is stored and the quote object is verified using the public key AIK_{pub}^{pB} that is included in the certificate $AIKCert_{ca}^{pB}$. 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_{pub}^{pA},K_{pub}^{pB}). Herewith, it is ensured that peer **pB** has received the authentic key K_{pub}^{pA} and that peer pA has received the authentic key K_{pub}^{pB}.

(3a) With message (3) a single quote object is transmitted to peer pB. This quote object is signed using the private key AIK_{priv}^{pA} of pA. It contains the hash of two concatenated keys (K_{pub}^{pB},K_{pub}^{pA}). 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)** Peer pB receives message (3) and verifies the signature using the public key AIK_{pub}^{pA} of pA. This key is obtained from the certificate $AIKCert_{ca}^{pA}$ that was received within the $ticket$ of message (1). It is not necessary to verify the signature of certificate $AIKCert_{ca}^{pA}$, 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_{pub}^{pB} and that the public key K_{pub}^{pA} received in message (1) is authentic and not altered. For that purpose, the local keys K_{pub}^{pB} and K_{pub}^{pA} 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) Now, both parties compute the shared secret and complete the DH key exchange. Peer pA computes $K^{pA,pB} = K_{priv}^{pA} \circ K_{pub}^{pB}$ and peer pB computes $K^{pA,pB} = K_{priv}^{pB} \circ K_{pub}^{pA}$. All following communication is encrypted by the symmetric key $K^{pA,pB}$.

(5a) 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)** 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.

IV. Experimental Set-up

Goal of the experiments is to compare the processing time of the original protocol implementation with the extended version in order to evaluate the applicability of our approach. Only the establishment of a trustworthy communication is considered, here. We do not consider the actual download time, since the exchange processing for contents is not amended and the communication over an encrypted channel does not differ from other applications. Each protocol step and selected algorithms (e.g. SML verification, TPM operations, key creation) were measured in detail. Measurement points are introduced within tracker and peer application itself. At each measurement point a time stamp is obtained from the java system environment (nanoTime) and printed to an experiment log file. The time elapsed between two measurement points 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.

Experiments are conducted in an isolated Trusted Computing test bed. From a pool of network nodes, only three (tracker t, peer pA and peer pB) are involved in these first experiments. Each node of the experimental network consists of the same physical hardware components: an Intel Atom Z530 1.60 GHz with a single core / two threads and 32-bit instruction set, 2GB DDR2 RAM, Infineon TPM Version 1.2.1.2 operating from a 33 MHz clock, Intel 82574L Gigabit Network Connection and a SAMSUNG M6 Series HM320JI hard disk drive with 5400 revolutions per minute. On all nodes the operating system Ubuntu 10.10 with kernel version 2.6.31.22 is installed. Other involved software applications are: IAIK/OpenTC [5] jTSS Wrapper version 0.4beta, Trousers version 0.3.1-7ubuntu3. The application under test will be executed within a Java Virtual

Machine: 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 Just-in-Time compiler [2] and garbage collection are active for all experiments.

These results are based on few KHL/SML entries, but we could observe that the described verification approach scales properly for larger sizes. Another influencing variable is the swarm size that increases the overall creation time of the list of tickets. In practice however, tracker respond only a limited amount (about fifty) contacts to peers within the same swarm. With a single ticket creation time in the area of microseconds the swarm size seems to be insignificant. Of cause the configuration of cryptographic algorithms and key lengths have a major influence to the processing time. Our configuration is shown in table III.

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

TABLE III
APPLIED CRYPTOGRAPHIC KEYS.

V. COMPARING ORIGINAL AND EXTESION

Results of two experiment sets, the extended and original version, are presented below. Each is based on 100 experiments. For both versions, only the initial protocol execution is considered. If a peer is interested in another resource, it must perform the TP and the PWP again. Reusing information, like shared keys, might lead to a better performance in following executions. It depends amongst others on the decision, how long the tickets respectively attested states are valid. These concepts however, are not considered here. After the described protocols steps, subsequent messages are transmitted equal to the original specification. The TP communication includes e.g. keep-alive whereas much more are defined in the PWP. They are used to exchange available pieces of the resource and the actual resource. Such subsequent messages are excluded from evaluation. They are equal to the original protocol, except that they must be encrypted and decrypted, now.

With the `announce` request, a peer pA announces to the tracker that it wants to become member of a swarm. Therefore the resourceID is transmitted. In response it receives a list of swarm members. The trustworthy protocol accomplishes the tasks

authentication, attestation, encrypted connection and authorization. In Figure 1 the results of two experiments are shown. Figure (a) depicts the result of the extended BitTorrent application whereas Figure (b) depicts the results of the original BitTorrent application, called jBitTorrent [4]. The *extended tracker protocol* (tBitTorrent) is performed between pA and the tracker. In table IV the results for each measure are shown.

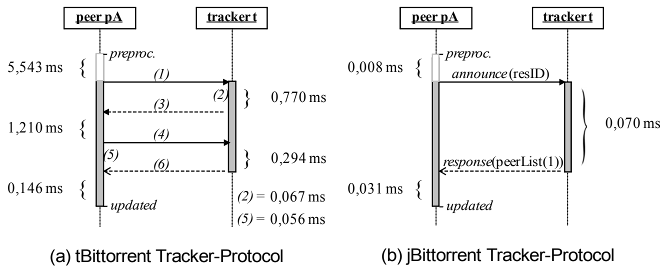


Fig. 1. Comparing TP `announce`. (MEAN, $1s = 1,000ms$)

(a) tBitTorrent: 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 pA's *service processing time* (SrvPT) 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. 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 TP is *SrvPT t,* since many peers are expected to contact a single tracker and therefore it has to be executed more often.

mp_t	mp_{t+i}	MIN	MEAN	MAX	STD	P90
preproc.	*send(1)*	1,646	5,543	17,786	3,558	10,372
recv(1)	*send(3)*	0,596	0,770	0,935	0,073	0,853
recv(3)	*send(4)*	1,042	1,210	2,282	0,154	1,321
recv(4)	*send(6)*	0,198	0,294	0,441	0,050	0,366
recv(6)	*updated*	0,131	0,146	0,226	0,019	0,174
begin(2)	*end(2)*	0,038	0,067	0,112	0,020	0,090
begin(5)	*end(5)*	0,049	0,056	0,104	0,007	0,063
Quo(pA)	*Quo(pA)'*	0,386	0,453	0,551	0,064	0,544

TABLE IV
(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 V the results for each measure are shown.

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

TABLE V
(B) TRACKER PROTOCOL RESULTS OF JBITTORRENT ($1s = 1,000ms$).

In Figure 2 the original and the extended PWP implementations are compared. With the `handshake` message, a peer requests to share a resource with another peer. The extension accomplishes additional tasks as previously described.

(a) tBitTorrent: The *extended peer-wire protocol* (tBitTorrent) is performed between pA and pB. It requires (see protocol I) four transmissions $\{(1), (2), (3), (5)\}$ and two

processing steps $\{pA\,(4)\,, pB\,(4)\}$. In context of this scenario, pA completes the protocol when the handshake response is received. Afterwards both parties may exchange asynchronous messages according to the original protocol, but over an encrypted channel. The decryption of the last message is excluded from measuring. In table VI the results for each measure are shown.

mp_t	mp_{t+i}	MIN	MEAN	MAX	STD	P90
updated	*send(1)*	0,419	4,205	23,834	3,726	7,408
recv(1)	*send(2)*	0,946	0,977	1,020	0,014	0,997
recv(2)	*send(3)*	0,974	1,092	1,897	0,157	1,142
recv(3)	*send(5)*	0,070	0,080	0,130	0,010	0,091
beg-pA(4)	*end-pA(4)*	0,020	0,032	0,063	0,004	0,037
beg-pB(4)	*end-pB(4)*	0,023	0,024	0,047	0,003	0,025
Quo(pA)	*Quo(pA)‘*	0,368	0,384	0,414	0,009	0,394
Quo(pB)	*Quo(pB)‘*	0,370	0,413	0,558	0,049	0,536

TABLE VI
(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 response is received. In table VII the results for each measure are shown.

mp_t	mp_{t+i}	MIN	MEAN	MAX	STD	P90
updated	*send(hs)*	0,021	0,041	0,080	0,009	0,050
recv(hs)	*send(hs)*	0,005	0,009	0,025	0,004	0,015

TABLE VII
(B) PEER-WIRE PROTOCOL RESULTS OF JBITTORRENT ($1s = 1,000ms$).

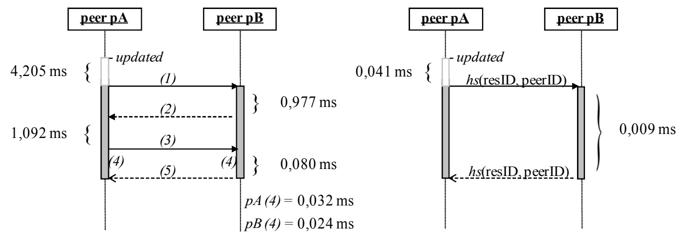
All measures of the extended application (tBitTorrent) are with less performance than the original application (jBitTorrent) as shown in Figure 2. This is caused by the security extensions of the PWP. A strong deviation (STD) of more than 3 seconds can be observed for the measure SrvPT *pA*. It occurs during preprocessing of the protocol $(updated, send(1))$ and it is caused by the same cryptographic library that is used during TP execution. 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 P2P model and contrary to the Client-Server model.

Both extended protocols can be expected to be performed in about 2 seconds each, assuming the parameters described previously. 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 (*Quo(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.

VI. CONCLUSION

Within this paper an evaluation of the extended protocol implementation is presented with the main intention to quantify its performance. The process of measuring is completely automated (using OMF[10]) 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. From the 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;

Our tBitTorrent application confirms, that the proposed protocols can be implemented on the basis of state of the art technologies. These protocols provide a solution to security concerns of commercial P2P networks. It has to be noted however, that the solution cannot be applied in the given business field without the context of a *Nano Data Centers* (NaDa)[8] like system. The NaDa system is now equipped with one of its main components that allows to study basic scenarios.



(a) tBittorrent Peer-Wire-Protocol (b) jBittorrent Peer-Wire-Protocol

Fig. 2. Comparing PWP handshake. (MEAN, $1s = 1,000ms$)

REFERENCES

[1] Bram Cohen. The bittorrent protocol specification (version 11031), 2008. http://bittorrent.org/beps/bep_0003.html.
[2] T. Cramer, R. Friedman, T. Miller, D. Seberger, R. Wilson, and M. Wolczko. Compiling Java just in time. *Micro, IEEE*, 17(3):36–43, 2002.
[3] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on information Theory*, 22(6):644–654, 1976.
[4] Baptiste Dubuis. Jbittorrent, 2011. http://jbittorrent.sourceforge.net/.
[5] Institute for Applied Information Processing and Communication (IAIK). Trusted computing for the java(tm) platform, 2011. http://trustedjava.sourceforge.net.
[6] Trusted Computing Group, 2011. http://www.trustedcomputinggroup.org.
[7] Trusted Computing Group. Tpm 1.2 main specification, 2011. http://www.trustedcomputinggroup.org/resources/tpm_main_specification.
[8] N. Kuntze, J. Repp, M. May, F. Picconi, and R. Teixeira. Trust in the P2P Distribution of Virtual Goods. *Virtual Goods*, pages 109–123, 2009.
[9] Nicolai Kuntze, Andreas Fuchs, and Carsten Rudolph. *Lecture Notes in Computer Science 6033*, chapter Trust in Peer-to-Peer Content Distribution Protocols, pages 76–89. Springer, Berlin / Heidelberg, Germany, 2010.
[10] Australia's ICT Research Centre of Excellence (NICTA). Omf developer portal, 2011. http://mytestbed.net.