OULUN YLIOPISTO
UNIVERSITY of OULU

DEPARTMENT OF ELECTRICAL AND INFORMATION ENGINEERING
DEGREE PROGRAMME IN ELECTRICAL ENGINEERING

ENHANCED PEER DISCOVERY FOR MULTIACCESS PEER-TO-PEER NETWORKS

Thesis author	______________________________ Teemu Rautio
Thesis supervisor	______________________________ Mika Ylianttila
Approved	_______/_______2010
Grade	______________________________

Rautio T. T. (2010) Enhanced Peer Discovery for Multiaccess Peer-to-Peer Networks. University of Oulu, Department of Electrical and Information Engineering. Master's Thesis, 78 p.

ABSTRACT

This thesis includes the design, implementation and evaluation of a multiaccess peer-to-peer content distribution prototype based on BitTorrent. The prototype is designed to enhance the current BitTorrent in terms of download duration and traffic locality in an overlapping multiaccess environment.

The thesis begins with related state of the art including future ways of networking as well as current state of peer-to-peer networks, object resolution and mobility management. Experiments with conventional BitTorrent show the importance of early stage peer discoveries by the BitTorrent tracker. We therefore enhanced the peer discovery process of BitTorrent and divided the functions of the tracker hierarchically into two layers. The higher layer global tracker keeps track of local trackers, while the local trackers maintain a list of peers in their area of the network.

The peer in our prototype retrieves the content locally from its own network whenever possible with extended BitTorrent signalling. While retrieving the content in an overlapping multiaccess environment, the mobile peer in the prototype recognises also a new connection to other networks. It is able to use this connection simultaneously with the previous one.

The prototype implementation and evaluation show the feasibility of multiaccess in the content distribution prototype and present the advantages of employing the multiaccess prototype instead of conventional BitTorrent in overlapping multiaccess environments in static and pedestrian mobility scenarios. The inter-network traffic was also reduced to almost zero with the prototype in both scenarios.

Key words: Future Internet, Information-Centric Networking, Traffic Locality, Peer-to-Peer, BitTorrent, Mobility.

Rautio T. T. (2010) Parannettu vertaisten löytäminen monipääsyvertaisverkossa. Oulun yliopisto, sähkö- ja tietotekniikan osasto. Diplomityö, 78 s.

TIIVISTELMÄ

Tämä diplomityö sisältää BitTorrent vertaisverkkotekniikkaan perustuvan monipääsyä tukevan sisällönjakoprototyypin suunnittelun, toteutuksen ja suorituskyvyn arvioinnin. Kyseinen prototyyppi on suunniteltu olemaan nykyistä BitTorrent vertaisverkkoa parempi langattomissa ympäristöissä, joissa verkkojen kattavuuden päällekkäisyys on yleistä niin tiedoston latausnopeudessa kuin liikenteen paikallisuudessakin.

Työ alkaa aiheeseen liittyvän teorian käsittelyllä, joka sisältää muun muassa tulevaisuuden suuntia tietoliikenneverkkojen tutkimuksessa. Työ käsittelee myös tietokoneverkkojen nykytilaa vertaisverkkotekniikoiden, objektin paikkatiedon ratkaisemisen ja liikkuvuuden hallinnan osalta. Perinteisellä BitTorrent asiakasohjelmalla tehdyt mittaukset osoittivat latauksen alkuvaiheessa vertaisten jäljityspalvelimelta löydettyjen vertaisten tärkeyden. Tämän vuoksi prototyypissä päätettiin keskittyä parantamaan BitTorrent järjestelmän vertaisten löytämisprosessia. Prototyypissä vertaisten jäljityspalvelimen toiminnallisuus jaettiin hierarkkisesti kahdelle eri tasolle. Ylemmän kerroksen palvelimet pitävät yllä listaa alemman tason palvelimista ja alemman tason palvelimet pitävät yllä listaa vertaisista omalla verkkoalueellaan.

Laajennetun BitTorrent järjestelmän signaloinnin avulla vertainen lataa halutun sisällön aina paikallisesti omasta liittymisverkostaan, jos se vain on siellä saatavilla. Liikkuvan vertaisen ladatessa sisältöä se huomaa yhteyden uuteen verkkoon ja pystyy käyttämään sitä samanaikaisesti edellisen yhteyden kanssa.

Prototyypin vertailu perinteisen BitTorrent järjestelmän kanssa osoitti, että prototyyppi suoriutuu tiedoston latauksesta huomattavasti nopeammin päällekkäisiä langattomia verkkoja sisältävässä ympäristössä kun käyttäjä ei liiku. Myös liikkuvassa skenaariossa prototyyppi suoriutui selvästi paremmin kuin BitTorrent käyttäen Mobile IP:ä. Lisäksi molemmissa tapauksissa lataajan aiheuttama verkkoliikenne pysyi lähes kokonaan oman verkon rajojen sisäpuolella.

Avainsanat: tulevaisuuden Internet, informaatiokeskeinen tietoliikenneverkko, liikenteen paikallisuus, vertaisverkko, BitTorrent, monipääsy, liikkuvuus.

TABLE OF CONTENTS

FOREWORD

The purpose of this thesis was to provide practical evaluation results for ongoing future Internet-related research. The thesis concentrates on data dissemination networks by implementing a content distribution prototype based on the current peer-to-peer system.

This thesis was carried out at VTT Technical Research Center of Finland as a part of future Internet research within the ICT 7th Framework Integrated Project 4WARD and the Finnish Strategic Centres for Science, Technology and Innovation (ICT SHOK) Programme “Future Internet” project. In these projects, the thesis provides implementation and proof-of-concept prototyping evaluation for key aspects of information-centric networking. The study of these future Internet-related issues started back in April 2008 when I came to VTT. The early stages of the work were spent familiarizing myself with current peer-to-peer networks and computer networks in general, through literature and learning in practice. During my studies, I worked as a part- and full-time research trainee until the beginning of June 2009 when I started this thesis.

The results of the thesis are also summarized in an accepted conference paper: “A Multiaccess Network of Information” [1] in the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2010 (WoWMoM 2010) and also in a submitted conference paper: “Multiaccess NetInf: a prototype and simulations” [2] in The 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities 2010 (TridentCom 2010).

I would like to thank my advisors Kostas Pentikousis and Jukka Mäkelä for their valuable guidance during the thesis, and my other colleagues at VTT, especially Markus Luoto and team leader Jyrki Huusko, for giving me help and comments. I would also like to thank my thesis supervisor Professor Mika Ylianttila.

Special thanks also go to my family for its valuable support throughout my studies and life in general. Finally, I would like to extend a great thank you to Susanna for her support and understanding.

Oulu, March 26, 2010

Teemu Rautio

LIST OF ABBREVIATIONS AND SYMBOLS

3G/LTE	3rd Generation family of standards for mobile telecommunications
3GPP	3rd Generation Partnership Project
3GPP2	3rd Generation Partnership Project 2
ABC	Always Best Connected scenario
ALTO	Application Layer Traffic Optimization
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency Network
AS	Autonomic System
BO	Binary Object
CCN	Content Centric Networking
CDN	Content Distribution Network
CN	Corresponding Node
CNL	Converging Networks Laboratory
CoA	Care-of Address
DHT	Distributed Hash Table
DISM	Distributed Swarm Management
DNS	Domain Name System
DO	Data Object
FIB	Forwarding Information Base
HA	Home Agent
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HUI	Hierarchical URI-like identifier
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IO	Information Object
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRTF	Internet Research Task Force
ISP	Internet Service Provider
L1	Physical layer at TCP/IP model
L2	Data link layer at TCP/IP model
L3	Network layer at TCP/IP model
L4	Transport layer at TCP/IP model
L5	Application layer at TCP/IP model
LAN	Local Area Network
MA	Multiaccess
MAC	Media Access Control
MICS	Media Independent Command Service
MIES	Media Independent Event Service
MIH	Media Independent Handover service
MIHF	Media Independent Handover Function
MIHU	Media Independent Handover User
MILSA	A Mobility and Multihoming Supporting Identifier Locator Split Architecture for naming

MIP	Mobile IP
MIPv6	Mobile IP version 6
MN	Mobile Node
NAT	Network Address Translation
NES	Network Expertise System
NetInf	Network of Information
NNS	NetInf Notification Service
NNSP	NetInf Notification Service Point
P2P	Peer-to-Peer
P4P	Provider portal for applications
PEX	Peer exchange
PIT	Pending Interest Table
RRG	Routing Research Group
RZBS	Realm-Zone Bridging Server
SA	Single Access
SAP	Service Access Point
SLD	Second Level Domain
TCP	Transmission Control Protocol
TLD	Top Level Domain
TRG	Mobility Trigger Management
UDP	User Datagram Protocol
URI	Uniform Resource Identification
URL	Uniform Resource Location
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
VoCCN	Voice over Content Centric Networking
XML	eXtensible Markup Language

1. INTRODUCTION

Today’s fast-growing Internet has made trustworthy service provision and network management extremely challenging. Nowadays, the majority of Internet traffic is generated by content distribution, especially by peer-to-peer file-sharing systems [3]. The availability of portable devices with Internet connectivity has raised expectations of network connection availability wherever and whenever, even with high mobility. The principles of the current Internet were developed almost half a century ago [4], however, when devices were static in terms of mobility, and communication was based on conversation between two specific machines. In those days, the main consideration was the definition of the end-host location [5]. As users are not usually interested in the location of the content any more, and the Internet is not currently used as it was designed to be used, researches are developing a new content- [5] or information-centric networking paradigm [6]. In this paradigm, the location of the content is left behind and most of the consideration is given to the content itself.

Current peer-to-peer file-sharing systems such as, for example, BitTorrent, provide a good approach to this new paradigm. In peer-to-peer file-sharing, the content is copied across a network to a large set of peers that distribute their resources (e.g., network bandwidth and storage) to other peers. This way, a newcomer peer downloads the content directly from other peer(s) and the traditional server location is not needed for downloading the content. Content distributor peer(s) in peer-to-peer systems are usually picked at random, however, and the network topology information is completely forgotten [7]. In practice, the content is retrieved from a random place, even though it could be available in the own network. Internet Service Providers are paying for generated traffic that crosses the boundary of their administrative network area(s). Random peer selection can therefore lead to extra costs for them. If we can minimize generated cross-traffic, this development could also affect customer prices. Moreover, if we could always retrieve the content from the closest possible source, even from our own network, this could affect the user experience through reduced download durations.

In today’s Internet, applications such as BitTorrent require the employment of mobility solutions when the user moves across heterogeneous wireless networks. Existing state-of-the-art mobility solutions are not necessarily completely suitable for the needs of location-aware, information-centric networks, however, on which users should be able to take full advantage of overlapping wireless networks (which are already commonly available) whenever they are available. If, for example, the user is within coverage of two wireless local area networks at the same time, these could be used simultaneously. With state-of-the-art mobility solutions, this is not possible. In principle, this multiaccess operation could reduce download durations by as much as half, which is a major advantage.

The main objective of this thesis is to design and implement an information-centric content distribution prototype based on BitTorrent that outperforms the current BitTorrent in terms of mobility and multiaccess, and to try to retrieve the content as near from a network that is possible. The locality of the traffic is made possible by implementing a hierarchical structure of the peer-to-peer network and extending BitTorrent signalling. The prototype takes full advantage of overlapping wireless networks, moving across them by employing them simultaneously whenever possible. Mobility support for the prototype could be provided with, for example, Mobile IP. A new network, however, is indicated using the Mobility Trigger

Management architecture [8]. As we implemented the prototype, we compared it with the current BitTorrent using experimental measurements.

In this thesis, we consider extending the current BitTorrent qualities. We do not implement a complete peer-to-peer system but a proof-of-concept prototype for which the main considerations are: traffic locality, mobility and the benefits of simultaneous multiaccess in content distribution. This work also provides valuable feedback for the work on information-centric networking and, to be more specific, for the Network of Information [9].

The thesis is organized as follows. Chapter 2 presents an information-centric network paradigm by introducing principles of current host-centric networking. Chapter 3 briefly introduces current information object resolution as well as two proposed solutions for the future Internet. Chapter 4 presents peer-to-peer overlay networks with consideration for file sharing. The main emphasis is given to the BitTorrent file-sharing system. Chapter 5 introduces a multiaccess environment and a few existing mobility solutions for IP networks. The implemented multiaccess content distribution prototype is described in Chapter 6, while Chapter 7 provides an evaluation comparing it with conventional BitTorrent and BitTorrent over Mobile IP. Finally, Chapter 8 concludes the thesis.

2. INFORMATION-CENTRIC NETWORKS

The basic principles of networking and the architecture of the current Internet were developed in the sixth and seventh decades of the last century. In those days, networking was mainly used for resource sharing purposes, for example, mainframe CPU sharing, and interaction was mostly based on communication between two machines. Host-centric networking based on TCP/IP was developed following these conditions. The costs of machines and storage have now fallen significantly, however, and the need for traditional conversation with a specific machine is uncommon. Users of the Internet are no longer interested in the source location of the content but in the content itself. Content delivery is still based on mapping the content to the location, whereas the whole architecture of the Internet could be based on information and, consequently, the binding from what to where could be removed. This new way of thinking about networking can be called information- or content-centric networking.[5][6]

We present two examples of this new networking paradigm in this chapter. First, however, we introduce the principles of the current host-centric Internet in Section 2.1. The basic principles of Content-Centric Networking, which is implemented by loosely following the ideas of the Internet, are presented in Section 2.2. Finally, Section 2.3 introduces a totally new way of networking, namely the Network of Information.

2.1. Internet Protocol Stack

The protocol suite used in today's Internet is called TCP/IP. The model was developed for the Advanced Research Projects Agency Network (ARPANET). In those days, the employment of heterogeneous transmission mediums led to the development of new protocols, namely TCP/IP. Afterwards, a new architecture, which we now know as the TCP/IP model or Internet Protocol Stack, was developed around these protocols. The name of the model comes from its two main protocols: the Transmission Control Protocol (TCP) and the Internet Protocol (IP). It is divided into five different layers, as shown in Figure 1.[10][11]

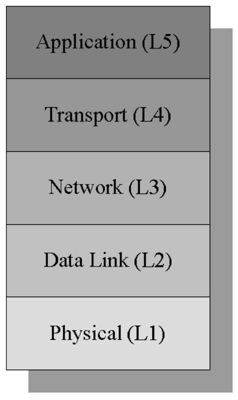


Figure 1. TCP/IP reference model.

In the TCP/IP model, each layer performs a relatively independent task. The name of first layer (L1) of the model from downside up is the physical layer. The physical layer takes care of bit-level communication between two different devices, for example, two personal computers or a personal computer and a wireless local area network (WLAN) access point. It defines the characters as a transmission medium (copper, optic, radio frequency, microwave, etc.), signal forms, modulation methods, bit rates and so on. The second layer is the data link layer (L2). The main purpose of this layer is to deliver data, in the case of the Internet – frames – between end systems within the same network (for example, inside the local area network, LAN). The data link layer has to handle the construction of frames and the routing of these inside the network. For routing purposes, MAC addresses are used to locate the destination of the frame. There are several named standards, for example, the IEEE 802.3 Ethernet and IEEE 802.11 Wi-Fi for local area networks.[12]

The interconnection between two subnets is made by the network layer (L3). The transmission units at the network layer are packets. The third layer basically adds some packet information to the frames and moves the packets between the end systems in different networks independently in a way that is also known as routing. The protocol used in the Internet is the Internet Protocol, currently version 4 (IPv4). For future purposes, IPv6 has been announced and will be a step forward, mainly due to its bigger address space (32-bit address vs 128-bit address).[11][12]

The transmission layer (L4) is the fourth layer in the reference model. It allows conversation between two peer entities on the source and destination host. In the Internet, two different protocols implement the tasks of the transport layer: TCP and UDP. TCP allows reliable connection between entities running in a specified TCP port using segments, while UDP just sends datagrams (IP packet + header) to the particular port on the particular host and does not worry about losses on the way. The application layer sits on top of the reference model and constructs a pair IP address – TCP/UDP port namely as a socket. The socket is an interface that allows conversation between two user programs on different machines. The number of protocols in the application layer is huge, but the most popular ones are BitTorrent and Hypertext Transfer Protocol (HTTP).[1][11][12]

2.2. Content-Centric Networking

Content-Centric Networking (CCN), developed by the Palo Alto Research Center, is an architecture for communications built on named data. In CCN, certain location-based packet addresses (IP addresses) are replaced with the names of the content, and CCN does this without losing the simplicity, robustness and scalability of TCP/IP. The protocol stack of CCN (in Figure 2) is similar to that of TCP/IP in several ways. Agreements between nodes at layer 2, and between the producer and consumer at layer 4 exist in both stacks (cf. Figure 1 and Figure 2). In both protocol stacks, the network layer (L3) is also the only layer that needs universal agreement. Due to the success of the Internet Protocol, CCN’s network layer is similar to that of the IP. The biggest difference from TCP/IP is the addition of two new layers on both sides of the network layer called Strategy (below) and Security (up). In CCN, communication can be built over, for example, IP, UDP or P2P (L2), whereas TCP/IP is built over, for example, lower-level Ethernet. The application layer (L5) of CCN is similar to that of the TCP/IP model however.[5]

CCN has interest and data type packets. The interest packet is used for requesting content by broadcasting it to other nodes. When a node receives an interest message and has the requested content, it can send the data packet back to the consumer node.

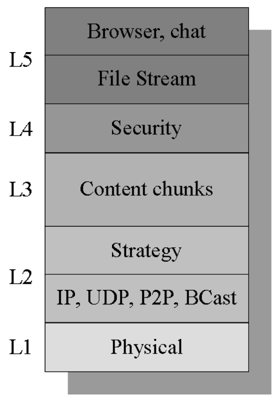


Figure 2. CCN Protocol Stack with the corresponding TCP/IP layer numbers on the left.

The match between the interest and data packet is performed through specific naming. The content name in the interest should be the same as a prefix of the content name in the data packet. Overall, CCN uses hierarchical names in a similar manner to IP with addresses. The packet-forwarding engine of CCN contains three different main data structures: the Forwarding Information Base (FIB), the Pending Interest Table (PIT) and the content store. The arriving packet comes to a face (name used for the interface), a longest match lookup is performed on the data name and, if the node has the content (in the content store), it is delivered to the consumer node. If the node does not have the content, however, the FIB is used to deliver the interest to the potential source node(s), and the name is stored in the PIT. The interest packets are the only packets routed in CCN, but they leave behind PIT entities to all the nodes that contain the name of the content and the id of the requesting face. By using these footprints, the data can be delivered to the requesting node, as the intermediate nodes can act as caches. Unlike TCP/IP, in CCN the content can be delivered with multipoint many-to-many communication.[5]

CCN is built on top of unreliable packet delivery, and unsatisfied interests must therefore be retransmitted within a reasonable time. A responsible retransmitting node of interest is the final consumer that ordinarily wants the content. In CCN, flow control and sequencing are implemented using interest messages, as in TCP, which uses acknowledgements. In CCN, all communication occurs between two local peers, and flow control is therefore only maintained at each hop and does not need additional techniques to maintain middle-path congestion. In CCN, sequencing is implemented in a similar manner to that in TCP but the actual content is segmented, and the segment number in the interest message corresponds to the TCP sequence number. In addition, CCN can operate over a new link in a mobile manner as quickly as it is physically possible to perform so because it has no binding between layer 3 and layer 2 identities.[5]

Routing schemes to CCN can be borrowed directly from IP because its forwarding model is a superset of the IP model. The core idea of most of the routing transport protocols is also very similar to the CCN flooding model (described in the previous paragraph). CNN can use conventional routing as well as operating on top of an existing infrastructure, due to similarities in routing. Security issues also play a big role in CNN and are constructed on term content-based security. This security capability allows all content to be digitally signed, and private content to be encrypted.[5]

A CCN prototype implementation cannot achieve the performance of TCP for bulk data transfer, but the advantages of CNN include security. CNN-over-Ethernet performs as well as HTTP, and twice as well as Secured HTTP (HTTPS). CNN-over-jumbo-UDP is twice as good as HTTP and three times better than HTTPS. Another definite advantage of CNN is performance in the case of multiple file downloaders. While the download duration of TCP increases linearly after the addition of a downloader, CNN's duration stays at the same level. Voice over Content-Centric Networks (VoCCN) also performs quite well.[5][13]

2.3. Network of Information

A Network of Information (NetInf) is an information-centric networking architecture developed by the 4WARD project [14]. The main aim of NetInf is to find alternative architectures for the future Internet [6]. NetInf is currently still under development and there are no ready-made solutions or implementations of the architecture. Several different simulation studies and prototypes, however, show that NetInf is on the right track and looking at new directions of networking, as seen in, for example, [9].

The current architecture of the Network of Information, from the view of a NetInf node, is divided into three levels. The lowest level of architecture is the transport control level, which is also divided into a transport control engine and cache engine entities. The function of the transport control engine is to coordinate the use of different protocols for accessing NetInf objects. The cache engine takes care of objects stored temporally in a local system. The transport control level can run on top of, for example, TCP/IP, IP multicast or totally new generic paths (path abstraction from the 4WARD project). The middle level of the NetInf node architecture is the name resolution level. The name resolution level offers resolution of object identifiers to locators or binary objects and involves entities such as a resolution engine, local storage engine and local resolution engine. The name resolution level also offers an Application Programming Interface (API) to user applications with some specific networking functions. More information about this resolution level can be found in Section 3.3. The top level of the three is NetInf additional services. Its purpose is to provide optional add-on services that can be used by, for example third-party service providers. The additional services level is split into two engines: storage and app x engines. Both engines have APIs to applications named in the same way. Each level of the architecture corresponds to its own protocol that communicates with other NetInf nodes. This communication is performed through the NetInf Information Network Interface.[9]

The NetInf defines three different scenarios with a significant information-centric context: content distribution, augmented Internet and personal mobile communications. These independent scenarios present the aspects that could be most relevant to the future NetInf and they also point to cases in which the benefits of NetInf could be certain. The first and last scenarios are most relevant to this thesis. In the context of the content distribution scenario, one of the main objectives is to retrieve the content locally whenever possible, even without a connection to the rest of the Internet. In a mobile scenario, for one, the connection to the network should be available whenever physically possible.[6][9]

3. INFORMATION OBJECT RESOLUTION

On today’s TCP/IP-based Internet, information object resolution is mainly built on the Uniform Resource Identifier and Domain Name System [15]. The movement from host-based to information-based networking and, more generally, from current to new, future Internet architectures also involves new design considerations and mechanisms for object resolution.

First, this chapter presents Domain Name System-based object resolution in Section 3.1 as a basis. BitTorrent’s object resolution is mainly based on the Uniform Resource Identifier and Domain Name System resolutions and is presented briefly in Section 3.2. The information-centric approach to future networking also involves new ideas for object resolution. The Network of Information is one proposal for future name resolution problems and is discussed in Section 3.3. Finally, Section 3.4 illustrates the future resolution architecture and resolution proposal named MILSA, but still follows the host-centric networking paradigm.

3.1. Domain Name System

The Domain Name System (DNS) is a major part of Uniform Resource Identifier (URI) resolution and its assignment is to convert unique human-readable domain names into IP addresses. URI resolution resolves at least the meaning of the transport, end-point and server-side location of the object. The DNS part of this chain is the end-point resolution.[11][15]

Domain name to IP address conversion is performed by a hierarchical, distributed database that consists of four levels of name servers: root, top-level domain (TLD), second-level domain (SLD) and local, in the top to bottom direction. When the user wants to visit, for example, a web page at *http://www.vtt.fi*, the user’s resolver first contacts the local name server and requests the IP address of *vtt.fi*. The local name server contacts one of the root name servers and delivers the conversion question to it. The root name server returns a list of TLD name servers that corresponds to the national domain *.fi*. The local name server then asks for the IP address of *vtt.fi* from one of the TLD name servers and the TLD returns a list of SLD name servers that corresponds to *.vtt.fi*. Finally, the local name server asks the same question of the SLD name server. The SLD returns the IP address of *vtt.fi* to the local name server, which returns the address to the resolver. This way of operation by the DNS is called recursive name resolution. Another approach to name resolution is iterative DNS in which the client aka resolver performs all the queries in place of the local name server.[15][16]

3.2. BitTorrent

BitTorrent (described in Section 4.5) has three different levels of resolution [15]. The client has to include its own IP address in the formulation of the tracker announce message. In principle, the client must resolve the hostname, describing itself to the IP address. Next, the tracker’s location is presented in .torrent in URI form as described briefly in Section 4.5.1. Contact with the tracker therefore first requires URI resolution, and it may also require the invocation of the DNS if the server location is revealed in domain name format. The final level of resolutions is dynamic peer discovery. Nowadays, there are many peer discovery methods (as seen in Section 4.5.2), but the tracker's announce method is the most important and relevant.

3.3. Network of Information

Network of Information (NetInf) architecture involves an entity called the resolution engine and it implements one of the core functions of the architecture, i.e., name resolution for NetInf object identifiers. In NetInf, routing is tightly connected to name resolution and without routing it cannot be performed. In NetInf, name resolution is also not restricted to a single protocol; several protocols are tailored to specific environments with a scope ranging from local to global.[9]

As mentioned before, the main objective of the name resolution mechanisms in NetInf is to resolve object identifiers (IDs) to, for example, network locators. Name resolution resolves object IDs to global locations as well as local locations whenever they are available locally. Resolution also covers environments, when there is no global connectivity. A basic chain of resolution is the Information Object (IO; what) – Data Object (DO; how) – locator/Binary Object (BO; where). The IO can be mapped to another IO, multiple IOs or multiple DOs, whereas DOs can be mapped to one or more BOs. DOs can also be bound to several chunks. NetInf name resolution mechanisms should cover features such as location-based resolution, mobility and multihoming support, etc. with exactly the same basis as in the upcoming prototype. In fact, this is essentially the resolution model used in our prototype.[9][17]

For local name resolution, solutions based on the Distributed Hash Table (DHT) are quite suitable due to the NetInf flat name space (in contrast to a hierarchical name structure such as that in DNS, MILSA and CNN). With global scope, however, DHT-based name resolution can cause problems with the nature of the Internet – a combination of several administrative domains with low co-operation. Binding placement and control are particularly problematic. NetInf has several different proposals for integrated name resolution and routing, but Multiple DHT [18] and Late Locator Construction [19] proposals are the strongest possibilities.[9]

3.4. MILSA

Mobility and Multihoming Supporting Identifier Locator Split Architecture (MILSA) is one of many proposed architectures for the next generation of the Internet. MILSA tries to combine two existing approaches (temporary solutions for the minimization of routing tables and identifier-locator split) to address all the design goals defined in [20] by the Internet Research Task Force (IRTF) Routing Research Group (RRG). The identifier locator split (basically a split of the network layer into routing and identifier sublayers), in particular, provides a possibility for better support for mobility and multihoming.[21][22]

The architecture of MILSA represents three types of hierarchies that divide entities into zone, realm and Realm-Zone Bridging Server (RZBS) hierarchies. The physical topology of the MILSA network is divided into zones, whereas network addresses and locators are also allocated hierarchically. Realms are logical groups of objects that belong to some specific group of entities and that trust each other. The bridging between realms and zones is performed by the RZBS hierarchy. For simplicity, the group of hierarchically organized bridging servers takes care of the mapping between the identifier and the locator. In MILSA, objects can belong to several realms and can therefore have multiple identifiers. The client can also support multihoming by belonging to several zones and can therefore have several locators.[21]

One of the key features of object resolution in MILSA is hierarchical object naming, known as the Hierarchical URI like Identifier (HUI). For example, if Teemu belongs to Subrealm-x in Realm-Y, the HUI will look like this: *Teemu.Subrealm-x.Realm-Y*. Realm-Y and Subrealm-x represent hierarchical locations in HUI, whereas Teemu is a flat name belonging to subrealm-x.[21]

MILSA uses three levels of mapping in object resolution. As HUIs are not necessarily a user-friendly approach to names, MILSA uses DNS for the upper names. For this reason, the first level of object resolution is to map the DNS name to the HUI, and this is done by the DNS infrastructure. The next level (RZBS infrastructure) converts the HUI to the locator. After the location of the object is clear, the Access Zone Router, Realm and Policy server deliver the data to the destination.[22]

The elements of MILSA have clear similarities with CCN. MILSA still follows the host-based networking paradigm, however, instead of CCN’s content-centric approach.

4. PEER-TO-PEER OVERLAY NETWORKS

A peer-to-peer (P2P) network is a popular distributed system at the application layer of the TCP/IP model. In Germany alone, 53% of all traffic on the Internet was generated by peer-to-peer file-sharing systems in 2008 [1]. Of the current Internet technologies, peer-to-peer networks are the closest to the NetInf concept and are therefore also used in the implemented prototype.

In peer-to-peer, the traditional pair client-server (on the left in Figure 3) is replaced with a client-client pair (on the right in Figure 3). This generally means that every node in the network can be represented as a client and a server, and in this context the term client is replaced with peer. The use of a peer-to-peer system reduces the possibility of single point failure in the network. If, for example, one single server is allocated to file distribution and the server breaks down, the file is not available anywhere (single point fails, see Figure 3, whereas file availability is indicated by the database symbols). If the file is distributed over a peer-to-peer network, however, and the initial distributor breaks down (one of the peers), the file is still available in the network from other peers (see Figure 3). In peer-to-peer networks, the requester peer can download the content from any peer or multiple peers simultaneously. In addition, by using peer-to-peer instead of a client-server pair, we can reduce the number of big central servers in the network and distribute the overall load of the network over a wider area. The actual cost of peer-to-peer-like systems is therefore that of the local resources that are shared with other users, such as storage space, processing power and network bandwidth. In general, to receive the benefits of using peer-to-peer, it is necessary to be prepared to also share one's own resources.[23][24][25]

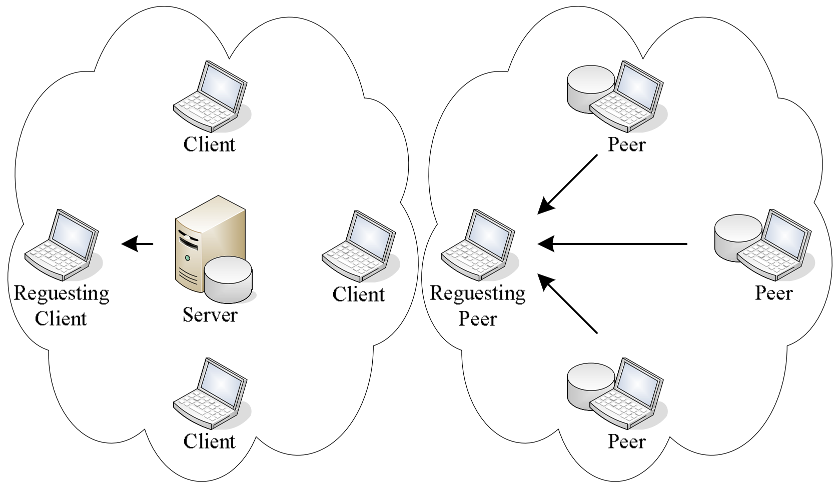


Figure 3. Comparison of client-server-based (left) and peer-to-peer-based (right) networks.

Typical ways of dividing peer-to-peer networks are the structure of the overlay network (network built on top of another network, e.g., peer-to-peer network run on top of the Internet) and the stage of centralization. In general, the peer-to-peer system can be structured or unstructured and centralized or decentralized. The structured network has rules for the organization of peers and where the data is stored etc. This makes, for example, searching more efficient than in unstructured networks.

Unstructured networks are organized randomly and data can be located anywhere in the network. The stage of centralization informs of the existence of centralized elements in the network. The network can have single elements allocated to a specific purpose (centralized elements such as servers) or functions of the network can be totally distributed over peers (decentralized network).[23]

In this thesis we concentrate on unstructured peer-to-peer file sharing networks and the protocols behind them. These are presented from the least common to the most popular. We start with the least common, though it is still often studied in literature, the peer-to-peer network, namely Freenet in Section 4.1. Next, we concentrate on FastTrack in 4.2, Gnutella in 4.3 and eDonkey in 4.4. Most of the attention is given to BitTorrent due to its use in the implemented prototype, and it is studied in Section 4.5. Finally, we introduce some of the biggest issues of peer-to-peer networks in Section 4.6.

4.1. Freenet: An Adaptive Peer-to-Peer Network

The primary purpose of Freenet was to build a global information storage system that is secure and has plenty of redundancy. Due to the redundancy, Freenet is fairly fault tolerant. Nodes in Freenet route messages around the network in a similar way to the Internet Protocol (not using flooding as in Gnutella) and are connected by TCP connections. Nodes in the network also store and deliver various files that are not their own. To join the Freenet system, a node needs to know the first node to which it should connect. The user can control three different variables in the network: local storage disk size, inserting a new file into the network and requesting the particular file from the network. Nonetheless, the overall performance of the Freenet network is not good.[23][26]

The Freenet network is totally secure (by using one-way secure hashes) and a node does not necessarily have access to data that are stored in its storage. In Freenet, the user has to know that actual data exist in the network as they cannot be searched from the network. Every query received by the node is delivered to its closest neighbour (if it does not have the file itself) and in that way Freenet does not waste limited network resources. A node only knows about its neighbours, and the actual file transfer is performed by replicating the file to nodes in the path, one by one.[26]

4.2. FastTrack: A Decentralized File-Sharing System

FastTrack is a decentralized peer-to-peer file sharing system that allows searching by metadata information. Peers in FastTrack are divided into two classes: ordinary and super peers. Super peers usually have a higher bandwidth, more disk space and processing power than ordinary peers and are involved in search functionality and query delivery. The system does not perform any hash-based file verification, however, and the number of corrupted and fake files in the system is therefore huge. Overall, super peers usually take care of the whole searching process in a FastTrack network (in all cases when at least one super peer exists in the network).[23][27]

After a user launches a FastTrack client (for example, KaZaA or iMesh), the ordinary peer chooses one super peer to be the parent peer. In the next phase, the ordinary peer establishes TCP connection to its parent peer and uploads metadata for the files to be shared by the ordinary peer. Super peers keep a file index of available files from its children peers and maps indexes to IP addresses. The location of files in FastTrack operates as follows: if the user wants to download a file, he or she has to search it with the client. The ordinary peer delivers a query including key words to the super peer and the super peer returns the IP address and some other necessary data to the ordinary peer if a query matches some file in the database. If the super peer does not find a match in its database, however, it can query the content of another super node. After the ordinary node receives the IP address of the file provider peer, the downloading can start directly between ordinary peers.[27]

4.3. Gnutella: A Decentralized Peer-to-Peer Protocol

Gnutella is a widely used (1.8% of all peer-to-peer file-sharing traffic in Germany in 2008) decentralized peer-to-peer protocol in which search functionality can be totally distributed over the network. Peers in Gnutella are also called servents and are either servers or clients. At first, a peer has to join to the Gnutella network by following some rules and become a servent. A new servent typically has to connect to the available known host. To locate wanted data from the network, a servent has to query them from its neighbours within a certain radius, most typically by flooding. The flooding generates bursts of traffic to the network, and because of this the search method is not scalable. The actual data downloading is done between two peers, where one is a client and the other a server.[1][23]

In Gnutella, the protocol defines three different message types and a response to these, one pair for all functionalities. To join the network, a servent uses a broadcasted PING message. Neighbours reply to the PING with a PONG message, which carry out information about the neighbour servent such as the connection information (IP address) and information of the data items it has. The search functionality is implemented to use QUERY and QUERY RESPONSE messages. The QUERY message consists of a search string specified by user that the corresponding servent compares with local files. The QUERY RESPONSE contains the necessary information about the downloadable file. The downloading is done by GET and PUSH messages directly between two participating servents. Afterwards, for enhancing the performance of routing of the Gnutella network, developers introduced a notion of ultra-peers (cf. FastTrack), but achieved the benefits are questionable.[23]

4.4. eDonkey: A Hybrid Two-Layer Peer-to-Peer Network

The peer-to-peer protocol named eDonkey and all of its variants (for example, eDonkey 2000) are hybrid two-layer, peer-to-peer networks that generated 24% of all peer-to-peer file-sharing traffic in Germany in 2008. The network is composed of two elements: eDonkey clients (stand for the same as a peer in the eDonkey protocol) and eDonkey servers. The clients are used to direct file-sharing in the network whereas the servers act as file locators and distributor of other server locations. Downloading is performed directly between the clients however. [1][23][28]

To start the eDonkey file sharing service, a client has to register on to eDonkey server over TCP. The client notifies all the files, it wants to upload to other clients, to the server and the server maintains a list of available files from its peers. When the user wants to download a file from the network, it first has to perform a search and send a query to the server. The server responds with a message consisting of the information about matched files and their network locations (client addresses). After the user makes decision about the file download, the client asks for a load slot from the clients that are providing the file. Since uploading is possible by provider, the file provider connects to the requesting client using TCP, reports the index of the chunk and starts the actual upload. The chunks in eDonkey are typically 10 MB in size. By dividing the file into smaller chunks, it is possible to download the content from several sources simultaneously and provide the part of the file to others before the downloader has a completely copy of the file.[28]

4.5. BitTorrent: A Centralized Peer-to-Peer System

BitTorrent [29] is one of the most popular peer-to-peer content distribution networks with a 71% share of the traffic, and it produces the biggest share (37%) of today’s Internet traffic when measuring single protocols (details from Germany in 2008) as per [1].

In BitTorrent, the actual content is replicated to a large set of peers so that, for example, download performance and scalability are enhanced compared with other peer-to-peer systems [30]. It also allows the content to be downloaded from several peers simultaneously. BitTorrent is generally used for the distribution of large files and, in particular, for the delivery of copyrighted material, acting as one of the biggest issues with it.

Users and research communities have established special terminology related to the BitTorrent content distribution network. These terms are generally utilized in publications, spoken language and, in particular, in this thesis. It is therefore reasonable to start with following list:

- Client: an application that implements the BitTorrent protocol at the application layer (TCP/IP model).
- .torrent: a metadata file of the distributed file made by the client. It includes necessary information about the content distribution session (described in detail in Section 4.5.1).
- Peer: a node in the BitTorrent system. It can behave as a leech or a seed.
- Leech: a peer that does not have a complete copy of the distributed file. The leech mainly downloads the file, but once it has received parts of it (pieces), it can also upload.
- Seed: a peer that has a complete copy of the distributed file and only uploads it.
- Swarm: a group of peers participating in the same content distribution session, specified by .torrent
- Tracker: a server that maintains a list of peers in the swarm. It does not have information on which one has what, only on peers and their stages of download.
- Piece: a fixed-size chunk of the distributed file. Each file comprises one or more pieces.
- Block: a network transmission unit of data. Each piece typically comprises several blocks.

The simplified BitTorrent overlay network is illustrated in Figure 4. After the need for content distribution is observed, the first thing that has to be done is to create the .torrent (1.). The .torrent cannot be created without having the complete file however. The .torrent is created by the client software at the initial seed (the initiator of the content distribution). After the creation of the .torrent, it is necessary to make the .torrent available to other users. This is typically done by uploading the .torrent to the public (or private) web server (2.). Some of the tracker implementations also require it before servicing any peers. Finally, the initial seed opens the .torrent with a client (3.), contacts the tracker to join the swarm (4.) and begins the upload of the content, known as seeding. After this procedure, the content is made available to the leeches.[29][31][32]

To download the content from other peer(s), a leech has to join a special torrent session that is separate from all downloads. The sessions are specified by the .torrent file, which contains metadata about the downloadable file and the tracker's Uniform Resource Location (URL). The first thing to do before downloading is therefore to search and download (5.) the metadata file from, for example, the web server. Once the leech has the .torrent, it opens the file with the client (6.) and asks for peers from the tracker (7.) with the tracker announce message (described in 4.5.2). The tracker maintains a list of peers in the transfer session being the only centralized entity in the BitTorrent network. The tracker does not participate in the actual content distribution, however, it only returns the peer list (in this case, the location of the initial seed) of this particular content distribution session. The leech contacts the initial seed and begins the download over BitTorrent's peer-wire protocol (8. – described in Section 4.5.3). If other peers also want to download the content, these only have to repeat the procedure from (5.) to (8.).[29][31][32]

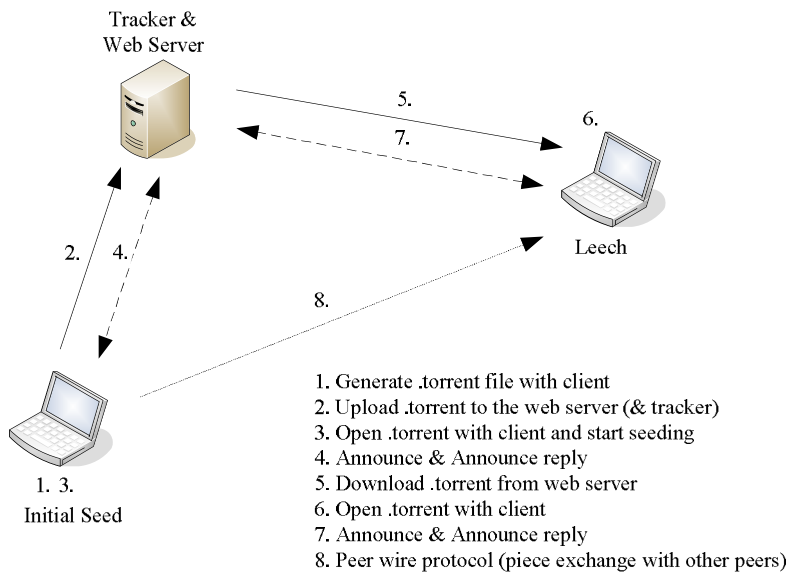

Tracker &
Web Server
5.
6.
7.
Leech
2.
4.
8.
1. 3.
Initial Seed
1. Generate .torrent file with client
2. Upload .torrent to the web server (& tracker)
3. Open .torrent with client and start seeding
4. Announce & Announce reply
5. Download .torrent from web server
6. Open .torrent with client
7. Announce & Announce reply
8. Peer wire protocol (piece exchange with other peers)

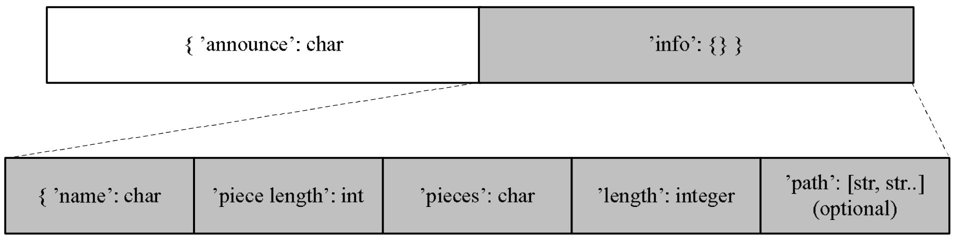
Figure 4. Simplified structure of the BitTorrent overlay content distribution network.

In BitTorrent, the downloading of the file from several peers simultaneously is made possible by splitting the file into pieces. The pieces are usually 256 kB, 512 kB or 1024 kB in size. Pieces are also divided into smaller network side-transmission units known as blocks. The size of a block is 16 kB. In BitTorrent, leeches can also upload the content during the download. The leech cannot offer a piece to others before it has a complete copy of it however. To keep the torrent session alive, there should be at least one seed in the swarm (a peer that has all the pieces). The swarm is the group of peers (down- and uploaders in different places) participating in the torrent session. In the torrent session, peers communicate via others with a set of exchange messages to execute the choke algorithm, which is described in more detailed in 4.5.4, to achieve fair downloading conditions for every leech. Another important algorithm in BitTorrent is the rarest first policy, which tries to maximize entropy of pieces in the swarm. All the phases of BitTorrent content distribution from peer discovery to seed phase are described in more detailed in Sections 4.5.2-4.5.5 of this thesis.[31][32]

4.5.1. BitTorrent Metadata: .torrent

The first action in BitTorrent based on content distribution is the creation of a metadata file known as .torrent. The creation of the .torrent is usually performed by the client, which is the application that implements the BitTorrent protocol in the user's machine, for example, the computer, external hard disk, Internet tablet, mobile phone, etc. The metadata file aka .torrent is the key to a particular BitTorrent content distribution session, and the most typical way to deliver that to other users is to upload it to some of the many dedicated torrent (web) servers. Other delivery possibilities such as e-mail and text-based Internet conversation applications like instant messaging or the exchange of USB sticks are of course also allowed.[29][33]

The structure of .torrent is illustrated in Figure 5. The complete metadata file is a bencoded (simple and flexible encoding method, especially for cross-platform applications, described in, for example, the official BitTorrent protocol specification [29]) dictionary consisting of two separate keys referring to its own fields. These two keys are *announce* and *info*. The *announce* field contains the announce URL of the tracker dedicated to maintaining peers for this .torrent. In practice, the *announce* field and URL of the tracker are used for finding the endpoint of the announce message (source of a peer list). The second field, namely *info*, involves a dictionary on the necessary information of the content delivery consisting of four necessary fields: *name*, *piece length*, *pieces* and *length*.



- ***'announce'***: *URL of tracker allocated to this particular .torrent*
- ***'info'***: *a dictionary representing necessary information about content delivery:*
  + *'name': name of the content/path name in multifile torrents*
  + *'piece length': utilized piece size*
  + *'pieces': SHA1 has string of every piece*
  + *'length': total length of the content*
  + *'path' (optional): a list with name of files in multifile torrents*

Figure 5. Structure of BitTorrent metadata file.

The *name* field includes the name of the content and provides descriptive information about the file to users. The *piece length* field indicates the used piece length in this torrent session. With this information a newcomer leech is able to, for example, gather blocks into complete pieces. Once a piece is received in full, the leech performs a checksum calculation to make sure the piece is received correctly. The *pieces* field therefore includes the SHA1 [34] value of every piece and the size of one value is 20 bytes. The final required field is *length* and it contains information about the length of the distributed file. This final field is also used for descriptive information to users.[29]

In the case of multi-file torrents, in practice, when a folder is distributed instead of a file, there are a couple of changes to the structure of the .torrent. First, the *name* field in the *info* dictionary involves the name of the folder instead of the name of the single file. The fifth and optional field, namely *path,* is also introduced and includes a list of file names in the folder specified in the field name.[29]

Figure 6 illustrates an example of the structure of a BitTorrent metadata file used in latter experiments described in Chapter 7. First, the field, namely *announce,* consists of a complete string of the tracker URL based on the IPv6 addresses. The name of the file in our .torrent is 100M and the piece length used is 256 kB. In our case, the field *pieces* include 8000 characters of hash values for every piece. Finally, the length field corresponds to the file size used, which is 100 MB in our case. All the described fields are represented in a BitTorrent metadata file in bencoded form, but, in practice, bencoding only indicates the start and end points of value. The fields are therefore in human-readable form.

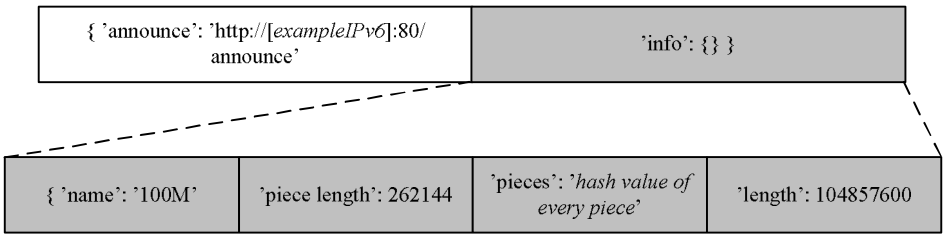

{ ’announce’: ’http://[*exampleIPv6*]:80/ announce’
’info’: {} }
{ ’name’: ’100M’
’piece length’: 262144
’pieces’: ’*hash value of every piece*’
’length’: 104857600

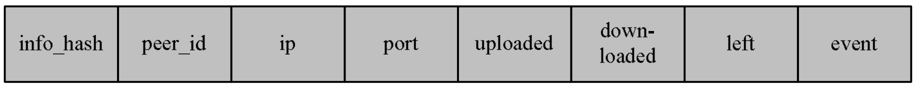
Figure 6. Example structure of the BitTorrent metadata file.

The next phases of BitTorrent-based content delivery are to open the particular .torrent with the client and then to register the tracker and start seeding the file. It is also important to deliver the .torrent to other users. The tracker registration phase is described in more detailed next.

4.5.2. Peer Discovery

A standard way to discover other peers in a BitTorrent session is to ask for a list of peers in the swarm from the centralized tracker using a simple protocol on top of HTTP [33], known as the tracker HTTP protocol. In the case of one central tracker per session, the probability of single-point failure increases and the availability problem can therefore take effect [35]. For example, if the content is distributed over a standard BitTorrent network and the tracker crashes for some reason, then the information about the peers is not available anymore. More information about availability in BitTorrent can be found from [35]. To avoid availability issues, a multitracker [36] and totally decentralized DHT [37] extensions have been developed for the protocol. These solutions are described briefly later in this section after we have considered the original version of peer discovery further.

A peer list from the tracker to the requesting peer is delivered using the tracker HTTP protocol. The protocol specifies two separate message structures: an announce message via an HTTP GET request and an announce reply message as a plain/text document in response to it. The structure of the announce message is illustrated in Figure 7.

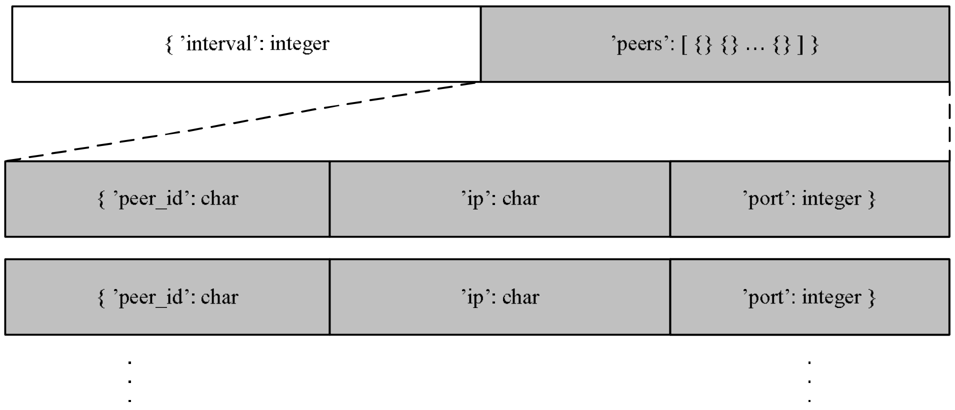


- ***'info_hash'**: SHA1 hash value of .torrent's bencoded info field*
- ***'peer_id'**: random peer identification string*
- ***'ip'**: IP address of requesting peer*
- ***'port'**: TCP port requesting peer is listening to*
- ***'uploaded'**: number of uploaded bytes by requesting peer*
- ***'downloaded'**: number of downloaded bytes by requesting peer*
- ***'left'**: number of bytes still needed to download*
- ***'event'**: current status of requesting peer*

Figure 7. Structure of announce message in the tracker HTTP protocol.

The first key in the request message is *info_hash*. This maps to the field carrying the SHA1 hash value from the bencoded .torrent's *info* field (see Figure 5). The field is used to identify the torrent session in the tracker. The next field, namely *peer_id*, is a randomly generated string in the client to avoid peer identification problems in, for example, the tracker caused by Network Address Translations (NAT). If, for example, the campus network is restricted from the Internet by NAT, all incoming announces from the network to the tracker located somewhere in the Internet seem to come from the same IP address. If several clients in the campus network are accidentally waiting for data at the same TCP port, the tracker also assumes that these separate clients are the same, and the *peer_id* is therefore used. The fields *ip* and *port* map to the IP address and the TCP port to which the peer is listening for incoming connections. These two fields as well as the *peer_id* make up the information delivered to other peers in the announce reply. The next three fields: *uploaded*, *downloaded* and *left* are only used for statistical purposes, indicating, for example, the current number of content copies in the swarm at the tracker. The last field in the announce message is *event*, which indicates the peer's current status in the swarm. In practice, the status could be started, completed or stopped. The peer always sends a new announce whenever it changes. When the peer performs its first announce, the *event* is started. After the peer has downloaded the content completely and has turned to seed, the *event* is completed, and whenever the peer leaves the swarm, the *event* is stopped.[29]

Figure 8 represents the tracker's response to the announce message and is a bencoded dictionary in a plain text document consisting of two keys: *interval* and *peers*. At first, the key interval maps to the field including the time that the peer should wait before re-announcing the tracker in seconds. As mentioned in the previous paragraph, the tracker is re-announced in the change of *event* and when the interval time of the latest announce message expires. The key named *peers* maps to the list of dictionaries, which consists of active peers in the session. The peers field consists of dictionaries with three different keys: *peer_id*, *ip* and *port*, with one dictionary corresponding to one peer. These fields in the dictionary contain the same information as reported to the tracker in the announce message (Figure 7). In the case of an error while announcing the tracker, the announce reply message sent by the tracker involves only one key, namely, *failure reason*. *Failure reason* maps to a human-readable string that indicates the reason for the failure.[29]



- ***'interval'***: *re-announce interval, which peer should wait before a new announce*
- ***'peers'***: *a list of of dictionaries, where one dictionary responds to one piece of peer connection information:*
    + *'peer_id': a random peer identification string (cf. announce)*
    + *'ip': IP address of a peer*
    + *'port': TCP port of a peer*

Figure 8. Structure of tracker's response message in the tracker HTTP protocol.

A peer requests the tracker for peers with an announce message before the actual download can start. According to the announce, the peer is registered in the peer list of the torrent in the tracker. In the announce reply message, the tracker returns a list containing 50 peers by default from the particular torrent picked at random (if there are many peers in the swarm). Each peer reports its download state periodically in the interval defined by the tracker in the response message (Figure 8). The peer state is also updated every time an event changes, for example, before disconnecting from the tracker. If the number of peers that the local peer knows drops below 20, then the peer asks for more peers by re-announcing the tracker. A peer tries to keep up 20-40 connections to other peers, in a manner that the interconnection between different lists of peers is secured. The maximum number of peers maintained by a local peer, however, is 80 by default.[31][38]

To ensure the availability of a list of peers, the multi-tracker extension is introduced for the BitTorrent protocol [36]. The extension allows two or more trackers to provide tracking feature of the same content. The extension basically defines a new field for the metadata file, called the announce-list, in addition to the announce and info fields. The announce-list contains the same information as the announce field but can have one or more trackers. The multi-tracker extension is created for two purposes: load balancing and back up. In the case of heavily populated swarms, the number of announces increases significantly, and if all the announces are directed at the same tracker, the effect can be serious overloading of the server. Load balancing is therefore reasonable for some of the swarms. In the load balancing, the announce-list field consists of several tracker URLs and the peer selects one at random and sends an announce message to that one. Trackers in the same list also communicate with others and exchange their information about peers, and in that sense all trackers are aware of the complete swarm. In the case of several lists of trackers, trackers are used for backup purposes. The peer basically tries to announce the actual tracker first, and after that the backup tracker and so on. In the re-announce, the peer repeats the same procedure.[35]

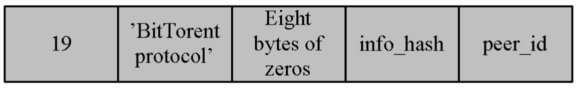
A completely trackerless alternative to BitTorrent is a DHT extension. In DHT, trackers are replaced by nodes implemented for every client and, moreover, DHT makes every client act like a small tracker. The node and the peer are separate components in the client, and the basic task of the node is to offer useful peers to the peer. The DHT in BitTorrent is based on peer-to-peer storage and a lookup system named Kademlia [39]. Messages from the BitTorrent's DHT are sent over UDP with its own set of exchange messages. In the creation phase of the torrent, the client usually asks users to add some DHT nodes with which the internal node can start bootstrapping. In practice, the user adds known node(s), the internal node starts looking for the file from this node first and, if necessary, moves toward the nodes known by the user-added node etc. In the end, the file is discovered somewhere.[35][37]

One alternative option for discovering peers is peer exchange (PEX). In PEX, peers interact directly with others and share their information about peers [40]. In practice, PEX is not used alone but with, for example, tracker announces, PEX can provide peer information that the local peer did not have before and can therefore improve download performance.

4.5.3. Content Retrieval

Once a peer has received information about other peers in the swarm, the proper piece downloading process can begin. The necessary message exchange between peers for downloading the file consists of actual pieces as well as a few controlling messages (shown in Figure 9, Figure 10 and Table 1). The information delivery is also called a peer wire protocol and is implemented over TCP. The controlling messages in the peer wire protocol allow peers to exchange information about piece distribution and to implement an algorithm named choke. The choke algorithm is a way of reporting to a remote peer that we do or do not want to upload pieces. In practice, whenever the local peer chokes another peer, the local peer reduces the upload blocks of the file to it. We present the choking algorithm in more detailed in Section 4.5.4. In the next paragraph, we describe how piece distribution information is exchanged and how the protocol aims to shape it.[29][32]

The first message sent in the peer wire protocol is a handshake message (in Figure 9). The connection initiator peer sends it to the remote peer to which it wants to connect. The remote peer also replies with a handshake message if the *info_hash* value of the torrent is the same as that which the remote peer is distributing, otherwise the remote peer drops the connection. The only change in the handshake reply message, compared with the handshake, is the *peer_id*, which is replaced by the remote peer’s own *peer_id*. If the replied *peer_id* is not expected (same as the information received in, for example, the tracker response), the initiator also drops out of the connection. After a complete handshake is performed, there is no need for another until the connection is lost. Connections between peers are symmetric and data flows can go in both directions. The complete handshake message structure is shown in Figure 9.

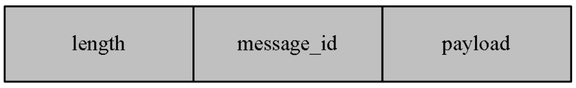


- ***19***: *a length prefix of the message*
- ***’BitTorrent protocol’***: *protocol identification string*
- ***’Eight bytes of zeros’***: *eight reserved bytes set to zeros*
- ***’info_hash’***: *SHA1 hash value of .torrent’s bencoded info field (cf. announce)*
- ***’peer_id’***: *random peer identification string*

Figure 9. Structure of the handshake message in the peer wire protocol.

The first field of the handshake message is the length of the message. In the case of the handshake, it is number 19 (because the length of the handshake is fixed). The second field indicates the protocol used with a human-readable string, as it is ‘BitTorrent’, whereas the fourth one contains eight reserved bytes that are set to zero. The fields, namely *info_hash* and *peer_id*, are the same as in the announce message (Figure 7) including session and peer identification details. The three last fields of the handshake message are encoded as four bytes of big-endian.[29][32]

The BitTorrent protocol defines a way of reporting pieces that peers have to remote peers at start-up and is known as a bitfield message. The bitfield message is sent after a successful handshake and is performed between the initiator and remote peers, in both directions. The structure of the bitfield message follows the formal message structure of the peer wire protocol (shown in Figure 10). The content of the bitfield message is shown in Table 1. The first field indicates the length of the message and, in the case of the bitfield, it is variable in size, depending on the size of the file and the employed piece size (more pieces – longer length). The next field specifies the identification number of the message, and with the bitfield the number this is five. Finally, the last field in the formal message is *payload*, the information actually delivered, which in this case is the bitfield. The bitfield in the payload field consists of a piece map with one number being dedicated to one piece, and if the peer has that particular piece, the referring number is set to one.[29][32]



- ***’length’***: *length of the message*
- ***’message_id’***: *an identification number of the message*
- ***’payload’***: *payload of the message*

Figure 10. Formal message structure in the peer wire protocol.

After the handshake procedure and the bitfield exchange, the local peer should have established a connection to all the peers in its peer list. It should also know what pieces other peers have. The actual download can now begin from the remote peers that allow the download. As long as the local peer has fewer than four pieces completely downloaded, it follows a random first policy. In practice, this means that the local peer is requesting pieces in random order by block request messages following the structure presented in Figure 10. The fields used are shown in Table 1: *length* is 13 bytes, *message_id* is 6 and *payload* consists of information about the requested blocks (piece_index = identification of piece, begin = place to begin in the piece, length = length of the requested data – 2^{15} = two blocks). This particular policy helps the peer to start downloads due to the choke algorithm. Requested blocks are transferred using piece messages, which again follow the same structure. Fields of the piece message contain information about the length of 9 + 2^{14} bytes, the message identification number seven and the actual block with the necessary identification details such as the piece_index and the place in the piece.

Table 1. Messages needed for the peer wire protocol

Message name:	length	message_id	payload
bitfield	variable	5	bitfield
cancel	13	8	piece_index, begin, length
have	5	4	piece_index
keep-alive	0	-	-
piece	9+2^14 (usually)	7	piece_index, begin, block
request	13	6	piece_index, begin, length

To avoid long delays between received blocks, the peer must have several requests pending all the time because of TCP's properties. This buffer of requests is also known as pipelining, and the number of pending requests is usually five. When the peer has downloaded one piece fully, it performs a hash check based on the piece hash value in .torrent. If the hash check performs correctly, the peer reports the holding of the piece to the remote peers in the list by sending a have message (shown in Figure 10 and Table 1. The length is five bytes, the message identification number is four and the payload consists of the piece_index). In this way, all the peers can update their peer-piece tables.[29][31][32][33]

The other important policy in the piece download strategy is a strict policy. The policy defines that since single block from a piece is requested, the blocks in other pieces are not requested before the earlier one is completed. This policy avoids the appearance of many partially received pieces.[31]

After downloading the first four pieces, the rarest first algorithm comes forward. The purpose of the rarest first is to replicate the rarest pieces as quickly as possible in the local peer's peer list. In practice, the local peer follows the peer-pieces table and defines the set of rarest pieces. The next piece to be downloaded is picked up randomly from the rarest piece set and, in this way, the entropy of the pieces becomes lower: there are fewer pieces with only a few copies.[31][33]

The last policy in the piece downloading phase is called the end game mode. The purpose of the mode is to ensure that the last few pieces are downloaded as fast as possible, even if the piece is requested from a peer with a slow transfer rate. In the end game mode, the local peer delivers all the block requests to all the remote peers after these blocks are requested once from the slower peer. After the local peer has received the block, it sends cancel messages (see Figure 10 and Table 1; length of 13 bytes, the message identification number is eight and the payload consists of information on the requested blocks) to the other peers to avoid wasting bandwidth.[33]

The BitTorrent's peer wire protocol also defines a keep-alive message (see Figure 10 and Table 1; length of 0 bytes, with no message identification number or payload – just an empty message) that is used to keep the remote peers alive that refused to upload to a local peer or to which a local peer refused to upload. The keep-alive messages are sent to remote peers once in two minutes.[29]

4.5.4. Peer Selection

The BitTorrent system is designed to act in a tit-for-tat-like manner. Tit-for-tat is a theoretical game strategy and, in principle, it works as follows: a local node initially cooperates, after it receives a response action from another node the local node repeats the action made previously by another node [41]. In the BitTorrent protocol, a tit-for-tat-like manner means that the amount uploaded by a remote peer to the local peer should be equal to the amount uploaded by a local peer to a remote peer. Tit-for-tat is implemented by a choke algorithm. This way, BitTorrent can provide a good level of download and upload reciprocation and BitTorrent can also minimize the phenomenon known as free-riding, which is downloading without uploading.[31][42]

Implementation of the choke algorithm requires four different messages to the peer wire protocol. These messages follow the same structure as the formal message in the peer wire protocol (in Figure 10) and these are used to indicate changes in the peer's game behaviour. Messages used in the choke algorithm and fields of these are represented in Table 2.

Table 2. Messages needed in the choke algorithm

Message name:	length	message_id	payload
choke	1	0	-
unchoke	1	1	-
interested	1	2	-
not interested	1	3	-

In practice, a remote peer can be choked (the remote peer is not allowed to download from the local peer) or unchoked (the remote peer is allowed to download from the local peer) by the local peer. The local peer can also be interested in the remote peer (the remote peer has pieces that the local peer does not) or not interested (the remote peer does not have pieces that the local peer does not). The default set up for all new connections between peers are: not interested and choked. After the peer notifies that the remote peer has pieces that the local one does not, it shows interest with an interested message. Downloading from the remote peer does not start, however, until the remote peer unchokes the local one with an unchoke message.[29]

The actual choke algorithm in the leech phase works as follows. No more than four interested remote peers can be unchoked by the local peer at the same time. The local peer unchokes the three fastest block source peers every 10 seconds. If a remote peer that is not interested about the local one and which has a better upload rate to the local peer becomes unchoked. Also if the local peer also becomes interested by the remote peer, then the worst uploader from the group of four becomes choked by the choke message and is replaced with the remote peer. Every 30 seconds, however, the local peer unchokes one interested remote peer at random, this is also called optimistic unchoke. The optimistic unchoke serves two purposes: it provides the possibility of raising the up download speed of new peers in the swarm and allows new ones to obtain the first piece.[29][31]

4.5.5. Seeding

The first version of the seeding phase choking algorithm was based on the same method as described in Section 4.5.4 with the exception that the measured download rates were replaced by upload rates and the choking decisions were made based on that information. This version of the algorithm strongly favoured peers with a high download bandwidth, which results from them being unchoked all of the time.[31][33]

The new choking algorithm contains several changes from the earlier one. The operation of algorithm is divided into three 10 second periods. Every 10 seconds, the local peer calculates the time to when an interested and unchoked remote peer was last unchoked. Interested and unchoked peers are ordered according to the previous rule in the order in which the most recently unchoked is first in the list. During the first two, ten-second periods, the first three peers in the ordered list are unchoked and the fourth one is selected at random from the interested and choked ones. In the third period, all four remote peers are stayed unchoked.[31]

4.6. Issues in Peer-to-Peer

Issues in Peer-to-Peer systems are divided into several sections. One of the biggest problems in P2P is certainly legality issues. The portion of illegal content in peer-to-peer networks has grown significantly in recent years. This thesis does not concentrate on this however. Nowadays, the P2P generates an increasing amount of traffic on the Internet, as seen in [1]. It can also be derived from this that the traffic that crosses the borders of the Internet Service Providers (ISP) has increased greatly, inflicting additional costs on the ISPs [7]. These issues are introduced in more detail in Sections 4.6.1-4.6.4. Major changes in the popularity of peer-to-peer file-sharing sessions cause availability problems [43], for example, with very wide popularity, the torrent session can cause overloading of the tracker and in the case of small swarms, peers can be overloaded or even hard to discover. The issue that is relevant to this thesis is swarm size optimization, which is discussed in Section 4.6.5. The movement of the user while downloading the file can now also cause demanding problems. This is discussed in more detail in Chapter 5.

4.6.1. Application-Layer Traffic Optimization

The Internet Engineering Task Force’s (IETF) Application-Layer Traffic Optimization (ALTO) working group has worked on issues in distributed applications. On today’s Internet, distributed applications such as peer-to-peer have grown to play a major role in the use of the capacity of, for example, routers all around the Internet. These applications do not have information about the underlying network, and the traffic is therefore generated by systems that can cross network boundaries several times. This cross-network or cross-ISP traffic can generate overloading in the backbone and transit links, and this also affects the user experience in a negative way. The increase in cross-ISP traffic could also be very costly to the network operators. The intended information is very often available in the own network or its neighbours, and the delivery of network topology information to the applications can therefore lead to a significant improvement in these issues. By using the topology information, the applications can perform better than random peer selections and select, for example, peers within its own ISP's network.[7]

The Application-Layer working group divides existing literature related to the ALTO problem into two categories. The first approach to the problem is topology estimation through some stage of layer cooperation. Two solutions based on this approach are introduced in Sections 4.6.2. and 4.6.3. Another approach is application-level topology estimation (using information only from application-layer) and one example realization of this is introduced in Section 4.6.4.[44]

4.6.2. Provider Portal for Applications

The Provider Portal for Applications (P4P) introduces a flexible architecture in which network providers can interact with applications in a positive way, concentrating mainly on peer-to-peer content distribution. A key component of the architecture is an entity called iTracker, which is a provider portal for content distribution. The iTracker can deliver information such as policy (usage policies in the network), P4P distance (distance between peers) and capability (network provider capabilities) to more traditional types of trackers known as appTrackers. The plan is for P4P to cover the trackerless systems as well and, in this case, the particular information is delivered directly to the peers. Based on the delivered information, the appTrackers make decisions about the peer list returned to the requesting peer. In this way P2P traffic can be directed in a more beneficial way.[45]

P4P can improve network efficiency as well as the peer-to-peer application performance. The standard BitTorrent is compared with two alternative realizations of BitTorrent: delay localized and P4P BitTorrent in simulations, real Internet experiments and large-scale tests with a commercial ISP. The results show that locality-based BitTorrent and P4P both perform substantially better than traditional BitTorrent when comparing competition times, and the localized BitTorrent performs better than P4P. The advantage of P4P, however, comes from bottleneck link utilization. The localized BitTorrent produces less bottleneck traffic than the traditional BitTorrent, but the P4P was even better.[45]

4.6.3. Oracle Service

The oracle service is another solution for topology estimation with cross-layer information and is provided by Internet Service Providers. The oracle service is realized by ranking potential neighbours by using certain metrics such as does it belong to the same Autonomous System (AS), the number of AS hops and the distance to the edge of the AS. These metrics are used for coarse tuning. For peers inside the same AS, the oracle can serve geographical, performance and link congestion information. By using the preceding information, the peer can choose peers that are close to it instead of selecting them randomly. The BitTorrent oracle service, in particular, could be used after the tracker response to define the locality of peers, or the tracker can call the oracle and sort the peer list directly. The advantages of the oracle service are proved by simulations and Testlab experiments. The evaluation is performed with Gnutella, and the main outcome is that with the oracles, the mean AS distance and the number of intra-peerings can be improved significantly.[46]

4.6.4. Peer Biasing

Another approach to application-layer topology estimation is estimation based only on application layer information. Peer biasing is a technique in which similarities in the Content Distribution Network's (CDN) dynamic DNS redirection behaviours are recycled to estimate the locality of peers. Peer biasing is implemented as an extension to the BitTorrent client called Azureus, and it is also distributed to a large portion of BitTorrent users (120,000 in 2008). The installed extensions provide information such as DNS redirection, transfer rates, path latencies, etc. to the working group. If the peers have the same kind of behaviour, the extension (namely Ono) tries to direct (bias) the connection towards these peers and hence to reduce cross-ISP traffic. The implemented client extension was able to recommend more than 33% of the time when the peer was within the same AS. The duration of the download also improved by 31%. In the case of a large amount of available bandwidth, the results were even more significant.[30]

4.6.5. Dynamic Swarm Management

Another relevant problem in peer-to-peer systems is the effect of the swarm size on system performance. Popular peer-to-peer content has been shown to follow a power law with a very long tail. In practice, a small part of the torrents has very huge swarms and most of the torrents have only a few participants in their swarms. BitTorrent already has mechanisms to divide the torrent into several trackers (and in this case also into several swarms) to prevent overloading. BitTorrent does not have mechanism to gather several swarms back together, however, after the popularity has decreased, and the drop in downloading performance is huge due the unacceptably small swarms.[43]

Distributed Swarm Management (DISM) provides a solution to the twofold problem described in the previous paragraph. This algorithm strives to keep the swarm size at the optimum level. If a swarm becomes too big, it can be split into smaller swarms, and conversely when the swarms become too small they can be combined. In practise, DISM could be realized using a simple extension to the traditional BitTorrent protocol. The extension consists of one new message to the protocol, called the tracker redirect. This message could be used to indicate to the peer that, it should use another tracker specified in .torrent. Measurements show that the improvement in throughput achieved with the use of DISM could be significant, especially, in the context of small swarms.[43]

5. MOBILITY MANAGEMENT FOR IP NETWORKS

This thesis focuses on the operation of a specific application (P2P) in a mobility scenario in which the user usually does not exist within the coverage of the same network all the time. The user should be able to connect to the Internet wherever and whenever she/he wants however. This is also the case in future Internet architectures. We therefore give a quick overview of the mobility scenario and existing mobility solutions in this chapter. All of them begin by introducing the multiaccess mobility scenario in Section 5.1. The standard operations of Mobile IP version 6 and the Media Independent Handover Service are introduced in Sections 5.2 and 5.3. At the end, the Mobility Trigger Management is introduced in Section 5.4 as a more general Media Independent Handover like a service. Mobility Trigger Management acts as one of the major components of the implemented prototype in which the Mobile IP version 6 and Media Independent Handover Service are only used in measurements and for planning discussions.

5.1. Multiaccess Environment

At the early stages of the Internet, the terminals where thought to be static in the sense of the geographic and network location (in practice, the IP address) [47]. As the development of terminals has made it possible to carry machines everywhere, people’s expectations of network connectivity have become extremely demanding. The Internet has to be available regardless of location, time of day and stage of mobility. In addition, it all has to work properly with a high bandwidth and without any configuration by the user. These requirements have brought out many technical challenges that researchers and developers are trying to solve.

The Always Best Connected (ABC) scenario defines the case in which the user does not only have to be connected to the network all the time but also as well as possible. The indicator of network goodness can be, for example, price, data speed or security. This scenario makes it possible to move around different access networks such as Wi-Fi, Worldwide Interoperability for Microwave Access (WiMAX) or the Third Generation of mobile phone technology / Long Term Evolution (3G/LTE) without losing connection at any time. The connection can be performed via one single access (network interface card) or several accesses at the same time. The multiaccess mobility scenario is illustrated in Figure 11.[48]

As is shown in Figure 11, handovers or handoffs (moving from one network access point or base station to another) can be made inside one IP domain or between two IP domains. In the figure, assumptions have been made that one ISP has one IP domain, which is not necessarily the whole truth. A handover within the same IP domain is called micromobility and a handover between IP domains is called macromobility [49][50]. The work of this thesis concentrates on the details of macromobility solutions.

The basic problem of the above scenario is that the movement from one subnet to another causes the IP address to change. The IP address represents the identity and location of the end host and changing it causes a problem of routing the packets to a new address. Against this problem, there are several mobility solutions to work out, and Mobile IP is studied in more detail, as an example, in Section 5.2.[47]

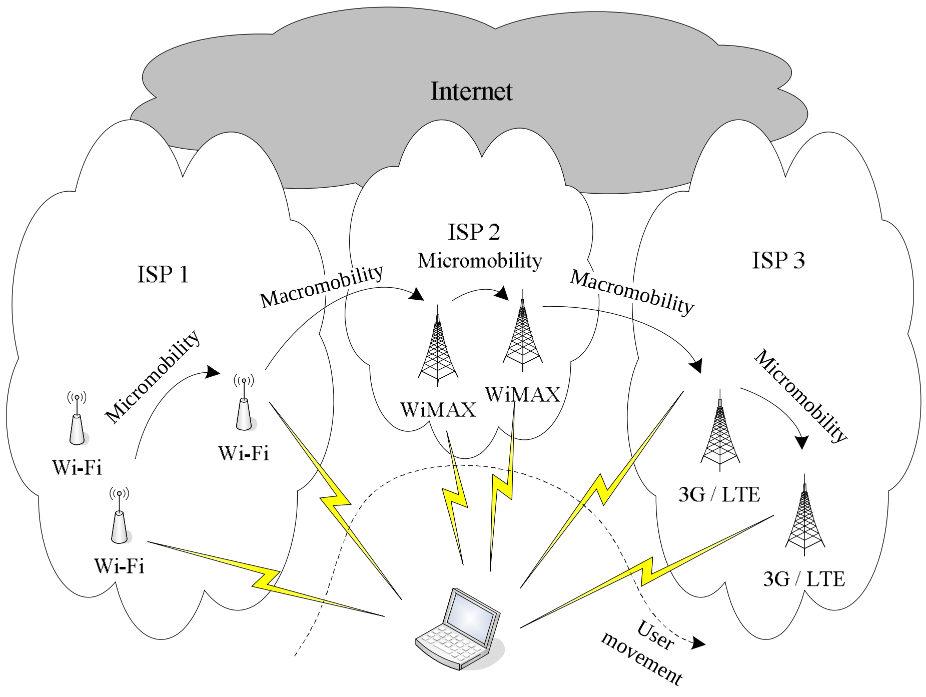


Figure 11. Multiaccess mobility scenario.

To make communication between end hosts more fluent while moving across several networks, simultaneous multiaccess or multihoming could be one possible solution. The end-host multiaccess node is attached to the network through two or more different access points, basically with multiple interfaces [51]. In practice, the node has several IP addresses for use in communicating with others. These interfaces can also be used simultaneously when network traffic goes through several links at the same time [52], for example, in Figure 11 when the node is on the boundary of the ISP1 and IPS2 networks with coverage of both Wi-Fi and WiMAX, the node can be connected to both interfaces. If it is reasonable, the node can also direct its traffic via both the Wi-Fi and WiMAX simultaneously or if, for example, it does not have enough battery, it can use only the Wi-Fi interface etc.

5.2. Mobile IP version 6

Mobile IPv6 (MIPv6) is a network layer (L3) macromobility solution for IP networks and is defined in [53] (IETF's Request for Comments 3775). There are at least three different entities in Mobile IPv6: a mobile node (MN), a home agent (HA) and a corresponding node (CN). In Mobile IPv6, each mobile node has a permanent home address with the home network's address prefix. When the MN is connected to the visited network, it also has a care-of address (CoA) with the visited network's address prefix, which is used for packet delivery to the visited network. The HA in the home network knows about the MN's care-off address due to bindings sent by the MN. Each packet sent to the home address by the CN is routed to the home network in all conditions. There, packets are delivered to the MN or the HA if the mobile node is in a visited network. If the MN is in a visited network, the HA tunnels packets to the mobile node's current CoA. After packets arrive at the MN, the node can send a reply directly to the corresponding node. If the CN also has MIPv6 support, it can send the next packet directly to the mobile node.[50][54]

Figure 12 represents the basic operation of MIPv6 with route optimization. So far the MN stays in the home network: everything works as normal without the mobile IP. After the MN has constructed a connection to the visited network and decides to move to it, the actual operation of the MIPv6 begins. The first phase of the handover from the home network to the visited network (1) is the configuration of the new connection. The configuration is usually performed automatically by the Router and Prefix discovery mechanisms. Afterwards, the connection establishing the MN can replace the old CoA with the new one, in this case only adding the CoA because the old network was the home network. The MN subsequently has to perform a binding update to the HA (2), in practice, tell the home agent that the MN can be found from that particular network with that address. The home agent has to acknowledge the binding before the next steps (3).[53]

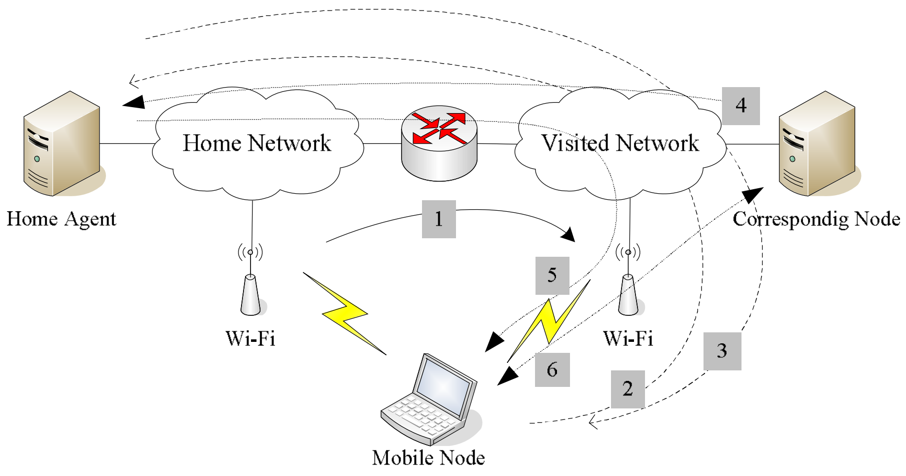


Figure 12. Mobile IPv6 example operation with route optimization.

When the CN wants to send data to the MN, it sends a message to the MN’s home address. In this situation, the message is routed to the HA (4) and tunnels the message to the MN’s CoA according to the last binding (with encapsulation, described in [55]) (5). The MN replies with binding information to receive the message directly by using normal Internet routing mechanisms (6). After that, communication between the CN and MN can be performed directly without the need of the HA.[53]

Every corresponding node that supports MIPv6 also maintains a binding cache. Before sending any packets, the CN checks the bindings for the current destination address and if there is a match it sends the packet to the CoA. Mobile nodes can also update bindings separately to the CN’s binding cache. It is updated to the HA afterwards. In this situation, communication between the CN and MN can naturally begin directly.[53]

Earlier operation of MIPv6 was described on condition that every participating party has mobile IPv6 support and route optimization is enabled. The CN does not usually have MIPv6 support and, in this case, all the traffic is routed through the HA using tunnelling between the MN and HA. This mode is called bidirectional routing and generates extra traffic to the HA and the home network compared with the previous route optimization mode. If the CN does not have binding for the MN, bidirectional routing is also used by adaptation. Packets are delivered through the HA, as otherwise the IP packet's source-destination address pair will be broken.[53]

5.3. Media Independent Handover Services – IEEE 802.21

Media Independent Handover (MIH) Services is a mechanism for providing handover-related link layer (L2) information to entities in the upper layers. It is defined by the IEEE working group in [56]. The MIH offers information about various types of communication links, for instance, to the mobility solutions, e.g., MIPv6 or a remote MIH implementation. Overall, MIH should allow a service to continue despite heterogeneous handovers.[57]

The design of IEEE 802.21 standard can be divided into three main parts: transparent service continuity, functions that enable handovers and service access points. The first part defines a framework enabling the service to continue while executing heterogeneous handovers. The transparent service continuity framework specifies mechanisms to gather all the required information for connection to a new access point before losing connection to the current access point. The second part of the design is the handover-enabling function that defines the Media Independent Handover function (MIHF). The MIHF is located logically between the Link and Network Layers and serves information about new access networks and any changes to current ones (from the Link Layer) to MIH Users (MIHUs), for example, to MIPv6 or adaptive applications. The MIHF does not make any handover decisions, it only gathers and delivers information about access networks. The Service Access Point (SAP) is an entity or interface between the MIHF and other participants in the handover preparation and execution such as MIHUs, the Link Layer and remote MIHFs.[57][58]

The general reference model of the IEEE 802.21 is illustrated in Figure 13. In the figure, the MIHF acts as a mediator between the Link Layer and the upper layer's MIHUs as described earlier. The Link Layer information to the MIHF is delivered via MIH_LINK_SAP. MIH_LINK_SAP provides an interface to receive information about heterogeneous links and to control these during handovers.

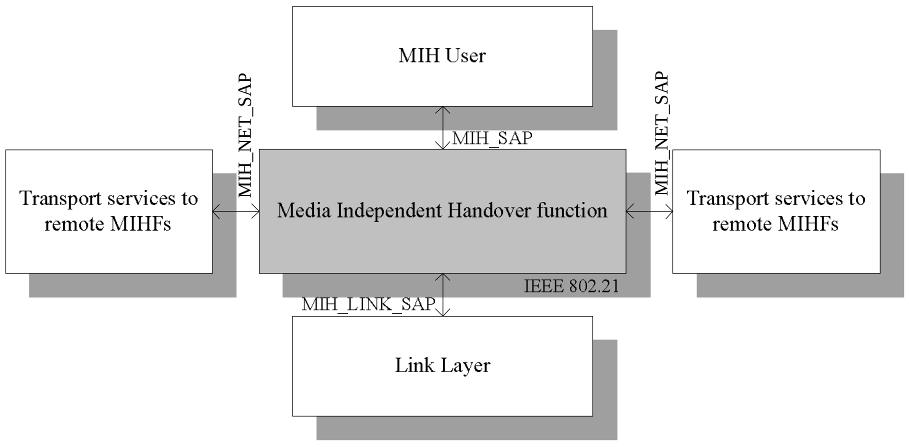

MIH User
MIH_SAP
MIH_NET_SAP
Transport services to remote MIHFs
Media Independent Handover function
MIH_NET_SAP
Transport services to remote MIHFs
IEEE 802.21
MIH_LINK_SAP
Link Layer

Figure 13. The reference model of the Media Independent Handover.

Currently, the standard supports Link Layer standards like IEEE’s 802.3, 802.11, 802.15 and 802.16 as well as standards defined by the third-generation partnership project (3GPP) and third-generation partnership project 2 (3GPP2). MIH_SAP provides an interface between the MIHF and MIHUs. The MIHF provides three different services through MIH_SAP: Media-Independent Event Service (MIES), Media-Independent Command Service (MICS) and Media-Independent Information service (MIIS). MIES provides event reporting to MIHUs. Events could be, for example, changes in link conditions, status or quality. Events could also come from a remote MIHF. MICS provides commands to control the parameters of handover execution and link behaviour. The commands can also be given to the remote MIHF as well as to the local one. MIIS enables MIHUs to receive information about current and other available networks. This information can be used, for instance, to choose the network to move next. Finally, the MIH_NET_SAP interface between the MIHF and the remote MIHFs provides a Link and Network Layer communication path to remote MIHFs. The communication can consist of information such as events or commands related to handover execution.[57][58]

5.4. Mobility Trigger Management

In a similar manner to the MIH, the Mobility Trigger Management system (TRG) operates as an information mediator between different layers of the TCP/IP model. The TRG can also deliver information to remote systems like the IEEE 802.21 does. IEEE 802.21 only delivers information used in mobility solutions, but the TRG can provide information from any TCP/IP layer to any another layer. The engine part of the TRG does not necessarily have to be located on same machine as in IEEE 802.21’s MIHF. The TRG architecture is therefore more generic than that of the IEEE 802.21 and more multipurpose.[8][47]

The general structure of Mobility Trigger Management is illustrated in Figure 14. As shown, the TRG is divided into three different parts: event sources, Triggering Engine and trigger consumers. Event sources provide information about fast-changing conditions in different layers to the Triggering Engine. These events could, for example, have information about congestion in a wireless channel as well as about a new available network in the area. The Triggering Engine delivers triggers to trigger consumers according to the identification number, filtering rules and triggering policies. The triggering consumer could be, for example, a BitTorrent application as in this thesis, and it could use the information to adapt its own operation to rise to the environment challenge. The consumer can also provide filtering rules to the Triggering Engine to confine the number of triggers.[8][47][59]

The Triggering Engine itself consists of four separate parts. Events are collected using the event collection interface. This particular interface is implemented with two alternative methods: a web interface to eXtensible Markup Language (XML) datagrams and a TCP socket interface to deliver information in string format. The Trigger processing module is used for filtering and also to time stamp the triggers. If a trigger has not delivered directly to the consumer, it is stored, and after some specified time it is removed from the repository.

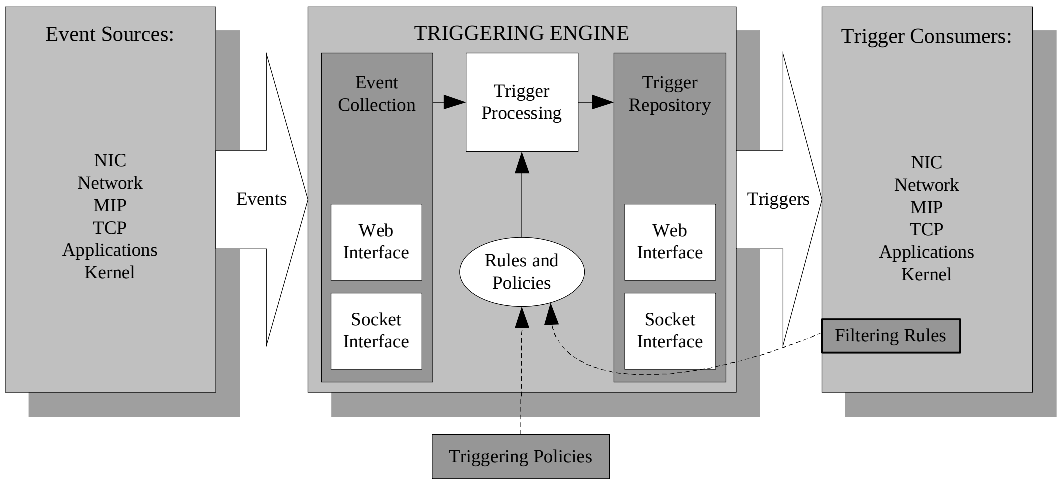


Figure 14. Mobility Triggering Management architecture.

The function of the trigger repository is twofold: it caches the triggers (in the cases described earlier) and delivers them to the consumers. The repository interface also consists of web and socked-based interfaces, as the event collection interface does.[8][47]

The very basic interaction between the Event source, TRG engine and Trigger Consumer is illustrated in Figure 15. The first of all the event sources is needed to register as a source to the Triggering Engine. After registration, the source can deliver events (triggers) to the TRG Engine, even without the existence of a consumer with that id. The TRG engine stores the trigger to a pre-defined time period and if the consumer does not exist within the time limit, the trigger will be removed. In the figure, however, the consumer subscribes as a trigger consumer before the source sends the trigger. As the source sends the event to the TRG Engine, it delivers the trigger directly to the trigger consumer. This is the procedure until the source or the consumer leaves the system. Leaving is done with UnRegister and UnSubscribe messages as shown in Figure 15.[8][47]

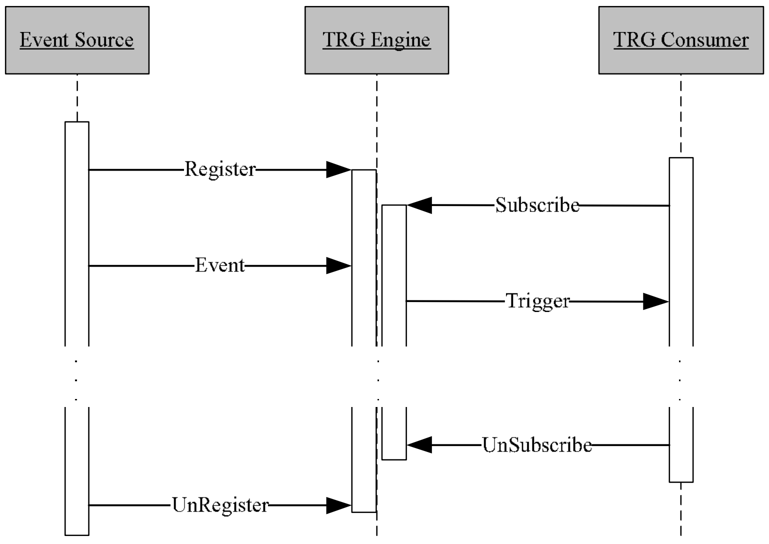


Figure 15. Signalling between triggering participants.

6. MULTIACCESS CONTENT DISTRIBUTION PROTOTYPE

Our first objective is to design and implement a proof-of-concept content distribution prototype for NetInf-based network architecture and provide some real evaluation results for the concept. The current peer-to-peer networks provide us with a good starting point for this, although these solutions are not well suited as they are to multiaccess scenarios in which the connection point of the network can change during the session or multiple connections can even be available simultaneously. The locality of the traffic is also taken into consideration, as growing cross-network traffic can waste limited network resources. As is pointed out in Section 6.1.1, resolution decisions made at an early stage can significantly affect the download performance, and we believe that they can also affect the overall traffic locality significantly.

This chapter starts with system design space (Section 6.1) by introducing experiments with conventional BitTorrent and real world torrent and describing the design phase of the prototype. The next section (6.2) presents our prototype implementation, describing the prototype architecture, signalling and extended message structures. Finally, we show how the prototype can be applied to the concept of NetInf.

6.1. System Design Space

In this section, we first present the experiments and results of conventional BitTorrent with a few Linux distributions in Section 6.1.1. We focused completely on the peer discovery process of downloads: how time evolution affects it, which portions different discovery methods have and the real usefulness of a peer discovered by a particular method. After the initial empirical study of BitTorrent peer discovery, Section 6.1.2 looks at the design of the prototype and how the former results can be applied to prototype design.

6.1.1. Experiments with Conventional BitTorrent

The design phase of the prototype started by studying the behaviour of a standard BitTorrent system in the real world. We performed experimental measurements to define the number of peers a BitTorrent client discovers through each of the three methods, namely: tracker announce, incoming peers and peer exchange (PEX). The experiments contain three different sets of measurements. We employed the aria2 multiprotocol download tool [60] and BitTorrent client part of the software. The aria2 BitTorrent client provides good logging capabilities that allow us to obtain valuable information about swarm behaviour and the overall progress of the download.

In the first set of measurements, a single client was connected in three experiments to a network using three different access technologies, namely: Ethernet, Wi-Fi (IEEE 802.11b) and WiMAX (IEEE 802.16e) at the VTT Converging Networks Laboratory (CNL) [61]. The mobile WiMAX access network is restricted with NAT from the rest of the Internet, whereas the other access networks (Ethernet and Wi-Fi) are not. We repeated the following experimental procedure ten times: the client contacts the tracker, receives a peer list, downloads a Linux distribution from peers received earlier and exits the swarm. This procedure was repeated for each access technology. Consequently, our client acts as a leech in all the measurements and does not upload (seed) the content after each successful download.

Linux distribution torrents were chosen specially because of their legal nature. We believe that, in general, the user behaviour of BitTorrent differs significantly when considering legal and illegal content distribution. The upcoming prototype is meant for legal use. Second, because the content size of the distributions is comparable to a full CD or DVD’s worth of storage, it acts as a proxy for the multimedia content (for example, low quality DivX or high-quality DVD movies). The first set of peers was discovered by the tracker, but while downloading the content our client discovered other peers through PEX and was able to receive incoming connections, except in the WiMAX experiments, due to the NAT restriction.

First of all, we try to define the importance of early stage peer discovery. Figure 16 presents the average total number of peers discovered per download through different access methods. The figure also presents the mean number of peers from which we download blocks of the file. In this figure, one of the ten downloads is not plotted due to problems with the results analyze scripts and the switchover day. This issue could take a while to fix, but as it does not affect the results significantly, the run was left out. We can see from Figure 16, for example, that when our client is connected to the Internet over WiMAX, it discovers, on average, a total of 166 peers. It successfully retrieves blocks from only 54 peers (33%) however. Peers from which our client has downloaded blocks are referred to as active peers in the rest of this thesis. The same number of active peers for Wi-Fi is 73 (41%) and for Ethernet it is 59 (41%), thus the ratio between the numbers of peers follows essentially the same trend. The stacked bars in Figure 16 represent the dynamic evolution of the peer discovery process by time from the beginning of the download. In our Fedora downloads, a significant number of peers were discovered within the first minute from the beginning of the initial announce to the tracker, which is specified in .torrent. In particular, 48% (WiMAX), 39% (Wi-Fi) and 69% (Ethernet) of the peers were discovered within the first minute of the beginning of the download (beginning from the first packet in the initial tracker announce). Most of the peer addresses are logged, on average, within five minutes of the download beginning with all the access methods (68% in WiMAX, 65% in Wi-Fi and 93% in Ethernet). The importance of early-stage discovery is certain by the mean number of active peer addresses. Three out of five active peers (from which we downloaded) are discovered during the first minute: 75% in WiMAX, 60% in Wi-Fi and 75% in the case of Ethernet.

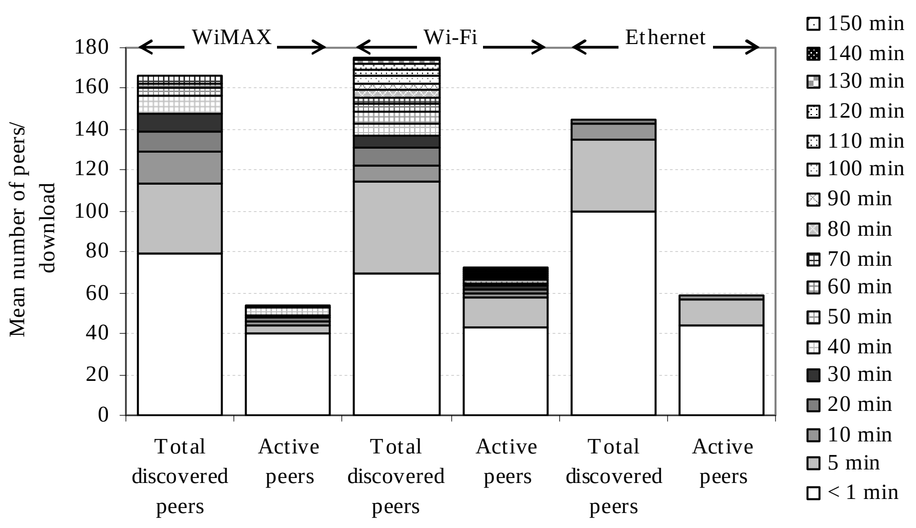


Figure 16. Mean number of peers/download stack plot ordered by discovery time.

As we know the importance of early stage decision in the peer discovery process, we like to categorize peers by the discovery method. Figure 17 presents the mean number of peers per download across the ten downloads, categorized by the discovery mechanism used. As can be seen from the figure, we were able to specify three different discovery methods during the experiments. The sets of bars in the figure were also classified as before, according to three different network access methods. In the WiMAX experiments, there are no incoming peers because the NAT prevents all incoming connections. Incoming peers are ones that join the swarm later than our client, are most likely to receive our location from the tracker and contact our client. In all three cases, our employed client discovers many more unique peer IPs through PEX than incoming connections or through the tracker, as shown in the figure.

Our client does not interact too much with peers discovered via PEX however. Fewer than half the peers from which we download are discovered from PEX. Useless peers discovered through PEX could possibly be “old” peers that have been in the swarm and have stayed, for some reason, in the remote peer’s peer list for a while. The figure also provides us with information on the importance of tracker announces where most of the active peers were obtained. Nonetheless, the conducted measurements are not representing behaviour of all swarms on the Internet today or even for all legal content swarms; they do point to the fact that the initial list of peers plays a big role in content retrieval.

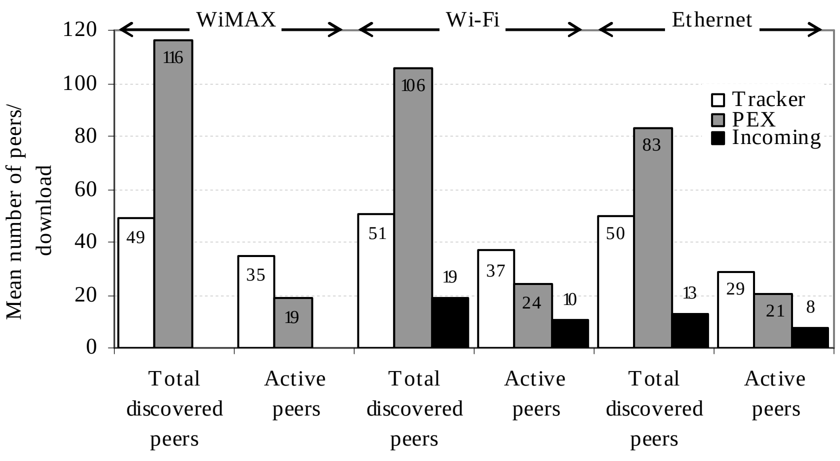


Figure 17. Mean number of peers classified by the discovery mechanism used.

As our client has discovered the peers and we have defined the popularity of different methods, we would like to know the degree of keeping with the original set of peers discovered from the tracker. We also want to know the real benefit of employing alternative mechanisms for discovering peers, such as PEX. As described in 4.5.2, PEX provides a different view of the swarm for our client, introducing new active peers, but as discussed before, it only provides a small, but still significant, part of the active peers. This view overlaps the tracker-provided list however, for example, tracker-provided peers are very often repeated in PEX lists (when a peer is "rediscovered", it is not summed to PEX but to the initial method).

At the time of the measurements, April 2009, Fedora 9 distribution had almost reached the age of one year. According to the power law nature of BitTorrent content with a long tail [43], the distribution .torrent was quite "old" (official publication date in May 2008). PEX-discovered peers may be of less worth than expected for archival torrents, such as Fedora 9, and the portion of new arrivals (incoming peers) could also be lower.

Cumulative downloaded bytes of ten runs for Fedora 9 over Ethernet and Wi-Fi is illustrated in Figure 18. We do not plot results from WiMAX experiments because the NAT prevents all incoming connections, and the figure can therefore be distorted. The aim of this analysis is to define the real importance of peers discovered by each of the three methods. Bytes received by a particular peer are classified, according to the corresponding discovery method, by three-colour coding (tracker – white, PEX – grey and incoming peers – black). The figure does not restrict the case in which the same peer is discovered through one method at the first download either, and all the bytes in the upcoming downloads are counted for this method. On the contrary, the discovery method is examined separately in every download, for example, if a peer's IP address is listed in the tracker response in one of ten downloads, all the downloaded bytes from this peer in this particular download are marked in white. In another download, the same peer IP can be discovered through PEX and all the bytes are then marked in grey etc. For the Fedora download set, peer indication threshold is 100 MB for space saving purposes (total content size 3580 MB).

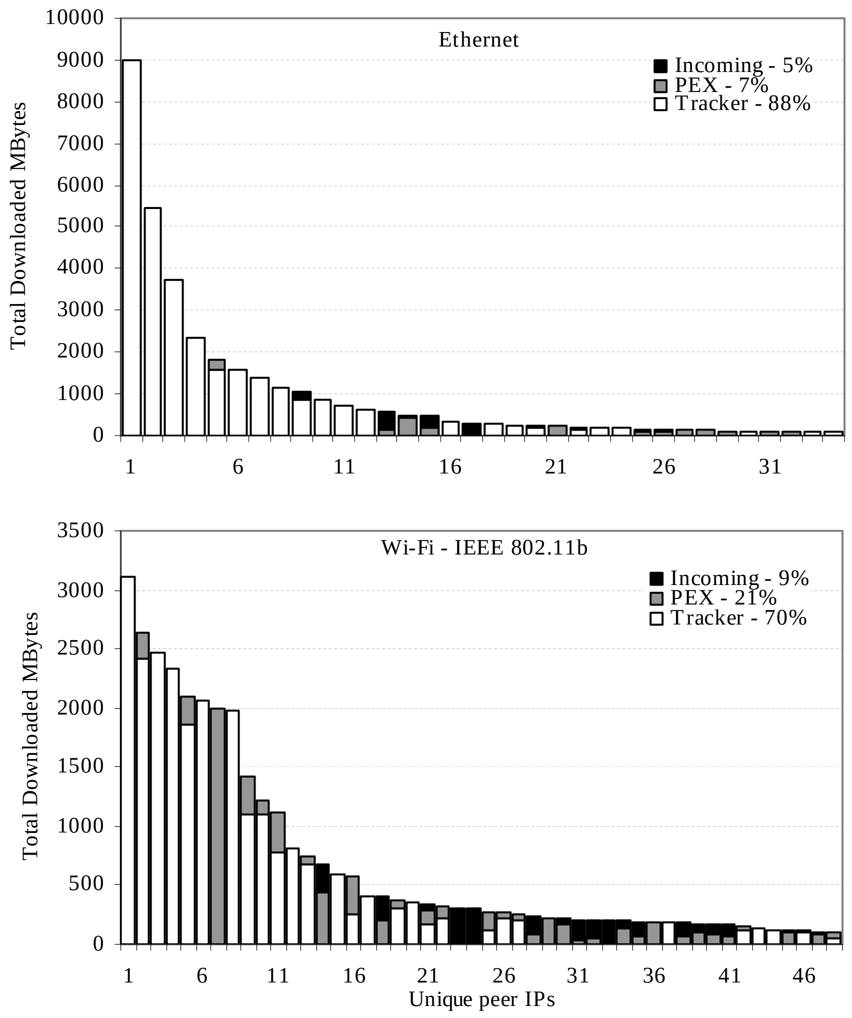


Figure 18. Cumulative application payload received in ten downloads (Fedora 9).

The average duration of downloads over Ethernet is (9 min) an order of magnitude less than over Wi-Fi (127 min) due the difference in the respective network access capacities. This difference in durations plays an important role, as surmised from Figure 18. When we download Fedora 9 distribution over Ethernet, the client keeps to the peers provided by the tracker. As the download completes quickly, blocks obtained from peers that are not listed in the tracker response provide on average only 12% of the total bytes, as can be seen from the upper plot in Figure 18. When we download Fedora 9 over Wi-Fi, alternative methods such as incoming and PEX peers play a bigger role. This can be explained as being mainly due to the longer duration of the download, which exceeds 2 hours on average. With such long retrieval durations, incoming peers have the necessary time to distribute blocks with our client. An increase in the average download duration from 9 min to 127 min results in, on average, only 6 more incoming peers (in Figure 17), and this is also an argument for the oldness of the Fedora 9 distribution.

The archival nature of Fedora 9 distribution most probably affects the swarm behaviour in such a manner that the content is more likely to be distributed over infrastructure seeds (seeds that are set up by, for instance, distribution providers etc.), than peers that have only cached the content (the top peer by provided bytes in Ethernet delivers 25% of the Fedora distribution). We therefore conducted the following two sets of measurements, this time downloading the Ubuntu 9.04 and Ubuntu 9.10 distributions. With Ubuntu 9.04, the setup used was the same as that described above, but the downloads were performed using only Ethernet and Wi-Fi (IEEE 802.11b) access technologies. In the case of the Ubuntu 9.10 experiment, the setup was the same as with 9.04, but the Wi-Fi interface card was replaced with IEEE 802.11g standard device. Once the content sizes of both Ubuntu distributions are less than in the case of Fedora 9 (v 9.04 – 730 MB and v 9.10 – 724 MB), the threshold for peer indication in cumulative payload by peer figures (Figure 19 and Figure 20) is 35 MB.

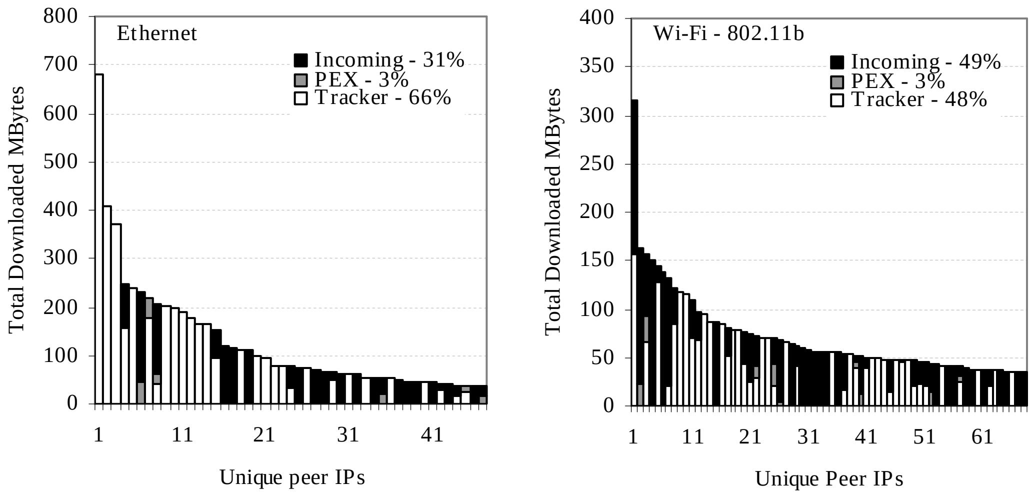


Figure 19. Cumulative application payload received in ten downloads (Ubuntu 9.04).

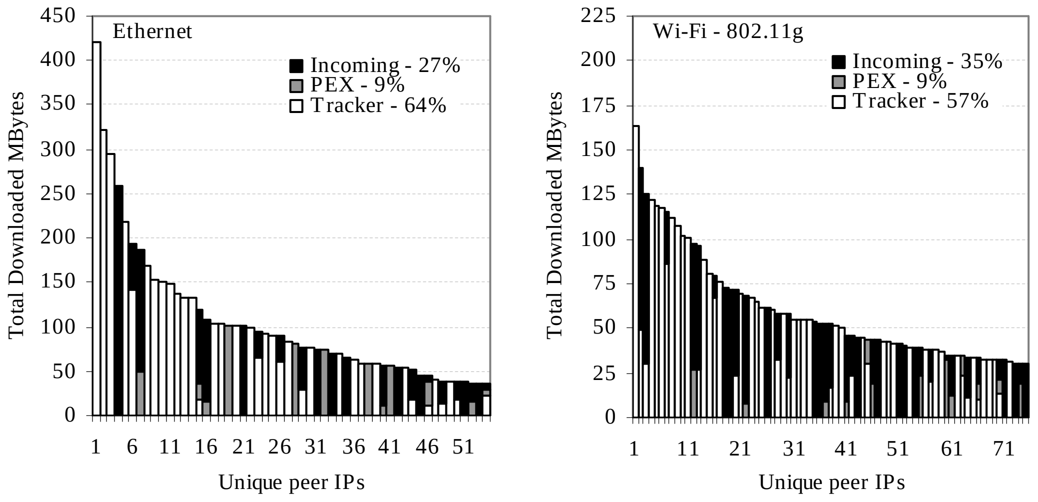


Figure 20. Cumulative application payload received in ten downloads (Ubuntu 9.10).

Apparently, peers discovered from the tracker once again play a big role, as shown in Figure 19 and Figure 20. As these two distributions have more active swarms (release published: v 9.04 in April 2009 and v 9.10 in October 2009), however, incoming connections take a larger proportion of the Ubuntu distribution blocks than before. As the experiments were performed over Ethernet, the majority of the blocks was downloaded from peers provided by the tracker, with percentages of 66% and 64%, again playing as a major peer discovery method.

The results of the download of Ubuntu distributions over Wi-Fi show that 49% of the distribution is retrieved from peers that join the swarm later and 48% from tracker-provided peers in the Ubuntu 9.04 downloads. The same percentage numbers for the Ubuntu 9.10 downloads are 35% (incoming) and 57% (tracker). Across ten downloads over Wi-Fi, our client found 344 unique active peers in the Ubuntu 9.04 experiments and 444 in the Ubuntu 9.10 case. From this we assume that 9.10 has the most active swarm. By following the same logic as before, however, the biggest portion of bytes should come from incoming peers. In the Ubuntu 9.10 case, the used Wi-Fi interface card was specified with the IEEE 802.11g standard and the increase in the average duration of downloads moving from Ethernet to Wi-Fi (4 min to 11 min) is therefore not so significant compared with the Ubuntu 9.04 case (3 min to 37 min). This provides an explanation for the small size of the download portion of incoming peers in the Ubuntu 9.10 case (the major part of blocks from the tracker-provided peers) compared with Ubuntu 9.04 (the major part of blocks from incoming peers).

To summarize the experiments, the early stage peer discovery decisions play a very significant role. In all the download sets, the blocks downloaded from the tracker-obtained peers have a very significant impact portion of all three methods. Apart from the 9.04 Wi-Fi case, in all the download sets the majority of blocks came from tracker-obtained peers. Besides, in this particular case the difference between incoming and peers from the tracker was extremely small. It is also good to note that in the case of incoming peers the tracker plays a central role as well, as they also discover our client from the tracker, and this fact raises the level of importance of peers discovered from the tracker.

6.1.2. Prototype Design

As mentioned in Section 2.1, NetInf introduces three main scenarios with a strong information-centric background, namely: content distribution, augmented Internet and personal mobile scenario. This thesis focuses strongly on content distribution. The personal mobile scenario is also heavily involved due to the directions of recent research and also due to the expectations of Internet connection everywhere by people. Last but not least, other issues in current peer-to-peer networks presented in Section 4.6, such as cross-ISP traffic, should be taken into consideration.

Our main objective in this thesis is to design and implement an improved content-distribution prototype by extending current BitTorrent. The prototype is aimed at outperforming the current BitTorrent with mobility and tries to retrieve content as near the network as possible. The prototype should not only perform sufficiently in a multiaccess environment, but also provide an opportunity to employ multiple interfaces simultaneously whenever possible due to the available access points (or base stations) or, for instance, when the user machine has enough battery resources.

As described in Section 5.2, MIPv6 could be used for a mobility solution for IP networks. The TRG architecture described in Section 5.4 can be used to deliver additional information such as congestion in a wireless channel, for instance, to a mobility solution, as done in [47]. Similarly, we can use the information received from the TRG engine to the tracker to reannounce or even redirect the announce to another tracker (see the significance of the tracker-provided peers in Section 6.1.1) in our prototype. The MIPv6 standard implementations, however, do not have support for simultaneous multiaccess or multihoming, which should be one of the key features of upcoming prototype implementation. We therefore do not employ MIPv6 in it. Nevertheless, the TRG architecture is still used in the prototype as an information mediator.

Nowadays, multiple trackers at the same .torrent are consumed for load balancing or back-up purposes, as described in Section 4.5.2. Our concept is division of trackers into two levels: one centralized tracker (and also only one specified in .torrent) and several lower level trackers. The main tracker could, for example, take care of the entire swarm, whereas lower level trackers could be assigned to maintain peer lists in its own areas. A natural area for the lower level tracker in the network infrastructure described in Figure 21 is the single subnetwork. Several approaches to future networking are introduced into hierarchically divided network architecture, for example, as zones in MILSA (Section 3.4), and this is exactly the case we have in Figure 21 with the placements of trackers. From now on, we will call our subnetworks domains, and the rest of the network a core network. As is obvious, the lower level trackers will be placed into domains, with one lower level tracker maintaining peers in its own domain. Later on, lower level trackers are referred to as SubTrackers. In principle, a centralized tracker (will be referred to later as CoreTracker) could be placed anywhere in the network, but we will place it in the Core Network (Willab in the figure).

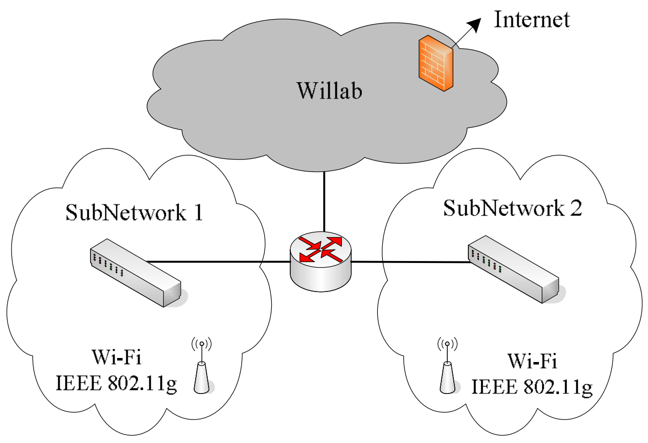


Figure 21. Network infrastructure used in the design of the prototype.

As described in Section 4.6, one of the major issues in peer-to-peer networks is the cross-ISP traffic generated. One of NetInf’s name resolution principles presents the idea of traffic locality: the content should be retrieved from local resources whenever possible even in cases without connectivity to the rest of the Internet. To achieve these goals, our prototype could direct or redirect the first announce (see the importance of early stage announces in 6.1.1) toward the SubTracker in the domain to which it is connected.

The discovery of the SubTracker could be problematic, however, as the .torrent only has information about the CoreTracker. The first option is to broadcast a query toward the domain, but in cases in which the content is not available in the local domain, the prototype does not outperform conventional BitTorrent. To handle this problem, the location of the SubTracker can be asked for from the CoreTracker, but this solution requires SubTracker registration to the CoreTracker. In the case of registration, the CoreTracker has to maintain two different lists: one on all the peers in the swarm and one on the SubTrackers. This can cause scalability issues, and we therefore decided to divide the swarm into several parts, one for each tracker (in this scenario two SubTrackers and a CoreTracker). As is certain, SubTrackers keep up the list of peers in their own domain, and the CoreTracker maintains the list of peers in the core network, which is not part of any domain and does not want to distribute the content locally (for instance, the initial seeds). This particular core swarm can also act as a backup, used in those cases in which the content cannot be downloaded locally. Methods used with, for instance, DISM (see Section 4.6.5), can also be applied to the presented scenario. With the represented two-tier tracker hierarchy, the implemented client does not need to bias its peer connections because it only knows about peers in the local domain. Localization is performed by biasing in, for example, 4.6.3 and 4.6.4. We therefore do not need to modify the peer wire protocol, which can be used as it is. This will ensure at least partial compatibility with the conventional BitTorrent system.

As the traffic locality is taken into account, the next consideration is mobility and multiaccess. Due to the lack of a mobility solution, we need to perform some actions while carrying out the handover from one domain to another. One solution to handling the handover is to perform an announce to the new domain’s SubTracker and receive local peer(s) from there. This also provides an opportunity to employ multiple network interfaces simultaneously. The download could be started using one interface, whenever the second interface has established connection to another domain, the SubTracker in this domain can be announced and the client take contact with other peers using this particular interface. The TRG can provide the information on connection to the new domain. With this kind of overall architecture, the local peer does not stick with same set of peers while moving around the Internet as the MIP does. Our proposed architecture tries to find peers “behind” access point (in same domain as the AP) and, moreover, considers the actual content retrieval instead of staying in contact with the same peer set for the duration of the download. This way of thinking is one of the key elements of the information-centric networking paradigm and the advantages based on the idea are promising.

6.2. Implementation

One of the key decisions at the initial stage of the implementation phase was to find the most suitable BitTorrent client software for our purposes. The selected software should fulfil characteristic requirements such as support for Linux/Unix-based operating systems, free and open source and also IPv6 compatibility. All the hosts in our prototype run the Linux-based Ubuntu 8.04.3 LTS operating system and the client should therefore be compatible with it. The free and open-source software requirement is needed for the agreement of freely made modifications and use of the client. The client should also have support for IPv6 addresses. Information on the alternative client implementations is summarized in Table 3. We have not only considered information on the characteristics described before, but the complexity of the implementation also plays a major role. The aspect of the popularity of the client (used and references in scientific publications) was also taken into account.

Table 3. The BitTorrent client comparison table

Name:	Free and Open Source:	Language:	IPv6:	Comments:
ABC	ok/Python Software Foundation	Python		
Aria	ok/GPL	C++		
BitTornado	ok/MIT	Python	ok	Simple
BitTorrent 5	ok/BitTorrent Open Source	Python		
Deluge	ok/GPL	Python/C++		
KTorrent	ok/GPL	C++	ok	Very wide
Transmission	ok/GPL/MIT	C/Cocoa	ok	Wide
Vuze	Proprietary/GPL	Java/SWT	ok	
Unworkable	ok/primarily OpenBSD	Portable ANSI C		Very simple

As we can see, three different client implementations emerge from the table: BitTornado, Transmission and Unworkable. Unworkable is very simple but has no support for IPv6. The transmission BitTorrent client seems to satisfy our needs, but the complexity of the realization affects the decision. The study of the complex implementation will take much time and the selection of the Transmission can therefore be a waste of limited resources. BitTornado [62] satisfies our needs and is relatively simple to extend. The distribution package also provides implementation for peer tracking, which is advantageous if we want to modify the tracker entity. Thus, by selecting BitTornado, the study of another implementation architecture is taken out of the account. The evident weak point of BitTornado is the lack of ongoing development. The latest version was published in 2006. We believe, however, that the major bugs have been solved, and our choice of client and tracker software is BitTornado. After a quick code study, we become aware of the remaining issues of BitTornado’s IPv6 compatibility, but a patch for these issues is provided in Appendix 1.

6.2.1. Prototype Description

As discussed in the context of designing the content distribution prototype, our prototype should avoid cross-domain traffic, enable the use of multiple interfaces simultaneously and provide an indication method of new available domains in the area. The low-level piece delivery steering to local copies of the content is achieved with the two-tier tracker hierarchy (CoreTracker – multiple SubTrackers) introduced earlier in 6.1.2. Multiple interfaces can be connected to access points in different domains in a leech capable of multiaccess, and in this sense interfaces could be used to retrieve pieces from local peers in their own domains simultaneously. Finally, the third approach to the prototype implementation (indication of new domain) is achieved with the employment of the TRG. This last part of the prototype is the NetInf Notification Service (NNS). Another approach to the NNS is the standardized (IEEE 802.21) Media Independent Handover Services (introduced in Section 5.3) and employing it instead of the TRG, as will be discussed in Section 7.3.

Figure 22 illustrates the scenario, considering the implemented prototype, and comprises two access domains, a third domain, namely Core Network, and a router between them. All the domains have different global IPv6 address spaces (because they are implemented as separate subnetworks) and both access domains have IEEE 802.11g access points. Equally, the access points could be, for example, 3G or IEEE 802.16 standard devices. The Core Network has a connection to the rest of the Internet. A multiaccess (MA) leech is equipped with two identical wireless network cards and connected to its own access points.

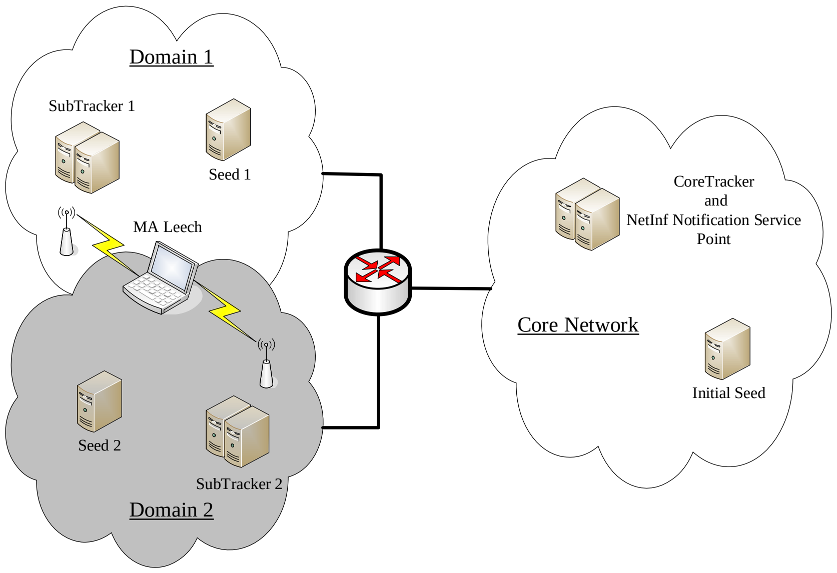


Figure 22. Employed scenario in the prototype implementation.

Each of the domains presented in Figure 22 has its own trackers. Both access domains have their own SubTracker, and the Core Network has a CoreTracker and NetInf Notification Service Point (NNSP), which in our prototype implementation is based on the TRG triggering engine. The initial seed is positioned in the Core Network and access domains also contain their own seeds once they have downloaded the content from the initiator seed. Traffic localization is coordinated with an extended BitTorrent client-tracker and tracker-tracker HTTP-based communication. This and the operation of NNS are explained next.

Figure 23-Figure 26 present the signalling governing the main operation of the implemented prototype. We consider two cases. The operation of the single-access BitTorrent peer is presented in Figure 23 and Figure 24, describing leeching and seeding. The operation of the multiaccess peer is presented in Figure 25 and Figure 26, describing leeching only. Seeding in multiple domains can be the same from a signalling perspective. We will assume that the content first becomes available in the core network only, whereas the initial seed exists. In practice, the existence of the file in the Initial Seed only means disappearance of Seed 1 and Seed 2 in Figure 22. To start with system bootstrapping (Figure 23), the CoreTracker registers with NNSP as an information source with identification number y (message 1) and subscribes with NNSP as an information sink with identification number x (message 2). As the leech in domain 1 wants to download the file, it starts by announcing the torrent from the CoreTracker (3.). The leech indicates the domain to which it belongs in message 3 with the domain identification number parsed from the current IP address. As domain 1 does not have a complete local copy of the file, after receiving the response (4.), the leech starts to download the requested content from the Core Network using standard peer wire protocol. The locator of the NNSP is also received in the response, and as the leech does not have multiaccess capability, it discards it. For a safe shutdown, the leech indicates its departure from the swarm by a standard announce (5.) and the CoreTracker acknowledges it with an announce reply (6.). The signalling from 3 to 6 is mainly the same (without information on the domain identification and NNSP location) as that of the standard BitTorrent HTTP protocol.

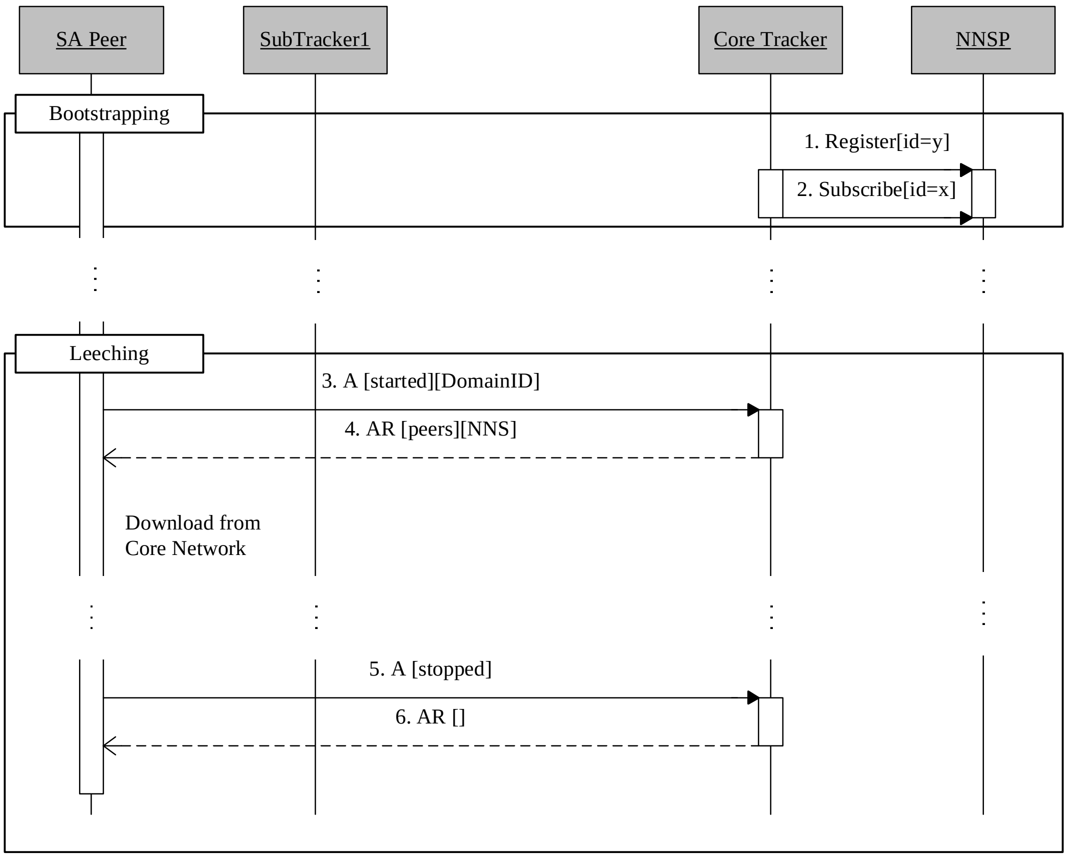


Figure 23. Signalling in single access leech content retrieval from the core network.

In Figure 23, the single access leech exits the swarm before it has retrieved the entire file. If this is not the case, messages 5 and 6 can be replaced with Figure 24. Once the SA Leech has downloaded the entire file, it can seed it in its access domain whenever the SubTracker is available (as it should always be). First, the leech exits the swarm in the Core Network in the same way as earlier (5. and 6.), as it does not want to upload the file for peers that are outside its domain. Subsequently, the leech locates the domain SubTracker (as it is connected to domain 1, SubTracker 1 in the case) in its domain. At this time, UDP broadcasting was used to locate SubTracker 1, but other methods are possible. The leech sends the broadcast message with the string ‘SubTracker’ (7.) to a separate location server at a specific known UDP port and SubTracker1 receives it and delivers its own location in a reply message (8.).

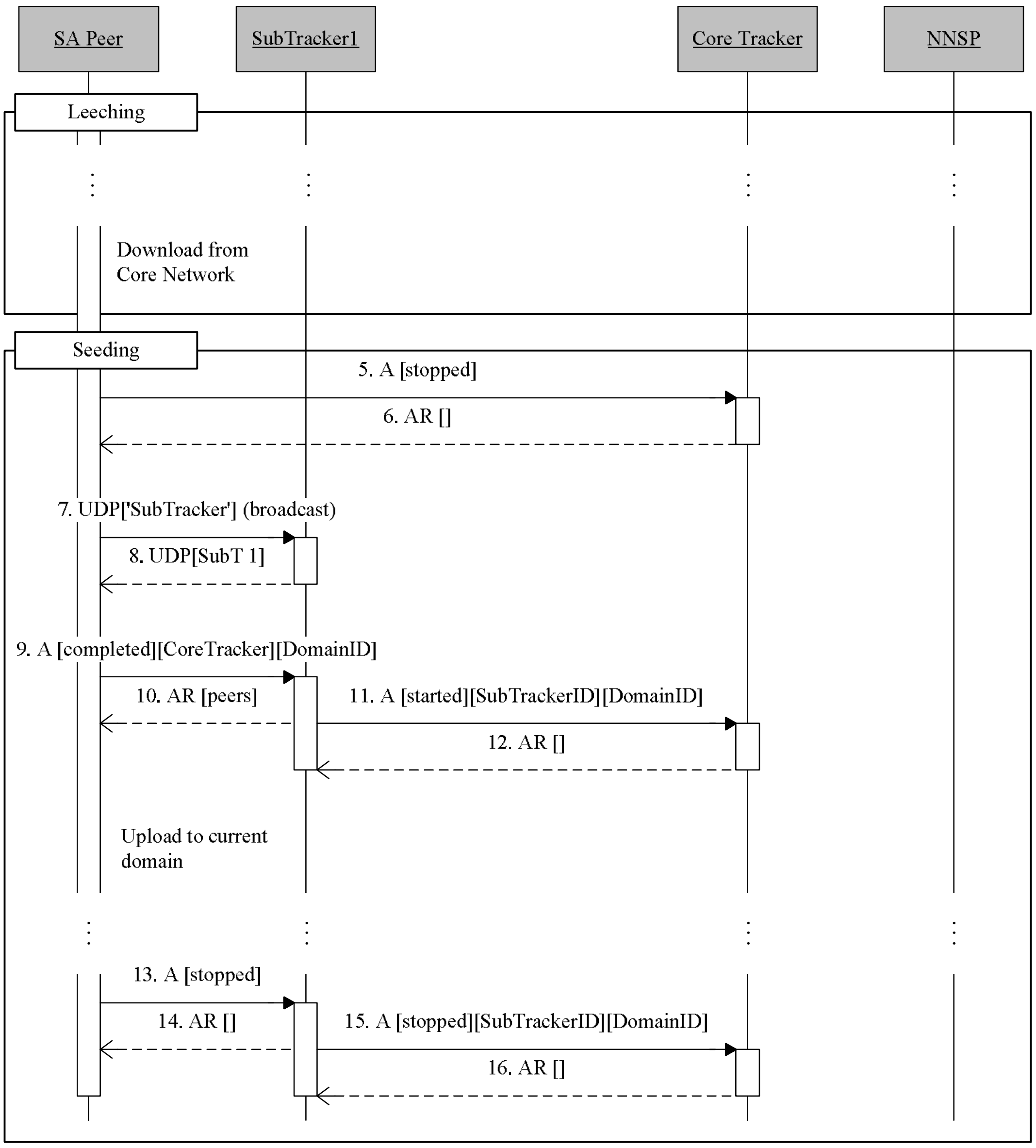

SA Peer
SubTracker1
Core Tracker
NNSP
Leeching
Download from Core Network
Seeding
5. A [stopped]
6. AR []
7. UDP['SubTracker'] (broadcast)
8. UDP[SubT 1]
9. A [completed][CoreTracker][DomainID]
10. AR [peers]
11. A [started][SubTrackerID][DomainID]
12. AR []
Upload to current domain
13. A [stopped]
14. AR []
15. A [stopped][SubTrackerID][DomainID]
16. AR []

Figure 24. Signalling in a single access leech changeover to seed.

Hence, the leech can join the local swarm in domain 1 by announcing SubTracker 1 (9.). This particular announce message is made special with the location of the CoreTracker because the SubTracker does not necessarily have the information about it. SubTracker 1 replies with a peer list (10.), which should in this case be unoccupied. In addition, SubTracker 1 informs the CoreTracker (11.) of the availability of a local copy of the file in its domain, and the CoreTracker responds with an empty peer list (12.). From now on, the leech can upload the content locally in domain 1. Messages 13 and 14 show how the leech exits the swarm. Moreover, in the case that domain 1 no longer has the entire file, SubTracker 1 unregisters itself from the CoreTracker's SubTracker file list (15. and 16.). If domain 1 has duplicated the file to another seed, however, SubTracker 1 will not unregister itself from the CoreTracker's list.

After an explanation of the appearance of the local seed, we will assume that both domains have seeds as shown in Figure 22 and have seeds registered to the

SubTrackers as explained previously. In the case of the leech with multiaccess capability (in Figure 25), the leech announces the CoreTracker (17.) in the same way as message 3 in Figure 23. The CoreTracker response (18.) is the same as in message 4, but instead of giving peer locations in the response, the CoreTracker replies with the location of SubTracker 1 (because the content has a copy in domain 1).

In the case of the leech with multiaccess capability, as it is now, it registers with the NNSP (locator obtained in 18.) as an information source with the identification number x and as an information sink with identification number y, vice versa to messages 1 and 2 (registration is not needed for the SA leech retrieving the content from the local seed). In this manner, the NNS message sent from the leech is delivered to the CoreTracker and the NNS sent from the CoreTracker is delivered to the leech. NNSP is shown as a single entity throughout this thesis, but nothing prevents the use of multiple service points in a distributed manner. NNSPs could, for example, be placed at all MA peers and domains, and these entities could exchange information as a distributed network. After NNSP registration, the leech proceeds with a standard announce message (21.) to local SubTracker 1, acting as the DNS (in Section 3.1) like announce redirection. SubTracker 1 responds with the peer list (corresponds to the swarm in domain 1) (22.). Finally the leech starts downloading the file from the peer(s) in domain 1.

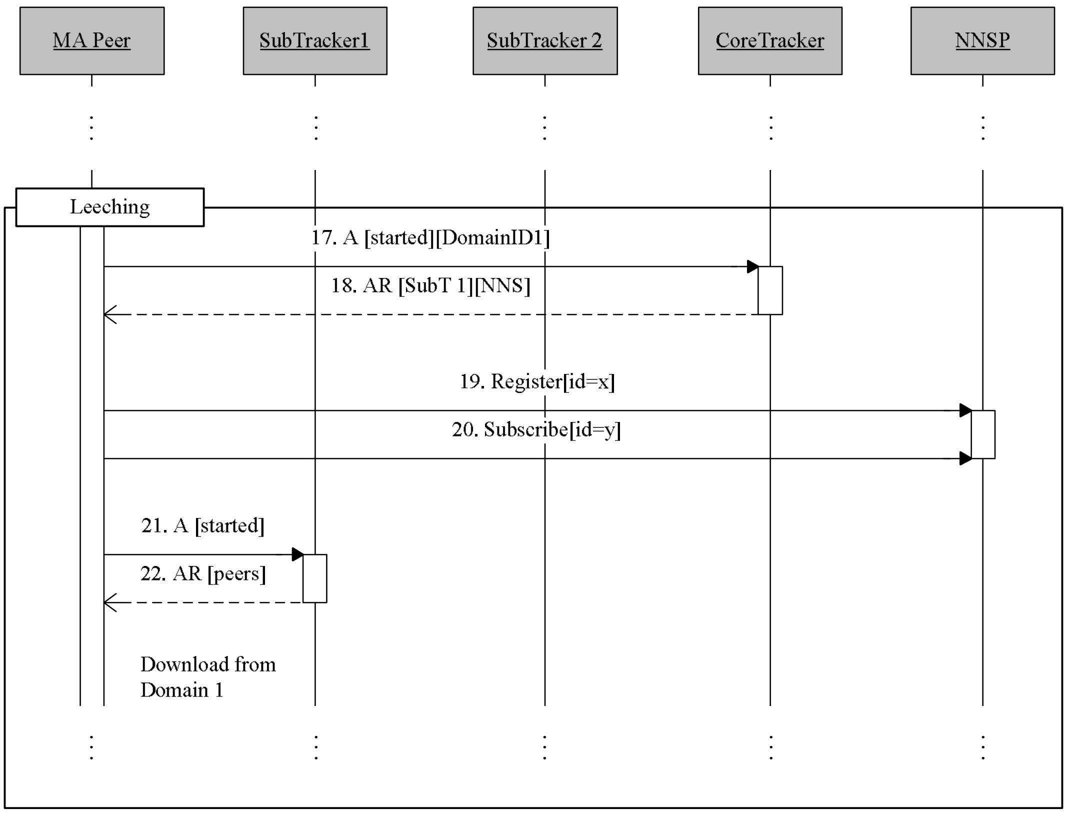


Figure 25. Multiaccess leech signalling in the download bootstrapping phase.

Figure 26 represents signalling and information exchange from the appearance of a connection with the second domain to a graceful leech shutdown. After a second interface becomes active (indicated by the appearance of a new IPv6 address, which is implemented at this moment as a separate program) and is connected to domain 2, the leech uses NNS to locate SubTracker 2.

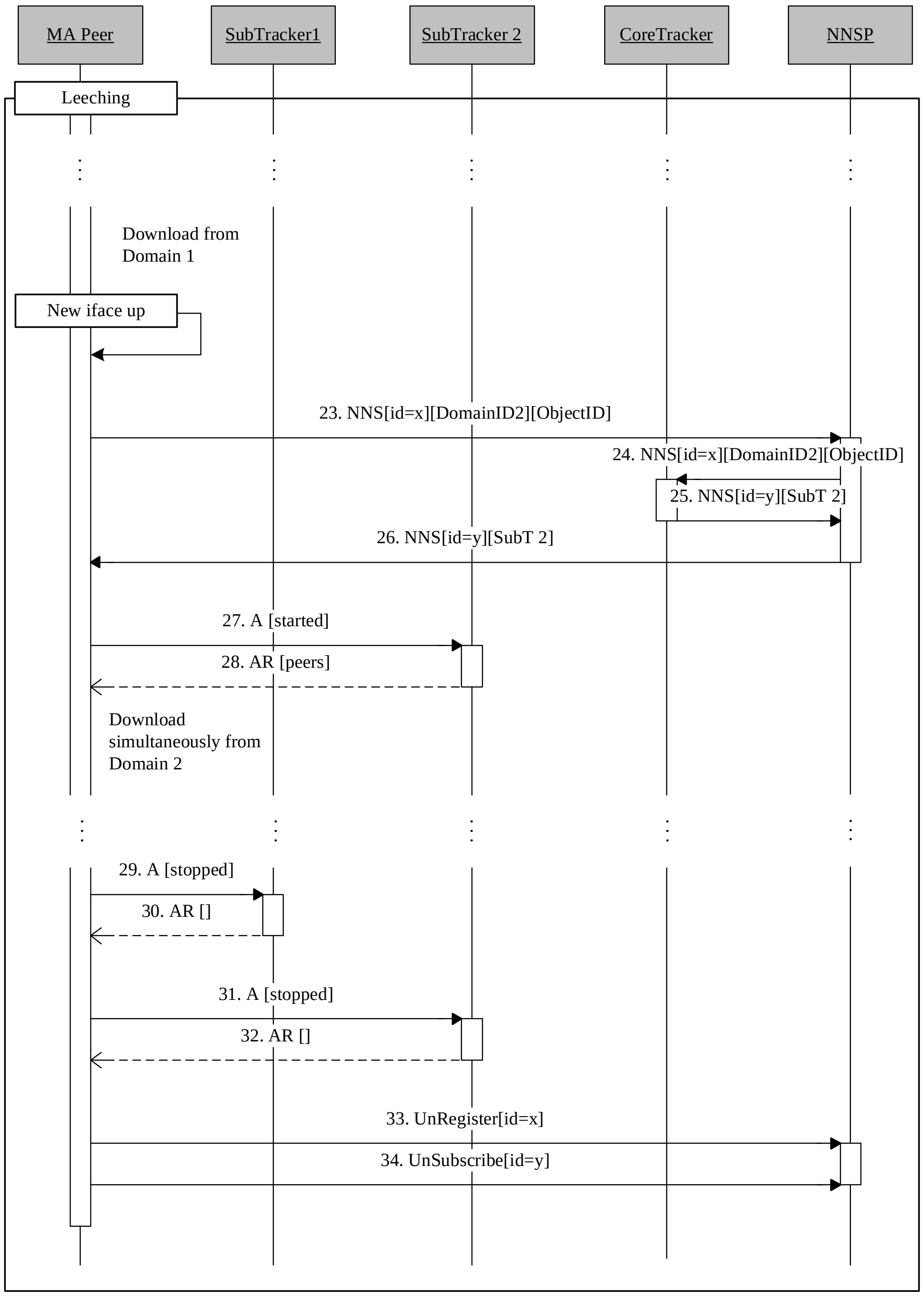


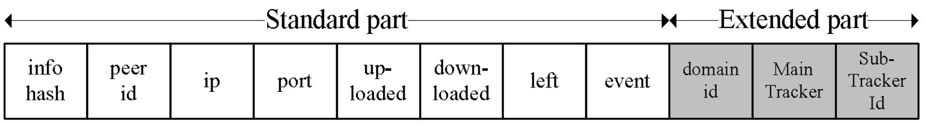
Figure 26. Multiaccess leech signalling from the emergence of a new domain to shutdown.

At first, the leech sends the NNS message to the NNSP carrying the domain and file identification details (DomainID and infohash) (23.). The NNSP delivers the NNS to the CoreTracker (24.) while the CoreTracker checks the content availability in domain 2. If the local copy of the file exists in domain 2, the CoreTracker responds with the locator of SubTracker 2 throughout the NNSP (25. and 26.); otherwise it does nothing. After the leech has obtained the location of SubTracker 2, it can announce again by the standard BitTorrent announce SubTracker 2 as joining the local swarm in domain 2 (27.). SubTracker 2 replies with a peer list (28.) containing peers in domain 2. After that, the leech starts downloading from the peers in domain 2 in parallel with the ongoing download from the peers in domain 1. When the leech wants to leave the content distribution sessions, it has to exit either local swarm by announces (message 29 and 31) and unregister (33.) and unsubscribe (34.) from NNSP.

6.2.2. BitTorrent Message Extensions

The prototype signalling described in 6.2.1 requires extensions to the standard BitTorrent protocol message structure. Implemented extensions are related to the tracker announce and announce reply messages, and the tracker HTTP protocol messages therefore needed to be extended. BitTornado's existing tracker-to-tracker communication messages, in practice, the tracker announce made by another tracker, had to be extended. Tracker-to-tracker communication uses the same message structures as the peer-to-tracker communication, however, and therefore we only need to extend the HTTP protocol messages.

According to Section 4.5.2, the tracker HTTP protocol involves two different types of message, namely, announce and announce reply; both these need a few extensions. The required extended fields for the announce messages are: *domainid*, *MainTracker* and *SubTrackerId*, as shown in Figure 27.



- ***'domainid'***: *identification number of domain requesting peer is connected to*
- ***'MainTracker'***: *network location information about Core Tracker (URI)*
- ***'SubTrackerId'***: *unique identification string of requesting SubTracker*

Figure 27. Extended Tracker's announce.

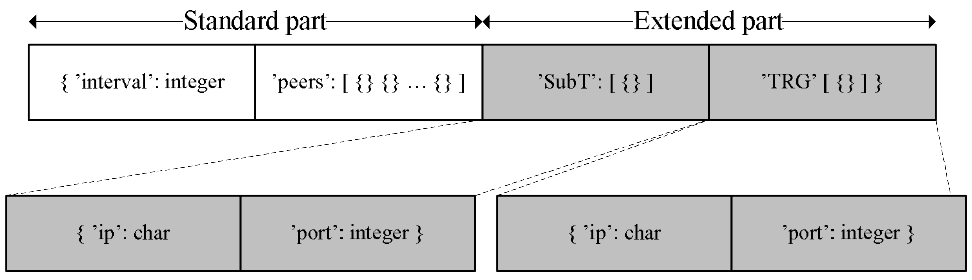
The extended announce reply message is shown in Figure 28, and the field extensions are *SubT* and *TRG*. All BitTorrent protocol additions are aimed at maintaining backwards compatibility as far as possible. We did not avoid all the issues related to this, however, and these are discussed in more detailed in Section 6.2.3. The addition of extra keys to both message types does not impact receipt of the announce message by the tracker and the announce reply by the peer; they simply discard the extra fields, making this a safe solution from a compatibility perspective.

Figure 27 represents the extended tracker announce message with all the extension fields. The fields are optional, however, and not bound together, for instance, if domain identification is needed to deliver to the tracker, it is not necessary to include the *MainTracker* and *SubTrackerId* fields. The domain identification number or *domainid* is used to indicate the domain to which the requesting peer is connected and from where there are wishes to retrieve the file. The CoreTracker brings in the decision on the content of the announce reply message accordingly.

In practice, the CoreTracker compares the received *domainid* and *infohash* that the SubTracker list maintains. If there is no match from the comparison, the CoreTracker replies with a peer list corresponding to the core swarm, as in Figure 23 (3. and 4.). If there is a match, the CoreTracker replies with the SubTracker location (*SubT* in Figure 28) as in Figure 25 (17. and 18.).

The second extended field is *MainTracker* and it contains the network location, in practice, the URI (specified in .torrent) of the CoreTracker. The location of the CoreTracker is needed in the context of the SubTracker registration to the CoreTracker, as shown in Figure 24 (9.). When the SubTracker is announced, it does not necessarily have any information about the location of the CoreTracker and the location is therefore delivered in the announce. Another option for the location information of the CoreTracker is to deliver .torrent to the SubTracker, though this is clearly, generally, a more unworkable solution.

Last but not least, the field extension in the tracker announce message is the SubTracker identification string, aka *SubTrackerId*. It contains the unique SubTracker identification string, specified in the same way as the *peer_id*. The SubTracker identification is used in the same phase of the download as the CoreTracker location in Figure 24. Once the SubTracker announces the CoreTracker (11.), the SubTracker delivers the identification information to the CoreTracker and the CoreTracker detects the source of the announce based on this. In this case, when the source of the announce is the SubTracker instead of a peer, the CoreTracker replies with an empty peer list (as in 12.) and does not add the SubTracker to the peer list.



*- **'SubT'**: a list of of dictionaries, where one dictionary responds to one piece of SubTracker connection information:*
+ 'ip': IP address of the SubTracker
+ 'port': TCP port the SubTracker is listening (usually 80 – HTTP)
*- **'TRG'**: a list of of dictionaries, where one dictionary responds to one piece of NNSP (TRG engine) connection information:*
+ 'ip': IP address of the NNSP
+ 'port': TCP port the NNSP is listening

Figure 28. Extended Tracker's announce response.

Figure 28 describes the required extensions to the tracker announce reply message. Once again, both the extended fields (*SubT* and *TRG*) are optional and stand-alone. The field on the SubTracker location information includes a list of dictionaries and dictionary responses to the location of one SubTracker. The dictionary includes two keys quoting the IP address (*ip*) and the TCP port (*port*) of the SubTracker. As the protocol is built over HTTP, the port number is 80 by default. The location of the SubTracker is required in the announce redirection in case the file exists in the connected domain, for example, in Figure 25. If the CoreTracker replies to the announce with the SubTracker location instead of a completed peer list (18.), the requesting peer contacts the SubTracker based on the information received (21.). The second extended field of the tracker announce response is a list of dictionaries, where a dictionary, once again, refers to a location of an entity. At this time, the entity is NNSP. The location of the NNSP is delivered constantly in the CoreTracker response to the peer, regardless of the need for the NNSP, for example, in Figure 23 (4.) and Figure 25 (18.). As is certain, the NNSP location is only required in the case of a peer with multiaccess capability. The NNSP is not delivered, for bandwidth saving reasons, to: a peer that is exiting the swarm (in Figure 23 and Figure 24 message number 6) or when the requester is a SubTracker (in Figure 24 message number 12).

The implemented extensions in the announce reply do not exclude the possibility of delivering several entity locations in the same response. If, for example, the requesting peer is connected to multiple domains in the download start-up phase, it could obtain information on both SubTrackers at the same time. At this time, signalling supports only one SubTracker location delivery; the second location is received through NNS. Multiple NNSPs could be delivered in the same announce reply message when NNSPs are used in a distributed manner as discussed in Section 6.1.2.

6.2.3. Backwards Compatibility

As mentioned briefly in Section 6.2.2, the implemented prototype, including extensions as in the signalling and in the message structure, should be as compatible with conventional BitTorrent as possible. The previous section also discussed interoperability between standard and extended message structures. Our experience, however, is based mainly on the BitTornado client and tracker implementations. Some tracker implementations, for example, could discard the whole announce message if it was not in standard form. The BitTorrent protocol is a loosely defined protocol and no authorities have made any official standard for it (the only specifications are in [29] and these can only be used as a basis). Discarding the message with rules that are too strict could therefore be an unworkable solution, especially when the implementation is distributed to a large community. In addition, a few tested implementations already add extra information to the announce and announce reply messages, and on that account we are confident that our message extension will not pose major compatibility problems.

Next, we consider the compatibility of different entities with conventional BitTorrent systems, one entity at a time. The client software for a single accessed peer is fully compatible with the standard BitTorrent system. In the swarm joining phase, the client only delivers the information about the domain to which it belongs, the tracker discards it and the download can begin from the swarm maintained by the tracker. The case is almost the same for the client software of peers with multiaccess capability: only the NNS functionality of the implementation needs to be configured.

The CoreTracker performs very well with elements of a standard BitTorrent system. It does not execute any announce redirections if it is not asked and does not affect to the BitTornado's tracker-to-tracker communication extension (load balancing and backup, described in Section 4.5.2). Moreover, the CoreTracker can maintain a peer list of a normal swarm and this can actually act as a backup swarm instead of a backup (or initial) seed. The only entity in the prototype system that has compatibility problems is the SubTracker. The available BitTornado's tracker-to-tracker communication extension is skipped completely in the SubTracker and employed for our purposes (SubTracker registration to the CoreTracker). As the SubTracker location is not specified in the BitTorrent metadata file, the SubTracker cannot be used in a conventional BitTorrent system. Moreover, some tracker implementations can produce problems for persons who realize systems like this particular prototype, as the tracker implementations can check the announce field from the announce message (shown in Figure 7) and validate the destination of the announce message. In practice, if the tracker is not specified in the announce field, the tracker discards the announce message. This was not the case with BitTornado however.

6.2.4. Towards Multiaccess NetInf

NetInf is an information-centric architecture for the future Internet (as described in Section 2.3) and should concentrate on the actual content instead of the place from which the content can be retrieved. In principle, this is what the BitTorrent does already. After it obtains the .torrent, the user does not necessarily care about the peer from which the blocks of the content are coming. Although BitTorrent is limited to very few scenarios and is not even close to solving all the issues of the current host-centric Internet, it is the right way. According to the NetInf name resolution (in Section 3.3), it typically follows the chain in which IOs are mapped to DOs and finally DOs are mapped to BOs. In general, the IO can respond to the .torrent (or the infohash in it) and the DO to the tracker response (including peers) in terms of BitTorrent. The actual content distribution can also be referred to as BOs. In NetInf terms, a node (leech) delivers the IO (infohash) to the NetInf resolution point (tracker). The NetInf resolution point responds with a DO (peer list), and the node performs the last binding to the BO (contacts other peers and starts downloading).

Our prototype uses a two-tier tracker hierarchy in which the NetInf architecture also divides the resolution into global and local level services. The CoreTracker in the prototype can act as the top-level name resolution service point, whereas the SubTrackers are local NetInf name resolution points. In the case of announce redirection, the top-level resolution point only redirects the bind to other resolution service points (cf. IO bind to another IO). The NNS is a completely new component of NetInf and the architecture of NetInf is not introduced at all.

7. EVALUATION

This chapter reports on the evaluation of our implemented content distribution prototype, which is described in Section 6.2. The primary purpose of the prototype was to prove the usefulness of a few ideas presented in NetInf. We therefore concentrate on clarifying the benefits of employing two network interfaces simultaneously and retrieving the content as locally as possible. Our evaluation is divided into two parts: static and pedestrian mobility scenarios. First, we examine the benefits of employing our multiaccess content distribution prototype over two network interfaces instead of using standard BitTorrent over a single network interface (in Section 7.1). Secondly we compare the multiaccess prototype to BitTorrent over MIPv6 in a pedestrian mobility scenario in which the user moves slowly from the coverage of the Wi-Fi access point to another (described in Section 7.2).

7.1. Single Access vs Multiaccess

In BitTorrent over a single network interface, our measurement scenario, seen on the left-hand side in Figure 29, consists of one tracker, three seeds and one leech (our downloader). All of the entities employ standard BitTornado applications for tracking peers, seeding and leeching. Every domain (including the Core Network) includes one seed, and the tracker locates in the Core Network. The leech is connected to a single domain using the IEEE 802.11g access point, which is presented in both access domains. We configured the access point in domain 1 to operate on Wi-Fi channel 13 and the access point in domain 2 on channel 3. Otherwise, the access points are identical in terms of technical specifications. The particular purpose of the configuration is to avoid interference caused by strongly overlapping Wi-Fi channels in the small area of our laboratory. The distance between the measurement laptop (SA leech; single accessed leech) and both access points are the same during the experiments.

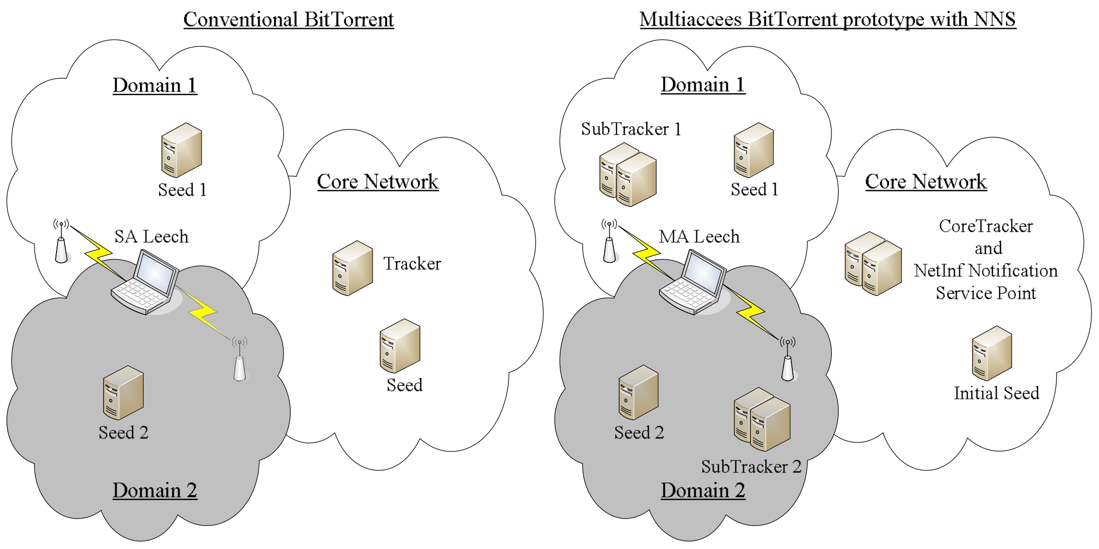


Figure 29. Comparison of experiment scenarios.

To minimize errors generated by equipment and wireless channel conditions, however, we run download measurement sets for both domains. We also introduce delays and small packet loss rates to the network using the Netem [63] network emulator. Delays and packet losses are added between all three domains, whereas the Core Network is the third one: between access domains 1 and 2 the added one-way delay is uniformly distributed in 50 ± 3 ms with a mean packet loss rate of 0.15% and between both of the access domains and the Core Network in 100 ± 5 ms with a packet loss rate of 0.3%. In practice, the response for a packet sent from domain 1 to domain 2 will take about 100 ms and the response to a message from domain 1 to the Core Network will take about 200 ms. The case is exactly the same for domain 2. With such long delays, the standard BitTorrent mechanisms (e.g., the choke algorithm) begin to direct the download to the closest peers (peers with the best capacity), and the advantage of the multiaccess prototype's peer localization plays a lesser role, but we still believe that there is a definite gain in our prototype. With smaller delays, cross-domain traffic would be the norm rather than the exception using standard BitTorrent.

In the multiaccess prototype over two network interfaces, as seen on the right-hand side of Figure 29, the network configuration is the same as in BitTorrent over a single interface including domains, access points, distances and delays. In this scenario, we also have three seeds, one in each domain as was the case with BitTorrent. Now we employ a two-tier tracker hierarchy (described in Section 6.2.1), however, in which the CoreTracker locates in the Core Network and one SubTracker in each access domain. The NNSP also locates in the Core Network. The multiaccess (MA) leech, for one, is connected to access points in both domains with identical interface cards.

One measurement set includes the following experiment ten times: the client announces the tracker, receives a peer list, downloads a file and exits the swarm. This measurement set is repeated for the SA leech connected to domain 1 and the SA leech connected to domain 2 employing standard BitTorrent, and also for the MA leech connected to both domains employing the prototype. Consequently, our peer again acts only as a leech in all the measurements. This time, we generated a 100 MB file instead of the Linux distribution in Section 6.1.1. We used the duration of the download as a performance metric, and we also measured intra- and inter-domain traffic.

The duration of each download and the median value of the download set are shown in Figure 30. Different download sets are represented with curves: blue indicates BitTorrent over domain 1, red BitTorrent over domain 2 and black the multiaccess prototype over both domains. As we see from the figure, the median download duration when the SA BitTorrent leech is connected to domain 1 is 42 seconds and when the SA BitTorrent leech is connected to domain 2 it is 39 seconds. The reduction in the median download duration is more than 7%, which is quite a big difference as we are using identical network equipment and configurations in both domains. It is also remarkable that the behaviour was the same for all of the ten downloads.

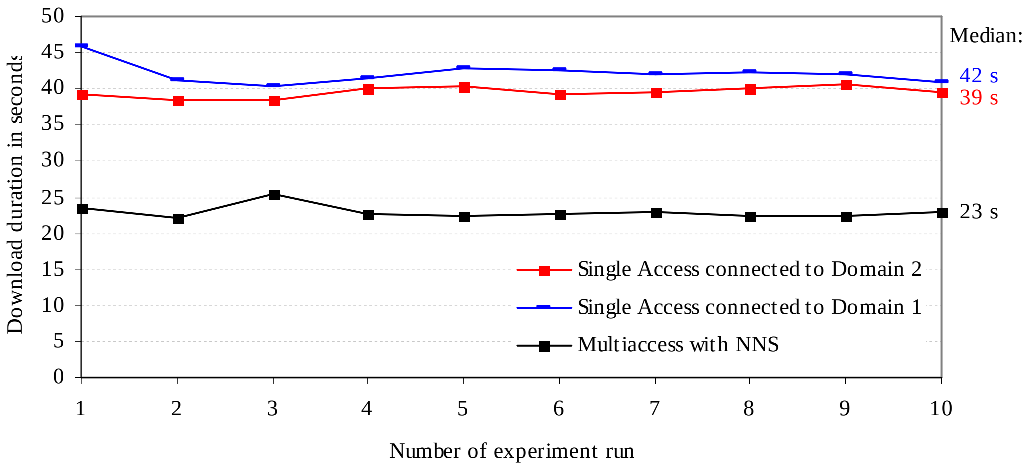


Figure 30. Download durations ordered according to the number of the run.

Our experiments were not performed in a completely restricted facility and several items of equipment could therefore have caused additional interference in the upper channel (13). The frequency difference between the channels (50 MHz) could also have caused extra, but minor, losses to the signal (higher frequency, bigger channel loss according to the channel models in [64]), and the attenuated signal could have affected the overall channel capacity. Nevertheless, most importantly by using the content distribution prototype over both interfaces, the median download duration was 23 seconds. The reduction in download time exceeded 41% when we compared the multiaccess prototype with single accessed standard BitTorrent. In principle, the download duration can be reduced by 50% when employing two interfaces instead of one, but in practice 41% is very close to that. Overall, the benefit of applying simultaneous multiaccess can be huge.

Results based on another performance metric, namely generated traffic, are presented in Table 4. In the figure, the traffic is categorized according to the leech connection point and source seed: intra-domain 1, intra-domain 2, Core-domain x and cross-domain traffic. In practice, when the leech is connected to domain 1: intra-domain 1 represents traffic between the leech and the seed 1 (cf. Figure 29), intra-domain 2 traffic should be zero (the leech is not part of domain 2), core-domain x (x is 1 in this particular case) is traffic between the leech and the initial seed and, finally, cross-domain is traffic between the leech and seed 2 (traffic crosses the domain boundary). As we see from Table 4, standard BitTorrent steers the download towards the best block source (local seed), derived from the added delays. When the leech retrieves the content from the domain to which it is connected, packets do not suffer from extra delays.

Table 4. Median portion of traffic according to the domain in the basic experiments

	Single access domain 1	Single access domain 2	Multiaccess with NNS
Intra-domain 1	86%	0%	47%
Intra-domain 2	0%	87%	53%
Core-domain 1/2	3.5%	3.0%	0.0037%
Cross-domain	11%	9.6%	0%

Overall, 86% and 87% of the traffic was generated inside the own domain. With relatively long delays, however, standard BitTorrent retrieves approximately 13% of the file from the Core Network or another domain and still generates significant cross-domain traffic. In contrast, when employing the prototype over two network interfaces, there is hardly any traffic between the local domain and other domains. In practice, all the traffic that crosses domain boundaries is announce and announce reply messages in download bootstrapping (described in Section 6.2.1), and it is negligible (0.0037%). The prototype with simultaneous multiaccess retrieves all the blocks locally from access domains, slightly favouring the faster domain 2.

7.2. Pedestrian Mobility Scenario

After showing the potential of employing simultaneous multiaccess with a multiaccess prototype, we demonstrate how the prototype can be applied to a slow mobility scenario and the benefits of the NNS in it. In experiments, we simulate the mobility of the leech with a simple Linux script. Again, we choose a standard BitTorrent system as a reference, but this time run it on top of MIPv6 in all entities (UMIP-DSMIPv6 provided by Nautilus6 [65]). The route optimization of MIPv6 is also enabled. Figure 31 represents the experiment scenarios used, with the upper half illustrating the pedestrian mobility scenario with BitTorrent over MIPv6, and the lower half illustrating the prototype with a leech with multiaccess capability. The experiment procedure and configurations are the same in both scenarios as in Section 7.1, but the availability of access points varies. In both scenarios in Figure 31, during the first 15 seconds, the multiaccess laptop (leech) is connected to domain 1 with one network interface. From time = 15 s to 25 s another interface is also activated and connected to domain 2. At time = 25 s, the first interface is turned off and only the second interface is in use until the download is complete.

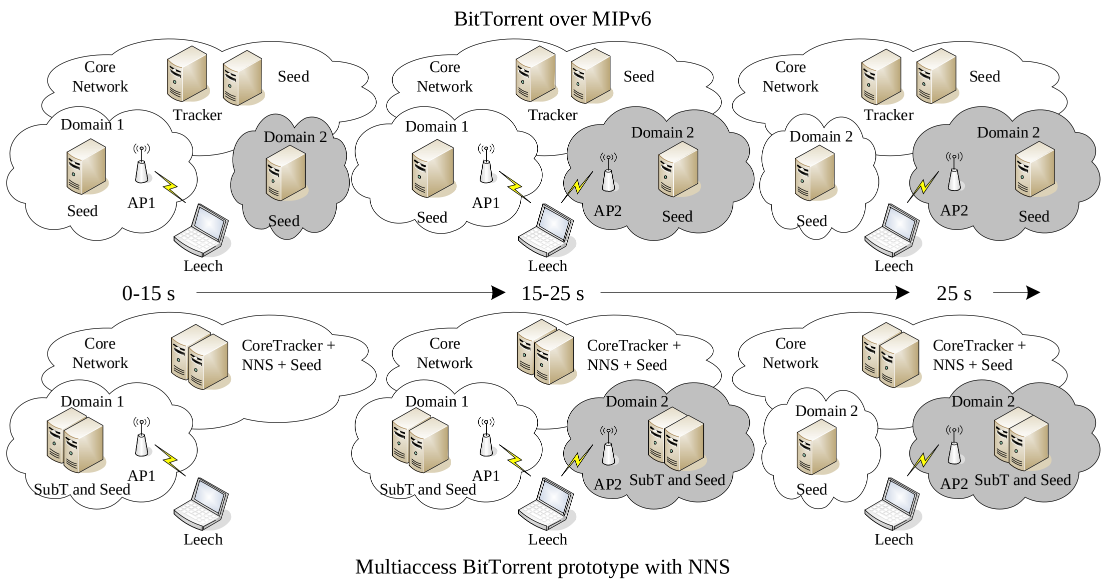


Figure 31. Comparative handover experiment scenarios.

In BitTorrent over MIPv6 (top of Figure 31), the leech retrieve pieces from the entire swarm (including seeds in all swarms) throughout the download. First, it only uses the interface connected to domain 1, but after the first 15 seconds, the second interface also become active (has IPv6 address and route to domain 2). The preference number of the first interface connected to domain 1 is higher, however, and the majority of the traffic therefore still goes through the connection to domain 1 (standard MIPv6 can only bind the HA to a single CoA). After 25 seconds from the beginning of the download, the first interface is deactivated and the MIP executes the handover to domain 2. The rest of the download is performed over the connection to domain 2. This kind of handover process can be called soft handover or "make-before-break" handover. In other words, the leech establishes a connection to a new domain before closing the connection to the earlier one.

In the multiaccess prototype with the same scenario (the bottom half of Figure 31), the leech tries to retrieve pieces from the local seeds, whichever the local seed(s) currently is/are. During the first 15 seconds, when the leech is connected to domain1, it exchanges information with the CoreTracker and SubTracker 1 and starts retrieving pieces from the seed belonging to domain 1. At time = 15 s, the leech activates the second interface. The prototype client detects the new connection to domain 2 at this time and asks the SubTracker information from the NNSP. Once the leech has the location of the SubTracker in domain 2, the leech can announce it and also start downloading pieces from the seed in domain 2 using the interface dedicated to that particular domain. Finally, at time = 25 s, the interface connected to domain 1 is deactivated and the rest of the file should be downloaded from domain 2. In this experiment, with the multiaccess prototype, we do not employ MIPv6, only IPv6.

Figure 32 represents the download duration of all downloads and median values of those sets. The red curve represents the download durations of the standard BitTorrent over MIPv6 and the portion of the black curve represents durations of the multiaccess content distribution prototype. In general, Figure 30 and Figure 32 show that the standard BitTorrent download slows down from 39/42 seconds (as discussed in the previous section) to a median of 54 seconds, due to handover execution. In per cent, the deceleration in download duration is more than 28%. On median, the download duration with the prototype is 38 seconds, which is still less than for standard BitTorrent without handover (39/42 seconds). It is significantly slower than with the static multiaccess scenario with the prototype (23 s) derived from the smaller portion of simultaneous multiaccess however. After comparing the handover scenarios, it may be said that the multiaccess content distribution prototype notably outperforms standard BitTorrent over MIPv6. On median, the download is 30% faster when employing the prototype. In other words, moving the BitTorrent user (over MIPv6) notifies the handover, with impaired user experience (longer download duration). With our prototype, the user experience is not lowered by handover execution.

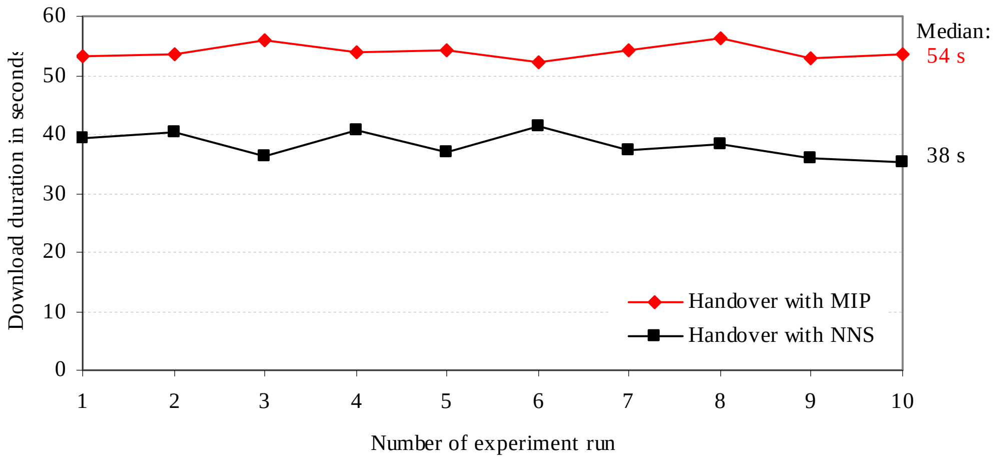


Figure 32. Download durations ordered according to the number of runs in HO scenarios.

To explain the major difference between the handover scenarios, Figure 33 represents the median throughput according to the time evaluation from the beginning of the download. In the bootstrapping phase of the download (first seconds in the figure), the cost of more complicated signalling in NNS can be seen clearly. The throughput in BitTorrent over MIP increases immediately after the first announce message (beginning of the download), whereas with NNS, it takes a while before it increases. The prototype clearly achieves a faster saturation throughput level because there is no need to locate "good" peers with the choke algorithm. The leech only has information about "good" local seeds (in standard BitTorrent, the leech has information about all three seeds). More importantly, after time = 15 s, the throughput of the multiaccess prototype with NNS is almost duplicated, resulting from the use of simultaneous multiaccess. In the case of BitTorrent over MIP, the throughput stays at the same level (no handover execution, no simultaneous multiaccess).

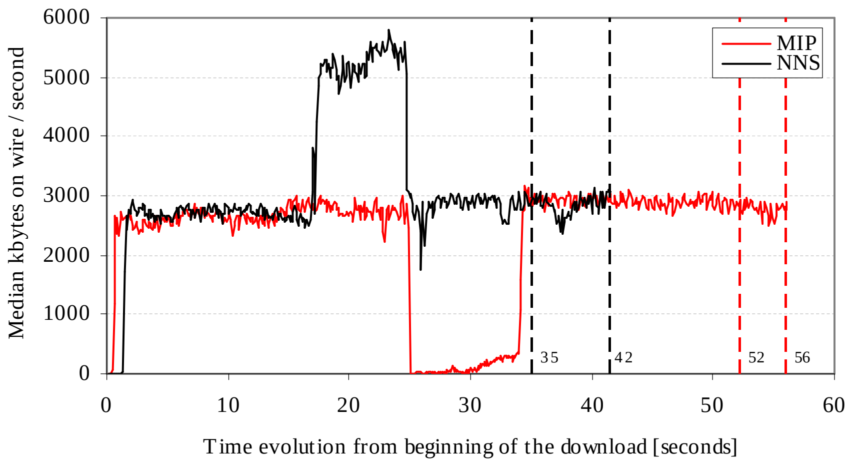


Figure 33. Median throughput in handover scenarios.

At time = 25 s, the first interface is deactivated and the throughput of the NSS therefore drops to the earlier level. The median throughput of the BitTorrent over MIP drops unexpectedly to almost zero and moreover stays at that level for almost 9 seconds. In practice, the MIPv6 handover latency (time when the connection is usable again) should be less than 150 ms (according to [66]), but it is probable that there is a HA inside the new network and that the propagation delay to the HA is therefore minimal. In our case, a two-way delay (message to the HA and back) is 200 ms and in this sense the underperforming throughput of the MIP is still surprising.

By analyzing in detail the behaviour of BitTorrent over the MIP with route optimization and the employed scenario, the explanation is certain. When the leech is connected to domain 1 (see Figure 31), most of the blocks come from the seed in domain 1, resulting from added delays, directly by using the leech's CoA. After the handover execution (at time 25 s), the MN (the leech) should bind the new CoA in domain 2 to the HA and to the CNs (seeds). Before the binding update to the CNs, all the traffic goes through the HA, which locates in the Core Network (this adds an extra delay to transmission). As binding updates are performed, downloading should, in principle, continue as before, but the leech still has pending requests aligned for the seed in domain 1, and responses to those take a while, again resulting from the added delay between domains. The leech over MIP recovers from the handover, however, and achieves the earlier level of throughput a little before the time of 35 seconds. Meanwhile, the fastest run with the prototype has finished the download (the first black vertical line) and the last one with the prototype takes place at 42 seconds (the second vertical black line). In addition, the first and the last download finishes with BitTorrent over MIPv6 are seen in the portion with the vertical red lines in the figure.

Table 5 illustrates the portion of intra- and cross-domain traffic, as does Table 4 in Section 7.1. Again, we can see the benefits of our prototype. In BitTorrent over MIP, the significant download portion of pieces is from the neighbour domain and the Core Network, whereas the multiaccess content distribution prototype retrieves all the pieces from inside its own current domain. In the handover scenario, BitTorrent over MIP even increases the portion obtained outside the domain (from 14% / 13% to 17%). The prototype essentially follows the same trend, however. The pieces are downloaded from their own domain and bootstrapping causes some, albeit minor, traffic between the current domain and the Core Network. Due to the slightly longer connection time, domain 1 has the biggest portion of traffic in both scenarios.

Table 5. Median portion of traffic according to the domain in handover experiments

	MIPv6	MA and NNS
Intra-domain 1	43 %	53 %
Intra-domain 2	40 %	47%
Core - domain 1/2	6.8%	0.0038%
Cross-domain	10%	0.000084%

One fact emerges from the table. The first time, in the case of the NSS prototype, there is little cross-domain traffic. After handover (at time 25 s), the leech still has information about the location of the seed in domain 1. Moreover, after the deactivation of the first interface, the connection to the domain 1 seed is just cut off without a safe shutdown. In the final stage of the download, the leech tries to contact to “old” seed, but the seed prevents the connection restarting due to the change in the leech’s IP address. In principle, the leech should establish the TCP connection again, but BitTornado does not (according to its own tests). This is the most likely reason for the minor cross-domain traffic.

7.3. Discussion

In general, our comparative measurements show the benefits of the multiaccess prototype compared with the standard BitTorrent system in both scenarios: static and pedestrian mobility. The most important finding is probably the usefulness of simultaneous multiaccess in the static as well as in the mobility scenario. Measurements show that the prototype can handle pedestrian mobility scenarios more efficiently than standard MIPv6 implementation, mainly due to the use of the two simultaneous accesses. This kind of multiaccess is a new aspect, which should also be taken into account in mobility solutions. The benefits of a reduction in inter-domain traffic are also remarkable. In the prototype, the content is obtained from the closest available sources. In this sense, valuable cross-domain or cross-ISP (in the current Internet) can be minimized. The results are even better than those achieved with other, orthogonal methods presented in, for example, Section 4.6.2.

As the multiaccess prototype is planned to be a general proof-of-concept prototype for a few specific scenarios defined by NetInf, several issues arise against the wider use of the current prototype. We divide the issues into two categories: general issues at the ideas level and more detailed technical issues, which are probably easier to clarify. The general issues are concentrated on first. To start with, the implemented prototype and BitTorrent are quite limited for file-sharing and content distribution purposes, especially for delivering files of a big size. For P2P IPTV systems, BitTorrent can be applied as a basis, as seen in [67], but for normal web browsing, for example, it hardly conforms. In general, if the content is static in some sense, it could be distributed over a BitTorrent-like system. Content distribution plays a major role in all traffic on the Internet, however, and therefore a prototype applied to such a scenario is still reasonable.

If the implemented system has thousands of files and machines, the CoreTracker will probably be the first point of failure in the sense of scalability. The load on the CoreTracker could be divided over several trackers, as in the multitracker extension described in Section 4.5.2, but multiple trackers will need more intelligent interaction. The DHT could also be applied somehow to this particular problem. Moreover, the division of swarms (several local swarms maintained by SubTrackers) can result in swarms that are too small which results in availability problems, but a DISM-like approach, for example, could be applied to provide some thresholds for division.

In the multiaccess prototype, simultaneous multiaccess can only be applied to local domains or some specific areas of the network. In practice, if the file does not have copies on the local domains to which the leech is connected, it can only employ a single interface as a default route when downloading pieces from the Core Network. The leech also does not have unambiguous identification from the Core Network point of view, and this causes problems in the NNSP and CoreTracker entities. In, for example, the pedestrian mobility scenario, the leech registers with the NNSP with an address belonging to domain 1. After completing the handover, the leech only has an address belonging to domain 2, and if communication between the

NNSP and the leech is needed, the NNSP does not recognize the leech. The NNS delivery chain is therefore broken. One solution to this could be the MIH, which also indicates a new access domain as the domain the leech is losing. This is discussed in more detailed later on in this section. Another approach to the Core Network identification problem is to expand the MIPv6 to handle simultaneous multiaccess in the context of local networks. In the real world, the multiaccess leech could use CoAs directly in communication within its own domains, while the home address could be used for communication with the Core Network. The leech is then seen as a single entity from every domain while moving across several domains.

When we consider the prototype in lower abstraction, several enhancement proposals arise. At first, as was defined in NetInf, the content should be available even without global connectivity. This is currently not the case with the prototype but can be implemented quite easily. When the leech connects to the CoreTracker and determines the lack of global connectivity, it can discover the SubTracker by broadcasting (as was done in the seed phase in Figure 24) and ask the peers directly from there. Peer registration to the SubTracker could be performed already at the leech phase, for example, after some portion of the pieces is retrieved. The access domain could also have dedicated seeds that automatically discover the need for a seed for a specific file and help the leech obtain the file in a similar way to that of Tribler [68]. Finally, we still have unimplemented portions such as MA peer seeding and delivery of several SubTracker locations in the same announce reply message, but these are only minor extensions to the prototype.

As mentioned earlier in this section, a change of domain causes identification problems in the Core Network in a pedestrian mobility handover scenario. A movable leech leaves behind peer information, especially to SubTracker entities in the past domain. Connections to peers should also be shut down carefully before leaving the domain completely. The MIH provides a suitable approach to these problems. In practice, when the leech leaves the domain, the MIH_SAP (interface between MIHU and MIHF) could provide an event to the client through the MIES by lowered signal, and the leech could notify the SubTracker of the departure, shut down connections to peers in the domain and unregister itself from the NNSP. Registration to the NNSP must be performed again, but now using a connection to the new domain. In this manner, the prototype could also be applied to larger scale tests. Moreover, all the functions in the current prototype by the NNSP could be moved to the MIHF, at least with minor extensions to the IEEE 802.21 standard. The need for two different information mediators is then minimized to one, decreasing the overall complexity of the system.

8. CONCLUSIONS

The main task of this thesis was to design and implement a multiaccess content distribution prototype by extending the current BitTorrent system and using Mobility Trigger Management. The prototype should outperform the current BitTorrent with mobility in an overlapping multiaccess scenario by employing several network interfaces, even at the same time, whenever possible. Our implemented prototype also retrieves the content as near the network as possible, thus decreasing to a minimum the cross-network traffic that is generated and improving the user experience by reducing the download duration. In this sense, we provide worthwhile evaluation information for a new information-centric networking paradigm and, especially, for the Network of Information [9].

First, this thesis described current state-of-art related to the implemented prototype. It began with the introduction of a protocol stack that is currently used on the Internet as a basis. Possible solutions for the future Internet were described by introducing Content-Centric Networking in which the current Internet is applied quite strongly together with a completely new Network of Information architecture, which also provides a basis for the prototype implementation. Information object resolution, today and in the future, was introduced briefly to obtain ideas for the later development of a peer discovery process. The main focus of the theory part of this thesis was given to peer-to-peer networks and, to be more specific, BitTorrent, which was described in detail, as we used BitTorrent as the basis of our prototype. Other existing peer-to-peer solutions were presented briefly: Freenet, FastTrack, Gnutella and eDonkey. Furthermore, mobility management on the current Internet was described at the end of the theory part. The operation of one existing mobility solution, namely mobile IP, as well as a couple of examples of cross-layer information exchange architectures such as Mobility Trigger Management, were described. Mobility Trigger Management was also employed in our implemented prototype.

The prototype design started with preliminary experiments with conventional BitTorrent with real content distributions on the Internet. We downloaded three different Linux distributions (Fedora 9, Ubuntu 9.04 and Ubuntu 9.10) with different access technologies such as Ethernet and Wi-Fi. The main consideration of the experiments was the peer discovery process. Our experiments proved the importance of early stage discoveries in, for example, the Fedora 9 download over Wi-Fi, on average almost 60% of the useful peers were discovered within the first minute. The portion of useful peers within the first minute was even bigger for other access methods, for example, Ethernet. In addition, we showed the importance of different discovery methods used in our client. The results showed that a remarkable portion, and in most cases even a major portion, of the content was retrieved from peers obtained from a global resolution point, alias tracker. Overall, these measurements clearly showed how we can affect the peer discovery process efficiently.

As we know the importance of early stage peer discoveries from the tracker in conventional BitTorrent, we decided to modify the operation of BitTorrent’s early stage peer discovery (in practice, tracker-obtained peers). We divided the operation of the tracker hierarchically into two levels, with local trackers maintaining a list of local content sources and the global tracker maintaining a list of local trackers. In practice, the peer that wants to download the content first contacts the global tracker. Whenever there is a local copy of the file, the global tracker returns the location of the local tracker and the peer asks for a list of peers from there. The peer retrieves the content directly from other peers. In this way, the content should be retrieved from the own network whenever it is possible to do so. In conventional BitTorrent, the global tracker returns the list of peers directly, as is also the case with the prototype, without a local copy of the content. The use of NetInf Notification Service (Mobility Trigger Management) makes it possible to indicate a new network connection by the peer. The peer delivers the information about the new available network to the global tracker through the NetInf Notification Service, and the global tracker returns the location of the local tracker in the same way. The peer asks for other peers from the local tracker and begins the download from the local peers in the new domain. In this manner, the peer can take advantage of several connections to different networks in parallel whenever overlapping wireless networks are available and even in mobile scenarios.

We compared the implemented content distribution prototype with conventional BitTorrent in two scenarios in our laboratory. First, we compared single accessed BitTorrent over Wi-Fi with the implemented prototype over two Wi-Fi access networks simultaneously. The results proved that with the use of our prototype, the reduction in median download duration was 41%, which, in practice, is very close to the theoretical maximum. In addition, we were able to decrease the portion of traffic between the different networks from 13% (conventional BitTorrent) to 0.0037%. Only inter-network traffic was generated by the initial stage peer discovery queries. In a pedestrian mobility scenario, we compared our prototype with BitTorrent over Mobile IP version 6. In this scenario, the peer was initially within coverage of the first of two wireless access networks. While downloading the content, the peer moved slowly from the first network to the coverage of another one. After the first 15 seconds, the access networks overlapped and our prototype was able to use multiaccess for ten seconds, whereas BitTorrent over Mobile IP used only one access network at a time. Rest of the content was retrieved through latter access network only. In this scenario, our prototype outperformed BitTorrent on median with 30%. As before, inter-network traffic was reduced significantly from 17% to 0.0039%.

As a conclusion, our prototype evaluation proves the significance of the employment of multiaccess whenever possible. In the static and also the mobile scenario, the improvement in the user experience was extensive. Our two-tier tracker hierarchy and improved BitTorrent signalling also gave very good results in terms of traffic locality.

9. REFERENCES

[1] Pentikousis K. & Rautio T. (2010) A Multiaccess Network of Information. In: IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), June 14-17, Montreal, QC, Canada. To appear.

[2] Rautio T., Mämmelä O. & Mäkelä J. (2010) A Multiaccess NetInf: a prototype and simulations. Submitted to: The 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities 2010 (TridentCom).

[3] ipoque (Accessed on 3rd of august 2009) Internet Study 2008/2009. URL: http://www.ipoque.com/news-and-events/news/ipoque-internet-study-2008_2009-finds-web-and-streaming-outgrows-p2p-traffic.html.

[4] Clark D. (1988) The Design Philosophy of the DARPA Internet Protocols. In: ACM Symposium on Communications Architectures and Protocols (SIGCOMM), August 16-18, Stanford, CA, USA.

[5] Jacobson V., Smetters D., Thorton J., Plass M., Briggs N. & Braynard R. (2009) Networking Named Content. In: 5th ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT), December 1-4, Rome, Italy.

[6] Dannewitz C., Pentikousis K., Rembarz R., Renault È., Strandberg O. & Ubillos J. (2008) Scenarios and Research Issues for a Network of Information. In: 4th International Mobile Multimedia Communications Conference (Mobimedia), July 7-9, Oulu, Finland.

[7] Seedorf J. & Burger E. (2009) Application-Layer Traffic Optimization (ALTO) Problem Statement (Work in procress), IETF Internet Draft, draft-ietf-alto-problem-statement-04.

[8] Mäkelä J., Pentikousis K. & Kyllönen V. (2009) Mobility Trigger Management: Implementation and Evaluation. In: International Journal of Communications, Network and System Sciences, vol. 2, no. 3, pp. 211-221.

[9] Ohlman B., Ahlgren B., Brunner M., D'Ambrosio M., Dannewitz C., Eriksson A., Grönvall B., Horne D., Marchisio M., Marsh I., Nechifor S., Pentikousis K., Randriamasy S., Rembarz R., Renault È., Strandberg O., Talaba P., Ubillos J., Vercellone V. & Zeghlache D. (2009) First NetInf architecture description, 4WARD project deliverable D-6.1, Work Package 6 (Accessed on 9th of February 2010). URL: http://www.4ward-project.eu/index.php?s=Deliverables.

[10] Forouzan B. (2001) Data Communications and Networking, 2nd edition. McGraw-Hill, New York, 944 p.

[11] Tanenbaum A. (2003) Computer networks, 4th edition. Prentice Hall PTR, Upper Saddle River, 891 p.

[12] Stallings W. (2004) Data and computer communications, 7th edition. Prentice Hall, Upper Saddle River, 847 p.

[13] Jacobson V., Smetters D., Briggs N., Plass M., Steward P., Thorton J. & Braynard R. (2009) VoCCN: Voice Over Content-Centric Networks. In: Re-Architecting the Internet (ReArch), December 1, Rome, Italy.

[14] The FP7 4WARD Project (Accessed on 12th of January 2010) Public web page. URL: http://www.4ward-project.eu/.

[15] Pentikousis K. (2009) Distributed Information Object Resolution. In: 8th International Conference on Networks (ICN), March 1-6, Cancun, Mexico.

[16] Mockapetris P. (1987) Domain Names – Concepts and Facilities. IEFT RCF 1034.

[17] Ahlgren B., D’Ambrosio M., Dannewitz C., Marchisio M., Marsh I., Ohlman B., Pentikousis K., Rembarz R., Strandberg O. & Vercellone V. (2008) Design Considerations for a Network of Information. In: Re-Architecting the Internet (ReArch), December 9, Madrid, Spain.

[18] D'Ambrosio M., Fasano P., Marchisio M., Vercellone V. & Ullio M. (2008) Providing data dissemination services in the future internet. In: IEEE World Telecommunications Congress (WTC'08), December 1-2, New Orleans, LA, USA.

[19] Eriksson A. & Ohlman B. (2007) Dynamic internetworking based on late locator construction. In: 10th IEEE Global Internet Symposium, May 11, Anchorage, AK, USA.

[20] Li T. (2008) Design Goals for Scalable Internet Routing Work (in process), IETF Internet Draft, draft-irtf-rrg-design-goals-01.

[21] Pan J., Paul S., Jain R. & Bowman M. (2008) MILSA: A Mobility and Multihoming Supporting Identifier Locator Split Architecture for Naming in the Next Generation Internet. In: IEEE Global Communications Conference (GLOBECOM), November 30-December 4, New Orleans, LA, USA.

[22] Pan J., Jain R., Paul S., Bowman M., Xu X. & Chen S. (2009) Enhanced MILSA Architecture for Naming, Addressing, Routing and Security Issues in the Next Generation Internet. In: IEEE International Conference on Communications (ICC), June 14-18, Dresden, Germany.

[23] Eng K. L., Crowcroft J., Pias M., Sharma R. & Lim S. (2005) A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys & Tutorials, Vol. 7, No. 2, pp.72-93.

[24] Aberer K. & Hauswirth M. (2002) An Overview on Peer-to-Peer Information Systems. In: 4th International Workshop on Distributed Data and Structures (WDAS), March 20-23, Paris, France.

[25] Wang C. & Li B. (2003) Peer-to-Peer Overlay Networks: A Survey. Technical report (Accessed on 9th of February 2010). URL: http://www.iwayan.info/Research/PeerToPeer/Papers_Research/P2P_Query/TR-P2P.pdf.

[26] Berkes J., University of Manitoba (2003) Decentralized Peer-to-Peer Network Architecture: Gnutella and Freenet. Academic paper (Accessed on 9th of February 2010). URL: http://www.sysdesign.ca/archive/berkes_gnutella_freenet.pdf.

[27] Liang J., Kumar R. & Ross K. (2004) The KaZaA Overlay: A Measurement Study. In: 19th IEEE Annual Computer Communications Workshop (CCW), October 17-20, Bonita Springs, Florida, USA.

[28] Tutschku K. (2004) A Measurement-Based Traffic Profile of the eDonkey Filesharing Service. In: 5th International workshop, Passive and Active Network Measurement (PAM), Lecture Notes in Computer Science, Vol. 3015 pp. 12-21, April 19-20, Antibes Juan-les-Pins, France.

[29] The BitTorrent Protocol Specification – BitTorrent Enhancement Proposal (Accessed on 4th of August 2009) Public web page. URL: http://www.bittorrent.org/beps/bep_0003.html.

[30] Choffnes D. & Bustamante F. (2008) Taming the torrent: A Practical Approach to Reducing Cross-ISP Traffic in Peer-to-Peer Systems. In: Conference of the Special Interest Group on Data Communications 2008 (SIGCOMM), August 17-22, Seattle, Washington, USA.

[31] Legout A., Urvoy-Keller G. & Michiardi P. (2006) Rarest First and Choke Algorithms Are Enough. In: International Congress of Mathematics 2006 (IMC), October 25-27, Rio de Janeiro, Brazil.

[32] Legout A., Urvoy-Keller G. & Michiardi P. (2005) Understanding BtiTorrent: An Experimental Perspective. Technical Report indria-00000156 version 3, November 9, INDRIA, Sophia Antipolis, France.

[33] Cohen B. (2003) Incentives Build Robustness in BitTorrent. In: 1st Workshop on Economics of Peer-to-Peer Systems, May 22, Berkeley CA, USA.

[34] SHA1 Secure Hash Algorithm – Version 1.0 (Accessed on 4th of August 2009) Public web page. URL: http://www.w3.org/PICS/DSig/SHA1_1_0.html.

[35] Neglia G., Reina G., Zhang H., Towsley D., Venkataramani A. & Danaher J. (2007) Availibility in BitTorrent Systems. In: 26th IEEE International

Conference in Computer Communications (INFOCOM), May 6-12, Anchorage, Alaska, USA.

[36] Multitracker Metadata Extension – BitTorrent Enhancement Proposal (Accessed on 4th of August 2009) Public web page. URL: http://www.bittorrent.org/beps/bep_0012.html.

[37] DHT Protocol – BitTorrent Enhancement Proposal (Accessed on 4th of August 2009) Public web page URL: http://www.bittorrent.org/beps/bep_0005.html.

[38] Izal M., Urvoy-Keller G., Biersack E.W., Felber P.A., Al Hamra A. & Gardes-Erice L. (2004) Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In: 5th International workshop, Passive and Active Network Measurement (PAM), Lecture Notes in Computer Science, Vol. 3015 pp. 1-11, April 19-20, Antibes Juan-les-Pins, France.

[39] Maymounkov P. & Mazieres D. (2002) Kademlia: A Peer-to-Peer Information Systems based on the XOR Metric. In: 1st International Workshop on Peer-to-Peer Systems (IPTPS), March 7-8, Cambridge, MA, USA.

[40] van den Berg A. (2009) A Complete Solution for Peer-to-Peer Widgets. Master's thesis. Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft, Netherlands.

[41] Myerson R. (1997) Game Theory: Analysis of Conflict, 1st edition. Harvard University Press, Cambridge, Massachusetts, 568 p.

[42] Pouwelse J., Garbacki P., Epema D. & Sips H. (2005) The Bittorrent P2P File-Sharing System: Measurement and Analysis. In: 4th International Workshop on Peer-to-Peer Systems (IPTPS), February 24-25, Ithaca, New York, USA.

[43] Dán G. & Carlsson N. (2009) Dynamic Swarm Management for Improved BitTorrent Performance. In: 8th International Workshop on Peer-to-Peer Systems (IPTPS), April 21, Boston, MA, USA.

[44] Hilt V., Rimac I., Tomsu M., Gurbani V. & Marocco E. (2008) A Survey on Research on the Application-Layer Traffic Optimization (ALTO) problem (Work in procress), IETF Internet Draft, draft-hilt-alto-survey-00.

[45] Xie H., Yang Y. R., Krishnamurthy A., Liu Y. & Silberschatz A. (2008) P4P: Provider Portal for Applications. In: ACM Special Interest Group on Data Communications (SIGCOMM), Seattle, Washington, USA.

[46] Aggarwal V., FeldMann A. & Scheideler C (2007) Can ISPs and P2P Cooperate for Improved Performance? In: ACM SIGCOMM Computer Communication Review, Vol. 37, Num. 3, July 2007.

[47] Luoto M. (2008) Mobility Triggering in a Multi-Access Environment. Master's thesis. University of Oulu, Department of Electrical and Information Engineering, Oulu, Finland.

[48] Gustafsson E. & Jonsson A. (2003) Always Best Connected. IEEE Wireless Communications, Vol. 10, pp. 49-55.

[49] Manner J. & Kojo M. (2004) Mobility Related Terminology. IEFT RFC 3753.

[50] Ylianttila M. (2005) Vertical Handoff and Mobility – System Architecture and Transistion Analysis. Acta Universitatis Ouluensis, Series C, Technica 220, University of Oulu, Oulu, Finland.

[51] Nikander P., Ylitalo J. & Wall J. (2003) Integrating Security, Mobility and Multi-homing in a HIP Way. In: 10th Annual Network and Distributed Systems Security Symposium (NDSS), February 6-7, San Diego, California, USA.

[52] Pierrel S., Jokela P. & Melen J. (2006) Simultaneous Multi-Access extension to the Host Identity Protocol. IEFT Internet-Draft, draft-pierrel-hip-sima-00.

[53] Johnson D., Perkins C. & Arkko J. (2004) Mobility Support in IPv6. IEFT RCF 3775.

[54] Perkins C. & Johnson D. (1996) Mobility support in IPv6. In: 2nd Annual International Conference on Mobile Computing and Networking (MOBICOM), Rye, New York, USA.

[55] Conta A. & Deering S. (1998) Generic Packet Tunneling in IPv6: Specification. IEFT RFC 2473.

[56] IEEE Std. 802.21-2008, IEEE Standard for Local and Metropolitan Area Networks – Part 21: Media Independent Handover Services.

[57] Piri E. & Pentikousis K. (2009) IEEE 802.21: Media Independent Handover Services. In: The Internet Protocol Journal, vol. 12, no. 2, pp. 7-27.

[58] Piri E. (2008) Implementation of Media Independent Handover for Wireless Broadband Multimedia Applications. Master's thesis. University of Oulu, Department of Electrical and Information Engineering, Oulu, Finland.

[59] Mäkelä J. & Pentikousis K. (2007) Trigger management mechanisms. In: International Symbosium on Wireless Pervasive Computing (ISWPC), February 5-7, San Juan, Puerto Rico.

[60] Aria2: The ultra fast download utility (Accessed on 15th of January 2010) Public web page. URL: http://aria2.sourceforge.net/.

[61] VTT Converging Networks Laboratory (Accessed on 15th of January 2010) Public web page. URL: http://cnl.willab.fi/.

[62] BitTornado (Accessed on 10th of August 2009) Public web page. URL: http://www.bittornado.com/.

[63] Netem (Accessed on 9th of February 2010) Public web page. URL: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[64] Goldsmith A. (2005) Wireless Communications, 1st edition. Cambridge University Press, New York, USA, 602 p.

[65] Nautilus6 working group (Accessed on 10th of February 2010) Public web page. URL: http://www.nautilus6.org/.

[66] Pinola J. & Pentikousis K. (2009) IPTV over WiMAX with MIPv6 handovers. In: IEEE 69th Vehicular Technology Conference (VTC2009-Spring), April 26-29, Barcelona, Spain.

[67] Hei X., Liang C., Liang J., Liu Y. & Ross K. (2007) A Measurement Study of a Large-Scale P2P IPTV System. IEEE Transactions on multimedia, Vol. 9, No. 8, pp.1672-1687.

[68] Pouwelse J., Garbacki P., Wang J., Bakker A., Yang J., Iosup A., Epema D., Reinders M., van Steen M. & Sips H. (2007) Tribler: a social-based peer-to-peer system. Concurrency and Computation: practise and experience, published online in Wiley InterScience.

10. APPENDICES

Appendix 1. Modified IPv6 patch for BitTornado BitTorrent client

Appendix 1. Modified IPv6 patch for BitTornado BitTorrent client

Different forums on the Internet argue that BitTornado is a totally IPv6-capable BitTorrent client. Practice shows, however, that the newest version (0.3.18), by default, is not. The NIIF (National Information Infrastructure Development Institute)/HUNGARNET (Hungarian Academic Research Network Association)'s IPv6 project[1] provides a solution to this. The solution disables the compact flag from the tracker announce message that makes it possible to return the peer list in compact mode. The compact mode affects issues with IPv6 addresses. The problem is addressed by removing the flag completely from the announce in the following way:

```
BitTornado/BT1/Rerequester.py:

Line 171:
remove        s += '&no_peer_id=1&compact=1'
insert        s += '&no_peer_id=1'

Line 188:
remove        s += '&no_peer_id=1&compact=1'
insert        s += '&no_peer_id=1'
```

This patch worked well with the NIIF/HUNGARNET IPv6 project's public BitTorrent tracker. After testing the modified client with other tracker implementations (for example, with BitTornado's tracker implementation), however, we figured out the major incongruity problem. BitTornado's tracker implementation needs information about the compact flag and we therefore modified the patch to the following format:

```
BitTornado/BT1/Rerequester.py:

Line 171:
remove        s += '&no_peer_id=1&compact=1'
insert        s += '&no_peer_id=1&compact=0'

Line 188:
remove        s += '&no_peer_id=1&compact=1'
insert        s += '&no_peer_id=1&compact=0'
```

From now on, the client is more fully IPv6 compatible than after the patch provided by the NIIF/HUNGARNET IPv6 project. While setting up a tracker to our own network, we also found problems with the BitTornado tracker implementation's IPv6 compatibility. By studying the code, we became aware of problems and fixed some bugs in the code. All these bugs were more or less related to the compact flag in the tracker implementation. After these fixes, we had a full BitTorrent system with IPv6 support. The changes are as follows:

```
BitTornado/BT1/track.py:

Line 690-691:
remove        bc[1][myid] = bc[0][myid]
              del bc[0][myid]
```

[1] NIIF/HUNGARNET IPv6 project. Public web page (Accessed 28.10.2009) URL: http://ipv6.niif.hu/.

insert

```
if bc[0] == {} and bc[1] == {}:
        pass
else:
        bc[1][myid] = bc[0][myid]
        del bc[0][myid]
```

Line 776:

remove

```
cache = self.cached.setdefault(infohash,[None,None,None])[return_type]
```

insert

```
cache = self.cached.setdefault(infohash,[None,None,None,None,None])
[return_type]
```

Line 928:

remove

```
if params('compact') and ipv4:
```

insert

```
if params('compact')==1 and ipv4==True:
```