Network Support for Intermittently Connected Mobile Nodes

Diploma Thesis

by

Simon Schütz

Matr.No. 0812224

submitted to

Department of Computer Science IV

Prof. Dr. Wolfgang Effelsberg

Faculty of Mathematics and Computer Science

University of Mannheim, Germany

June 13, 2004

Supervisor:

Prof. Dr. Wolfgang Effelsberg

Advisors:

Dr. Marcus Brunner (NEC)

Dr. Lars Eggert (NEC)

Abstract

Some nodes are only intermittently connected to the Internet. When they use different access networks, their IP addresses change, which disrupts ongoing connections. Even if IP addresses do not change, connections abort if disconnection periods are too long. Every disconnection period decreases the performance of ongoing TCP connections, because TCP does not resume data transfer immediately after connectivity is re-established. This thesis introduces the TCP Abort Timeout Option in conjunction with the Host Identity Protocol to solve the problem of connection abortion and introduces the TCP Retransmission Trigger to improve the performance of data transmissions. Detailed experiments with a prototype implementation of these mechanisms evaluate the effectiveness of the proposed solution. Measurements show that the proposed solution lets TCP tolerate arbitrary-length disconnections and IP addresses changes. Transfer time of a fixed amount of data could be reduced by a maximum of nearly 120 seconds and 43 seconds on average for one intermediate disconnection period.

Acknowledgements

There are a number of people to whom I would like to express my gratitude.

Special thanks go to my advisors at NEC Dr. Marcus Brunner and Dr. Lars Eggert and to Prof. Dr. Wolfgang Effelsberg from University of Mannheim for their support and guidance on the work for this thesis.

I also would like to thank Dr. Jürgen Quittek and Dr. Stefan Schmid for their feedback and helpful comments. Thanks to Joao Girao and Miquel Martin who supported me with their knowledge on Linux.

Last but not least, I would like to thank my parents for their ubiquitous support during the time of my studies.

Contents

List of Figures

List of Abbreviations

AC	Address Check
ACR	Address Check Reply
API	Application Programming Interface
ATO	Abort Timeout Option
CN	Correspondent Node
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DoS	Denial-of-Service
DTN	Delay-Tolerant Networking
DTNRG	DTN Research Group
ESP	Encapsulating Security Payload
FA	Foreign Agent
FQDN	Fully Qualified Domain Name
GPRS	General Packet Radio Services
GSM	Global System for Mobile communications
HA	Home Agent
HI	Host Identifier
HIP	Host Identity Protocol
HIP MM	HIP mobility and multi-homing
HIT	Host Identifier Tag

IANA	Internet Assigned Numbers Authority
IP	Internet Protocol
IPsec	IP Security Protocol
LAN	Local Area Network
LSI	Local Scope Identifier
MIP	Mobile IP
MN	Mobile Node
M-SCTP	Mobile SCTP
PDA	Personal Digital Assistant
PN	Peer Node
QoS	Quality of Service
SCTP	Stream Control Transmission Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
VIP	Virtual IP
WLAN	Wireless LAN

Chapter 1

Introduction

Within the last years, many users have started to use mobile devices such as notebooks, PDAs or mobile phones. They connect to the Internet using various access technologies, mainly GSM, UMTS or Wireless LAN. When the Internet was established, hosts were stationary and connected through wires links. Therefore, locations and network addresses of hosts were rather static and changed only very seldom. Link outages were mainly caused by hosts shutting down or broken cables. They lasted for rather long periods in the order of hours or days. In contrast, mobile devices can change their locations and access networks quite frequent. This implies frequent changes of network addresses, because most access network use a different address range. Leaving the coverage area of a wireless access point is a new source for link outages in the range of seconds, minutes or hours.

This chapter presents the basic scenario (see Section 1.1) that has to be supported by an appropriate network solution. Section 1.2 discusses networking problems that arise from this scenario. Section 1.3 summarizes the main contributions of this thesis. Finally, a short overview of the document structure is given is Section 1.4.

1.1 Basic Scenario

In the basic scenario of this thesis a mobile user is moving, for example by car or by train. The user connects to the Internet using wireless technologies such as GSM, UMTS or Wireless LAN. Physical connections are lost when entering tunnels or areas with bad coverage by base stations (see Figure 1.1). Access networks and internet service providers are switched on demand. Periods with high quality connections alternate with periods where there is no connection at all. In such scenarios, network services usually get interrupted.

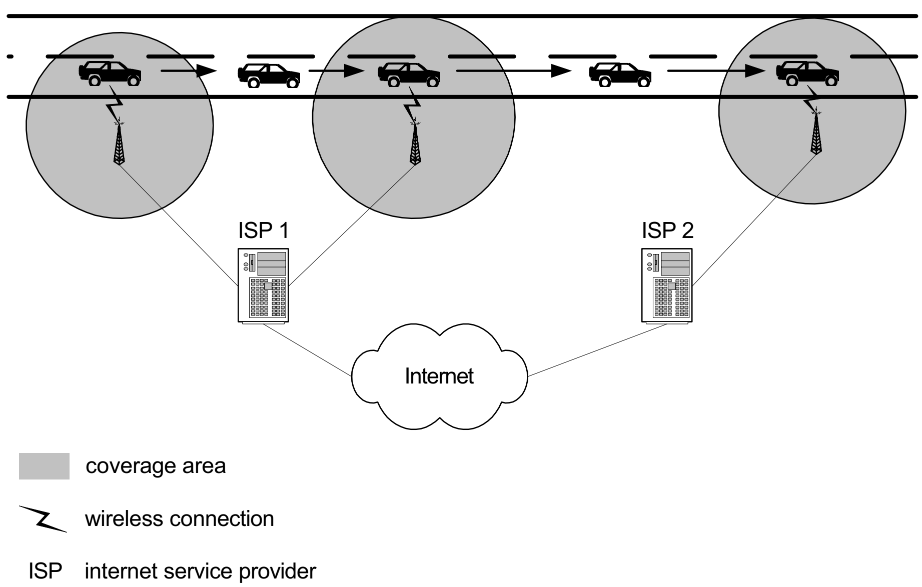


Figure 1.1: Example scenario

However, delay-tolerant applications like file download or web services do not rely on immediate or permanent responses by their communication peers. In this case, communication does not have to fulfil any real-time requirements. Guaranteed data delivery at any time in the future is more important than meeting Quality of Service parameters like maximal round-trip time or minimal bandwidth.

This type of applications can be supported by transparently reconnecting to the Internet and seamlessly continuing network services as soon as an access networks become available. For example, field managers might want to update databases or download files while travelling. In such a scenario, it is not important that the data transfer is finished within a few seconds, but it should be finished when he arrives at the next customer. In this case, each period of connection should automatically be used to send or receive data.

Remote connections often break if nodes move or temporarily loose network connectivity. Most of the time, these remote connections have to be re-established manually by providing username and password. If network support is able to keep remote connections open during disconnection periods, they could be continued immediately as soon as network connectivity is back again.

1.2 Problem Statement

Several networking problems arise from the scenario described above. Mobile nodes change geographic location and access networks. Short or long disconnection periods can occur at any time without prior warning. The transport protocols TCP and UDP as well as the Domain Name System (DNS) are not designed for such scenarios. In the following, the effects of these problems are discussed in more detail.

Mobility: When mobile nodes change location, they may change their access network, too. In this case, mobile nodes are assigned new IP addresses.

Communication end-points of TCP or UDP connections are identified by a triplet containing the IP address, port number and the transport protocol that is used. This information is set up when a connection is established and cannot change over the lifetime of a connection. Therefore, changing the IP address of a mobile node invalidates communication end-points. This means that each TCP or UDP connection breaks as soon as a mobile node changes its IP address. The problem arises from the dual role of IP addresses. They serve as node identifiers and locators at the same time. Ideally, identifiers should only names to uniquely identify nodes while locators should define the current location of nodes in the network.

Lookup services such as DNS experience problems with IP address changes as well. If a node should always be addressable by a static fully qualified domain name, DNS entries have to be updated on each IP address change. Therefore, frequently moving nodes imply a high load on DNS. Furthermore, DNS entries are usually cached for several minutes, hours or even days. Even with Dynamic DNS, cache times are in the order of several minutes. Consequently, propagation of new IP addresses might be delayed for several minutes as well.

In addition to these problems, new authentication mechanisms might be desirable to ensure that communication partners are still the same even if their IP address changes.

Disconnection: It is a challenging task to decide when a peer node should be treated as unreachable. With TCP, this decision is based on timeouts. Long timeouts waste resources like buffers, ports, memory and CPU time. However, dropping connections to early incurs bad service.

The TCP specification [Pos81] defines a *user timeout*. Connections abort when a TCP segment remains unacknowledged for a time longer than the *user timeout*. Currently, most TCP/IP stacks use a local default policy that applies to all connections on a host [Ste96, p. 299]. This implies that hosts are not able to adapt to current networking conditions of a particular connection.

In addition, adjusting local policies is not sufficient. If a host decides to extend its timeout values, connections might still be aborted by its peer hosts. Therefore, policies have to be propagated to or negotiated with the peer hosts.

TCP Performance: TCP is a reliable transfer protocol. Its reliability mechanisms are based on acknowledgements and timeouts. A segment, which is not acknowledged by the receiver before a *retransmission timeout* occurs, is treated as lost. Lost segments are retransmitted to ensure reliable delivery. Retransmission timeouts are generated by a timer called *retransmission timer.*

When TCP was developed, segment losses were mainly caused by congestion within an overloaded part of the network. Therefore, the retransmission timer was designed with an *exponential backoff* [Ste96, p. 299]. A segment is retransmitted several times if no acknowledgements are received. However, the value for the retransmission timeout is doubled after each retransmission attempt (see Figure 1.2). Therefore, retransmission timeouts grow exponentially. Most TCP/IP implementations use an upper limit of one to two minutes[1] for retransmission timeouts.

[1]Retransmission timeouts on a SuSE Linux 8.2 operating system are limited to 120 seconds by default.

Exponential backoff rapidly reduces the load on the network and adjusts to an assumed congestion within the network. In the case of physical disconnection, the retransmission timeout grows very soon to values of tens of seconds or up to minutes (compare Figure 1.2).

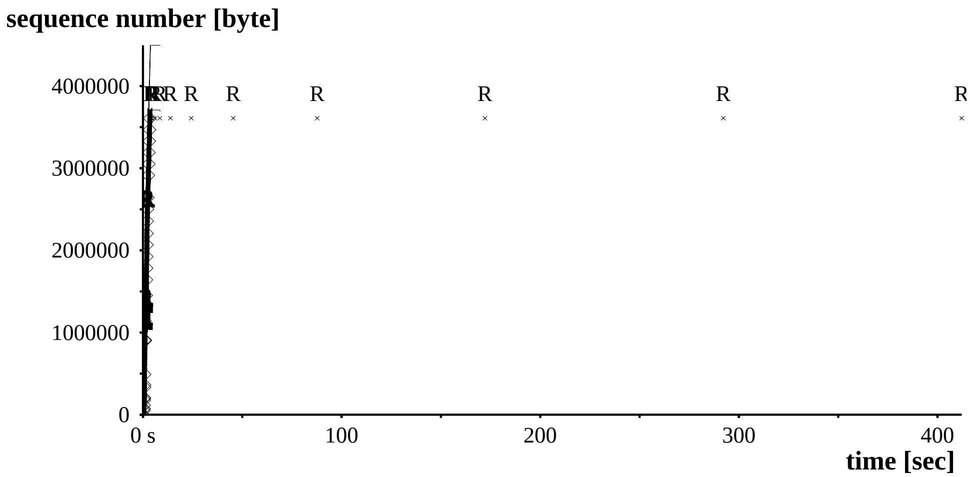


Figure 1.2: TCP retransmissions with exponential backoff

Exponential backoff leads to poor performance in scenarios where communicating hosts are temporarily disconnected. If a physical connection becomes available, TCP still waits until the next retransmission timeout expires. Retransmission timers are not influenced by events in lower network layers. Potential sending time is wasted if a physical connection is available but no segments are scheduled for transmission. In the extreme case, the retransmission timeout value might even be larger than the period of physical connections. TCP might not even try to transmit a segment, although network connectivity is available. Waiting for retransmission timeouts does not only lead to poor performance for bulk data transfer, but also incurs bad response times for interactive applications like ssh or web services.

1.3 Contributions

This thesis proposes a system that supports TCP-based communication for intermittently connected nodes and improves performance in such cases. The system is designed as an end-to-end solution and is therefore well supported by the current Internet infrastructure. It consists of three complementary parts. First, the Host Identity Protocol (HIP) serves as a mobility solution. It decouples connection end-points from IP addresses. End-hosts inform each other about IP address changes such that connections can be redirected after movements. HIP also includes a mechanism for host authentication. The second part is the TCP Abort Timeout Option that allows end-hosts to negotiate appropriate abort timeout values. Consequently, TCP can tolerate arbitrary-long disconnection periods if the negotiated timeout values are sufficiently large. The third part is a TCP Retransmission Trigger to improve TCP performance. It reschedules timeouts for outstanding retransmissions to occur immediately after re-establishment of network connectivity. This way, TCP connections resume data transfer earlier than with standard TCP.

The thesis also presents an experimental evaluation of the proposed system based on a prototype implementation. Results show that each part of the system fulfils its task. TCP connections tolerate IP address changes as well as long disconnection periods. The TCP Retransmission Trigger is able to reduce data transfer times by 43 seconds on the mean in the presence of one intermediate disconnection. In the best case, data transfer timer was even reduced by nearly 120 seconds. The maximum saving is limited by the upper limit for retransmission timeout values[2]. In the evaluation scenario, worst case bandwidth usage could be raised from 14% without TCP Retransmission Trigger to 69% with TCP Retransmission Trigger.

[2]This upper limit is 120 seconds on Linux systems.

1.4 Document Structure

Chapter 2 provides an overview of possible approaches to solve the problems of intermittently connected nodes. A few existing solutions are summarized in chapter 3. Chapter 4 presents the proposed system, which is experimentally evaluated in chapter 5. A summary and an outlook on future extensions is provided in chapter 6

Chapter 2

Design Space Analysis

This chapter provides an overview on different approaches that could support communication for intermittently connected mobile nodes. Two major aspects span the design space for such solutions: First, solutions can be introduced on different layers of the ISO/OSI reference model [Tan96, p. 28-35] (see Section 2.1). Second, different nodes throughout network infrastructure can provide these solutions (see Section 2.2). In the following, these two design options are discussed in more detail.

2.1 Network Layers

Support for mobility and disconnection as well as performance optimisations can be introduced on different layers of the network architecture. Sections 2.1.1 to 2.1.3 present various approaches.

2.1.1 Mobility Support

Node mobility can cause a change of IP addresses. This disrupts TCP and UDP connections, because bind to IP addresses of communicating hosts.

Therefore, mobility support must either decouple connection end-points from IP addresses or it has to provide a mechanism to dynamically update connection end-points. Link layer mechanisms operate only on connections to direct neighbours. Thus, they cannot provide mobility support if a node moves from one network to another. Consequently, mobility support can only be introduced on network layer or a layer above.

Network Layer

If mobility support is introduced in network layer, transport protocols like TCP and UDP still bind their connection end-points to IP addresses. TCP segments are encapsulated in IP packets that are sent to the IP address specified by transport layer. These packets have to be redirected if a mobile node changes its IP address. This can either be done by changing the destination address of the original IP packets or by IP-over-IP tunnelling. A tunnelling solution works as follows. On the sending side, the original IP packet is encapsulated in a new IP packet. The destination of the new IP packet is the foreign tunnel end-point, for example the current IP of a mobile node. On the receiving side, the original IP packet is retrieved by unpacking the received packet. *Mobile IP* ([Per02], [JPA03]) is an example for such a solution (see Section 3.1).

Indirection Layer between Network and Transport Layers

Another approach introduces a new layer in between network and transport layers. Its task is to separate the dual role of IP addresses. Currently, IP addresses identify a node and specify its location. An indirection layer allows the transport layer to bind connection end-points to identifiers that are independent of IP addresses. In this way, connection end-points remain valid even if nodes change their IP addresses. Transport-layer segments are passed to the indirection layer, which resolves the current IP address of the destination node and generates an IP packet addressed to the destination.

Several solutions that follow this approach have already been developed. Sections 3.2 and 3.3 present two examples for this approach, *Virtual IP* [YGD+01] and the *Host Identity Protocol* ([MN03], [MNJ03]).

Transport Layer

Transport-layer solutions bind connection end-points to IP addresses. Usually, this implies that IP addresses changes invalidate connection end-points. Thus, transport protocols must dynamically update connection end-points on demand during ongoing connections. If a mobile node changes its IP address, all of its peer nodes must update their connection end-points accordingly. This way, connection end-points remain valid and connections are not disrupted by IP address changes. *Mobile SCTP* [XKWM02] is an extension to the *Stream Control Transmission Protocol* [SRX+03] that follows this approach.

Session/Socket Layer

A last possibility to deal with IP address changes is to hide disruption of transport layer connections from the application layer. Therefore, new TCP connections have to be re-established automatically. This can be either be done on socket[1] or a new session layer on top of transport layer.

Mobile TCP Sockets [QYB97] and *Persistent Connections* [ZD95] modify socket implementations. Applications open TCP connections as with

[1]The socket layer is not an actual layer in the ISO/OSI reference model. It is the Application Programming Interface of the TCP implementation.

standard TCP. Thus, mobility is completely transparent to the application. If a TCP connection is disrupted due to changing IP addresses, the specialized implementations still accept I/O calls. In the background, a new TCP connection is established that can continue data transfer. This new TCP connection binds to the new IP addresses of the communicating nodes.

A second approach is to introduce a new session layer in between the transport and the application layer. Applications explicitly open a session instead of a standard TCP connection. This means that this solution is not completely transparent to applications, as they have to open a session instead of a TCP connection. However, mobility is still transparent once the session is established. Similar to a socket layer solution, the session layer transparently exchanges underlying TCP connections if one of the communicating nodes changes its IP address. *Migrate* [Sno03] is such a session-based solution.

2.1.2 Disconnection Support

In contrast to mobility support, disconnection support is difficult to solve in a layer lower than transport layer. This is due to the abort timeouts of TCP. Even if the network layer is able to delay packet transmission until a end-to-end connectivity is available, TCP aborts connections as it does not receive any acknowledgements. Thus, disconnection support has to be introduced in the transport layer or higher layers. This thesis concentrates on support for TCP connections. UDP connections do not provide reliable data transfer, such that packets could be discarded during periods of disconnection.

Transport Layer

Connection-oriented transport protocols, like TCP, abort connections after certain timeouts. Therefore, transport layer connections are disrupted if periods of disconnections become too long. Transport layer support can be provided by modifying those protocols to allow longer timeout values.

However, simply extending timeout values is not appropriate. It is desirable to introduce mechanisms that enable hosts to react to network conditions. Furthermore, a decision should not only be based on local information. In the best case, timeout values would be negotiated between communicating hosts.

Session/Socket Layer

Transport-layer solutions try to avoid disruption of ongoing connections. Another approach allows disruption of ongoing connections but hides these disruptions from applications by changing socket layer or introducing a session layer.

Disconnection support based on session- or socket-layer mechanisms is similar to mobility support at the same layers. If a connection is disrupted due to disconnection, I/O calls are still accepted. If necessary, data is buffered. As soon as a new end-to-end connection is possible, a new transport layer connection is established that continues data transfer. The three mobility solutions *Mobile TCP Socket* [QYB97], *Persistent Connection* [ZD95] and *Migrate* [Sno03] also integrate such a disconnection support.

2.1.3 Performance Optimisation

As mentioned in Section 1.2, TCP experiences severe performance problems in disconnecting environments, because retransmission timeouts grow to large values. Thus, lost segments are not retransmitted as soon as end-to-end connectivity becomes available after a disconnection. Restarting data transfer earlier can enhance performance.

One approach buffers a few packets at the link layer. This allows the link layer to retransmit these packets as soon as the disruption ends. If this link provides an end-to-end connection to the destination, retransmitting the packets fakes TCP retransmissions. The *Implicit Link-Up notification* [DW03] and the *Smart Link Layer* [SM03] serve as examples for link layer solutions (see Sections 3.8 and 3.9).

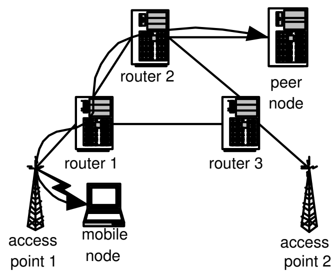
A similar approach can be applied at the network layer. In this case, the network layer buffers IP packets created by TCP connections. However, the network layer has more information about network topology. If a new link does not provide IP connectivity to a destination host, than retransmission of these IP packets is useless. Thus, the network layer only retransmits buffered packets over links that are likely to provide IP connectivity to the destination.

Finally, TCP's retransmissions themselves can be pre-scheduled. In this case, TCP segments are not buffered in lower layers, but new retransmission are generated immediately network connectivity is re-established. If TCP implementation offers a mechanism to pre-schedule retransmissions, this mechanism could be triggered when new IP connectivity becomes available. Thus, TCP retransmits lost segments earlier and data transfer is restarted earlier.

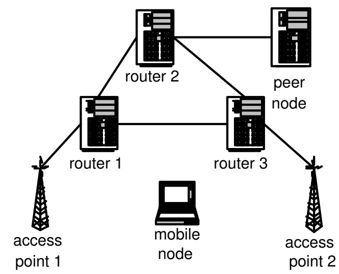
2.2 Network Nodes

Different nodes throughout the network can provide mobility or disconnection support. This section provides an overview three different possibilities. A comparison is given in Section 2.2.4.

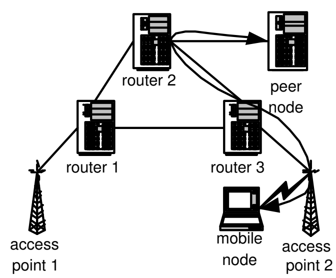
2.2.1 End-Host Approach



(a) Initial connection



(b) Disconnection



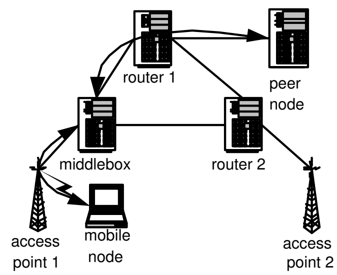
(c) New connection

Figure 2.1: End-Host approach

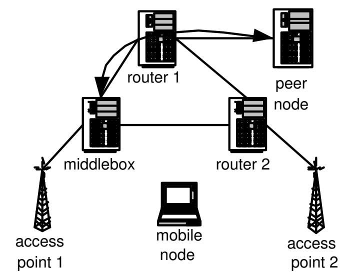
In an end-host approach, communicating nodes have to support mobility and disconnection themselves.

A mobile node (MN) and a peer node (PN) establish a TCP or UDP connection when an initial end-to-end connection is available (see Figure 2.1(a)). If MN looses network connection, no data can be exchanged between MN and PN (see Figure 2.1(b)). When a new network connection is available, MN and PN have to ensure that data transfer is continued(see Figure 2.1(c)). This can either be done by redirecting the old connection or by establishing a new one.

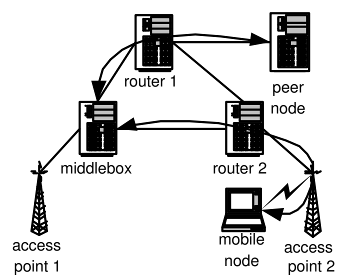
2.2.2 Middlebox Approach



(a) Initial connection



(b) Disconnection



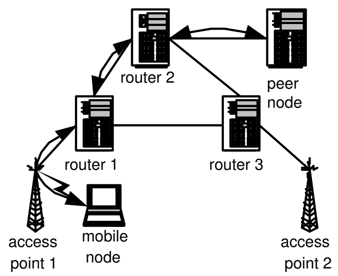
(c) New connection

Figure 2.2: Middlebox approach

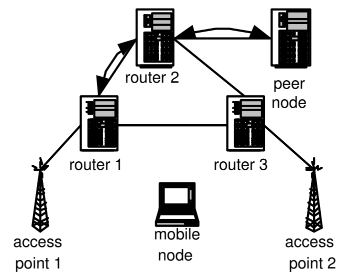
In a middlebox approach, connections are split into two parts (see Figure 2.2(a)). The first part connects the mobile node with the middlebox. The second part connects the middlebox with the peer node. The middlebox serves as a buffering proxy server.

This architecture completely hides mobility and disconnection from the peer node. If the mobile node looses network connection, the connection between middlebox and peer node is not affected (see Figure 2.2(b)). The connection between mobile node and middlebox has to be handled as in an end-host approach.

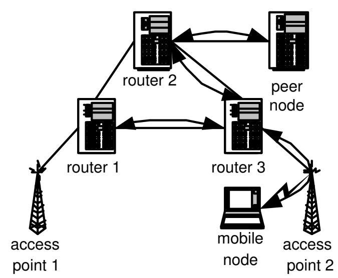
2.2.3 Store-and-Forward Approach



(a) Initial connection



(b) Disconnection



(c) New connection

Figure 2.3: Store-and-Forward approach

The Store-and-Forward approach is a continuation of the middlebox approach. It introduces many proxy-like servers to network infrastructure. Instead of using only one middlebox, data is forwarded from one proxy server to the next using short term connections (see Figure 2.3(a)). This way, data is pushed through the network until it finally arrives at the destination node.

If MN is disconnected, data that PN sends to MN is still pushed through the network (see Figure 2.3(b)). On reconnection, each proxy server that stores data destined for MN forwards this data towards MN (see Figure 2.3(c)). In reverse direction, MN transfers data to the first proxy server when an connection is available. Even if MN is disconnects after that, proxy servers forward data until it reaches PN.

2.2.4 Comparison

An end-host approach has two main advantages. First, it does not rely on any changes of network infrastructure. Second, it keeps TCP's reliable end-to-end semantics of TCP as a node only receives acknowledgements if its peer node has received data. The main disadvantage is that end-to-end communication is only possible when an end-to-end connection is available. If both communication nodes are mobile, it is possible that always at least one of them is disconnected. Therefore, these two nodes then cannot communicate.

This problem is solved by the middlebox approach. Data can be exchanged between a connected node and a middlebox. The communication partner can retrieve the data from and buffer a response on the middlebox as soon as it is connected. Thus, two nodes can communicate although no end-to-end connections exists at any time. A second advantage is that mobility and disconnection of mobile nodes is hidden from their peer nodes. Peer nodes only communicate with stationary middleboxes. However, a middlebox approach needs new elements in network infrastructure that have to be setup. A second disadvantage is, that middlebox solutions change TCP's reliable end-to-end semantics, because TCP connections are only established to the middlebox. Thus, a node assumes successful transfer to the destination as soon as data is received by the middlebox. However, it is still possible that data never reaches destination.

It is a trade-off decision whether a end-host or a middlebox approach should be preferred. This decision must be based on application and user requirements as well as on the possibilities to change or add network infrastructure elements.

The Store-and-Forward approach has many drawbacks. Reliable end-to-end transfer is not supported at all. End-hosts as well as many nodes throughout the whole network infrastructure have to be modified. Buffered data is spread over several proxy servers and mobile nodes have no information about these locations. Store-and-Forward implies a high overhead, because a separate connection has to be setup between each pair of proxy servers. The main advantage of a Store-and-Forward approach is its ability to deliver data if network infrastructure is unreliable. This means that many links throughout the infrastructure frequently disconnect. In this case, data can still be forward as soon as a connection between two proxy servers is available.

However, only links to end-hosts are subject to fail, such that the main advantage of a Store-and-Forward architecture does not come into account. Therefore, it is ill-suited to be used within this thesis.

Chapter 3

Related Work

This chapter gives an overview of existing solutions. Sections 3.1 to 3.3 present mobility solutions that are compatible with TCP. In Section 3.4, Mobile SCTP is presented as an example for transport layer mobility. The Mobile TCP Socket and Migrate solve both, mobility and disconnection problems (see Sections 3.5 and 3.6). Section 3.7 describes the Delay-Tolerant Networking architecture that uses an Store-and-Forward approach. Solutions for performance optimisations are shown in Sections 3.8 and 3.9.

3.1 Mobile IP

Mobile IP [Per02] provides a mechanism such that mobile nodes are always addressable with static IP address. This static IP address is called *home address*. Legacy nodes communicating with Mobile IP nodes do not have to support Mobile IP themselves.

The network that its home address belongs to is the mobile node's *home network*. All other networks are called *foreign networks*. The *visited network* is a foreign network that a mobile node is currently connected to.

Mobile IP defines several architectural entities. First, a *mobile node* (MN) is a node that is able to communicate using a constant IP address independent of its current point of attachment. Second, a *home agent* (HA) is a router in MN's home network. The HA tunnels incoming data towards the mobile node if it is outside the home network. Third, a *foreign agent* (FA) is a router in a visited network which extracts tunnelled data delivered received from HA and forwards it to MN.

(a) Mobile node in home network (b) Mobile node in foreign network

Figure 3.1: Communication with Mobile IPv4

Communication partners of a MN are referred to as correspondent node (CN). If a CN starts to communicate with MN, it always uses MN's home address.

If MN is in its home network, routing is performed like in usual IP networks. Packets are routed to the home network, pass through HA and finally arrive at MN (see Figure 3.1(a)).

If MN is connected to a foreign network, it is assigned a second IP address, the so-called *care-of address*. This care-of address is propagated to HA. If CN is communicating with MN, CN still sends IP packets to the home address of MN. However, HA intercepts these packets and tunnels them to FA. FA finally delivers them to MN. In reverse direction, two route are possible. With the first alternative, packets are passed from MN to FA, which tunnels them back to HA. From there, they are sent back to CN. The second alternative optimises the routing path. FA does not tunnel the packets to HA but sends them directly CN. However, these IP packets carry MN's home-address as the source address. Thus, they originate from a different network than the source address belongs to. Routers do not forward these packets if they use *network ingress filtering* [FS00] to prevent address spoofing.

3.2 Virtual IP

Virtual IP [YGD+01] introduces a new layer in between transport and network layer to support node mobility. Nodes are identified by fully qualified domain names (FQDNs). An FQDN is not directly mapped to an IP address but to a virtual IP (VIP). In a second step, VIPs are mapped to IP addresses. Like IP addresses, VIPs are 32 bit identifiers to provide compatibility with existing protocols. However, they are formed of Class E addresses[1] that are currently not used within the Internet. This way, IP addresses and VIP can be easily distinguished. VIPs are not globally valid. Instead, they are pair-wise negotiated identifiers between two communicating nodes.

If a VIP node wants to contact its peer node PN, it sends two DNS queries. The first query requests the current IP address of PN. The second query prepends the so-called VIP_MAGIC_NUMBER to the FQDN of PN. This returns a proposed VIP for PN that can be used to start a VIP negotiation. This creates a pair of locally unique VIPs that are used during later communication. The mapping from VIP to IP address is locally cached.

A mobile node MN updates its IP address in Dynamic DNS on each IP

[1] Class E addresses are in the range of 240.0.0.0 - 247.255.255.255

address change. Additionally, it sends an invalidation hint to its peer nodes. Thereupon, peer nodes start a new DNS query to retrieve the new IP address of MN and update the cached VIP to IP mapping accordingly. VIP to IP mappings are only valid for a certain lease time. It the lease expires, nodes also issue DNS queries to retrieve actual IP addresses of peer nodes.

All VIP traffic is encrypted using IPsec [KA98a]. The keys used to encrypt and decrypt packets are bound to VIPs instead of IP addresses. Thus, keys remain valid even if IP addresses change.

3.3 Host Identity Protocol

The Host Identity Protocol (HIP) introduces a new protocol layer in between the network and transport layers ([MN03],[MNJ03]). It defines a new namespace that decouples fully qualified domain names from IP addresses. The new namespace is called the Host Identity Namespace. Its primary goal is to provide a mechanism for host authentication. FQDNs are not directly mapped to IP addresses. They map to a Host Identity, which identifies a host. The HIP layer maps host identities to IP addresses, which define the physical location of nodes in the network. Transport layer connections bind their end-points to host identities and are therefore independent of IP addresses.

A Host Identity identifies the abstract entity that has to be addressed, a host for example. HIP defines three representations to refer to a specific Host Identity.

Host Identifier: The public key of an asymmetric key pair is used to address a Host Identity. This public key is the Host Identifier HI. A host holding the corresponding private key is the Host Identity addressed by a Host Identifier.

Host Identifier Tag: HIs can be of variable length. This is not ideal for use within a protocol. The Host Identifier Tag (HIT) is a hash value of the HI with a fixed length of 128 Bit. The HIT can be used for addressing within transport layer protocols.

Local Scope Identity: In IPv4 based protocols and Application Programming Interfaces (APIs), 128 Bit addressing schemes are not supported. The Local Scope Identity (LSI) is a 32 Bit compression of the HIT to be used in these situations. However, the LSI has only a local scope. Each host communicating with a specific Host Identity chooses its own LSI to represent the Host Identity.

Host Identities can either be public or anonymous. Public ones should be stored in a DNS so they can be resolved by all other hosts.

On a first connection, two hosts start the so-called *HIP base exchange*. The base exchange is a four way handshake that includes a Diffie-Hellman key exchange [MvOV97, p. 515] and host authentication. As the Host Identifiers are the public part of an asymmetric key pair, they can be used for authentication purposes.

The symmetric key that is generated during base exchange is used to set up a pair of IPsec security associations on communicating hosts. The security associations are bound to the Host Identifiers that performed the base exchange. All later traffic between the two Host Identifiers is transferred as ESP packets [KA98b] using these security associations.

Transport protocols bind their end-points to HIs. Within the HIP layer, these HIs are mapped to the actual IP address. This architecture is a first step to provide mobility support, as communication end-points remain unmodified on IP address changes.

Nikander [NA03] defines an extension to HIP that supports end-host mobility and multi-homing. A host accepts HIP packets of its peer from arbitrary IP source addresses. If the peer is assigned a new address, it announces this new address to all of its communication partners. Thus, those partners can send all further packets the new IP address of the peer. However, a host checks the availability of a peer at the new address before redirecting data traffic. This is to prevent denial-of-service (DoS) attacks against third parties.

HIP's mobility and multi-homing extension (HIP MM) provides a mechanism to support mobility while a HIP connection is already established. It does not solve the problem of retrieving an initial HI-IP mapping for contacting a new host. For this purpose, HI-IP mappings might be stored in a Dynamic DNS. However, nodes that are subject to frequent IP address changes would overload dynamic DNS. In addition, Dynamic DNS updates propagate quite slow, as DNS entries are cached for tens of seconds or minutes.

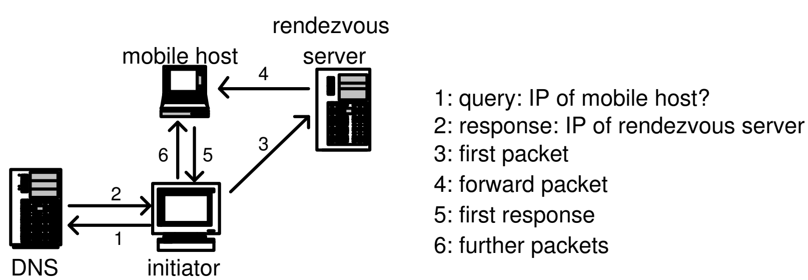


Figure 3.2: Initial communication setup using HIP

Therefore, HIP introduces a new infrastructure element called *rendezvous server*. Instead of registering its own IP address in Dynamic DNS, a mobile host registers the IP address of a dedicated rendezvous server. This rendezvous server is updated on each IP address change, such that it always stores a valid HI-IP mapping.

On connection setup, an initiator retrieves the IP address of a mobile host's rendezvous server from DNS (Steps 1 and 2 in Figure 3.2). It sends the first packet to the rendezvous server, which transparently forwards the packet to the current IP address of the mobile node (Steps 3 and 4). After the mobile host sent a first response packet (Step 5), the initiator can update its HI-IP mapping according to the source address of the response packet. All further packets are directly sent to the current IP address of the mobile node (Step 6)

3.4 Mobile SCTP

Mobile SCTP (M-SCTP) [XKWM02] is an extended version of the Stream Control Transmission Protocol (SCTP) [SXM+00]. It provides mobility support in when one node changes its IP address. Similar to TCP, SCTP is a connection-oriented transport protocol that offers error-free, non-duplicated and reliable data transfer. However, SCTP adds some new features. It supports multiple independent streams, each of them supporting in-order delivery. End-points of SCTP connections are not bound to only one IP address, but to a set of IP addresses. Data can be sent from each source address to each destination address.

M-SCTP uses the *Dynamic Address Reconfiguration* extension [SRX+03], that allows SCTP end-points to dynamically add or remove IP addresses of established connections. If a mobile node receives a new IP address, it adds this IP address to the connection's set of active addresses. When an old IP address gets invalid, it is removed from the set.

One major drawback is that this readdressing mechanism only works if an old IP addresses is still valid when a new one is assigned. Therefore, this mobility solution can only be applied if the coverage areas of access networks are overlapping.

The *Mobile SCTP* described in [RT03] and *Cellular SCTP* [AS03] are other solutions that use similar techniques to provide mobility support with SCTP.

3.5 Mobile TCP Socket

Qu, Yu and Brent [QYB97] propose Mobile TCP Sockets to support mobility. They define mobility as a combination of *portability* and *continuity*. Portability supports communication for mobile nodes while they stay in an arbitrary network. However, no communication service can be provided during movement. This kind of support as well as support for overcoming periods of (short) disconnection periods is provided by a continuity solution.

Portability is provided by an approach similar to Mobile IPv6 [JPA03]. A portable host has a home address belonging to its home network. When away from home, it is assigned a current address by a *Foreign Portable Support System* in visited networks. The current IP address is registered with the *Home Portable Support System*. A host wanting to contact the portable node sends a request to the Home Portable Support System using the portable node's home IP address. The Home Portable Support System replies by sending the portable host's current IP address. Having this information, the host can setup direct communication with the portable host.

Continuity is provided within socket API. Qu, Yu and Brent distinguish between a TCP association and a TCP connection. While a TCP connection is a communication channel bound to specific IP addresses, a TCP association is a reliable communication channel between two communication end-points, independent of IP addresses. With Mobile TCP Socket, sockets do not open a TCP connection but a TCP association. A TCP association is based on TCP connection, but if IP addresses change, underlying TCP connections have to be exchanged. To keep TCP semantics, the socket API accepts I/O calls even if there is currently no TCP connection at all. In this case, data is buffered and send out later when a new TCP connection can be established.

3.6 Migrate

Migrate is a session-based architecture to provide mobility support. A session layer on top of the transport layer is introduced. Applications open a session that remains valid even if underlying transport layer connections are aborted. On disconnection, Migrate explicitly suspends processes that communicate over a session. These processes are resumed when a new transport layer connection is available.

Migrate is designed to support arbitrary naming schemes and transport protocols. Therefore, an application is responsible to setup an initial transport layer connection. On session creation, it passes this connection to the session. In addition, an application has to pass the name lookup function.

Session layer is responsible to exchange transport layer connections as needed. If a node changes its network address, the session layer propagates this address to all peer nodes change by sending a *binding update*. Thereupon, peer nodes create new connections. All connections that are bound to the old address are aborted. Binding updates are encrypted with cryptographic keys, which are exchanged during session creation.

If both communicating nodes move during a period of disconnection, a binding update cannot be sent to the peer node because its new address is unknown. In this case, the session layer uses the lookup function provided by the application to retrieve the new address of its peer node. Using the retrieved address, it creates a new connection to the peer node and resumes the session.

3.7 Delay-Tolerant Networking

The *Delay-Tolerant Networking Research Group* (DTNRG) is working on an architecture to interconnect disparate networks. Each single network may experience different networking conditions and use different network protocols. Especially, communicating hosts can not rely on available end-to-end connections, continuous connectivity or a common naming scheme throughout all networks. Therefore, the *Delay-Tolerant Networking* (DTN) architecture defines the *Bundle Layer Protocol* [SB03].

The DTN architecture builds an overlay network to span over different existing networks. Routers at the borders of networks must support all network protocols of all adjacent networks. The Bundle Layer Protocol is used to bridge between these different networks.

Nodes implementing the Bundle Layer Protocol are called *DTN nodes.* Data that has to be delivered is grouped into bundles. These bundles are transferred to a DTN node within the same network using a local transport protocol. This DTN node uses the Bundle Layer Protocol to transfer the bundle to a DTN node in the network of the destination node. From there, it forwarded to the destination using a local transport protocol.

Bundle Layer Protocol operates in a Store-and-Forward manner. A bundle is locally stored until it can be forwarded to another DTN node that is nearer to the destination. However, each receiving DTN node has to ensure that the bundle will be delivered towards destination as soon as possible. This way, the Bundle Layer Protocol does not rely on an available end-to-end connection. Instead, it uses temporary connections between neighboured DTN nodes to forward data towards destination.

3.8 Implicit "Link-Up" Notification

Dawkins and Williams [DW03] propose a mechanism to notify TCP connections if new link layer connections are available. To accomplish this, they propose that subnetwork implementations on end-hosts store the last packet of each open TCP connection. If a new link is available, these packets are transmitted again. This *Implicit "Link-Up" Notification* fakes a TCP retransmission and may restart TCP data flow under certain conditions.

However, not all subnetwork systems are able to track TCP connections. In addition, TCP segments cannot be tracked they are sent encrypted in ESP packets. In these cases, a subnetwork system can only store the last packet per destination host, which limits the effectiveness of an Implicit "Link-Up" Notification.

3.9 Smart Link Layer

Scott and Mapp [SM03] propose a link layer approach to improve TCP performance in disconnecting environments. Similar to the Implicit "Link-

Up" Notification, the *Smart Link Layer* buffers one packet per TCP connection. However, this buffered packet is not necessarily the last one sent over a TCP connection. The Smart Link Layer selects packets such that they most likely restart an idle TCP connection. This decision is mainly based on the acknowledgement and sequence number fields of TCP segments. A second difference to the Implicit "Link-Up" Notification is that the Smart Link Layer may be introduced on any system in the network. It does not necessarily have to be an end-host.

The Smart Link Layer was tested by Scott and Mapp with two different modes, either *re-receiving* or *re-sending* packets. In re-receiving mode, hosts retransmit packets that they received over a disconnecting link before disconnection occurred. In re-sending mode, hosts retransmit packets over an upcoming link that they already transmitted before disconnection.

Scott and Mapp showed in experiments [SM03] that re-sending packets is more effective than re-receiving packets. In some cases, re-sending packets leads to a link utilisation of nearly 100%. However, the Smart Link Layer must read TCP segments to buffer appropriate packets. This is a layering violation. In addition, TCP segments can only be read if they are not encrypted with IPsec.

Chapter 4

System Design

The main objective for this thesis is a networking solution supporting intermittently connected mobile nodes that is backwards compatible with existing applications and services. It should not rely on major changes throughout the whole network infrastructure. Especially, TCP should be supported well because it is the most important transport protocol. As long as it can still be used, TCP-based applications need no or only minor modifications to benefit from the provided network support. Therefore, this thesis concentrates on supporting TCP-based communication for intermittently connected mobile nodes. To keep TCP semantics unchanged, the concept is designed as an end-to-end approach. Consequently, modifications only apply to end-hosts.

The Host Identity Protocol including its mobility and multi-Homing extension provides necessary mobility support (see Section 4.1). A TCP Abort Timeout Option offers the possibility to negotiate appropriate abort timeout values for TCP on a per-connection basis (see Section 4.2). Thus, TCP connections should be able to tolerate long periods of disconnection if abort timeout values are chosen sufficiently large. Finally, a TCP Retransmission Trigger introduces a mechanism to improve TCP performance in disconnecting networks (see Section 4.3).

4.1 Mobility Support

This thesis uses the Host Identity Protocol (see Section 3.3) with its mobility and multi-homing extension to provide mobility support. HIP meets the main objective for this thesis, which is support of TCP-based applications with minimal changes to network infrastructure and applications. Although HIP introduces rendezvous servers, this is only a minor change to network infrastructure. Rendezvous servers can be located anywhere in the network. In addition, relatively few of them are needed as each is able to serve a large amount of clients [MN03, p. 12]. In general, rendezvous servers are only needed if mobile nodes operate as servers. Only in this case the peer nodes have to retrieve the current IP address of a mobile node. If the mobile node acts as a client, it implicitly propagates its current IP address to the server on connection setup.

VIP (see Section 3.2) meets the main objective, too, but HIP is preferred over VIP for the following reasons. First, HIP is based on globally valid Host Identifiers, whereas VIP uses only locally valid identifiers that have to be negotiated on connection setup. Second, HIP introduces rendezvous servers for frequently moving nodes. VIP completely relies on Dynamic DNS. This might overload the Dynamic DNS and propagates updates only slow, as DNS entries are cached for tens of seconds or minutes.

All other mobility approaches presented in Chapter 3 violate the main objective and are therefore not used in this thesis. For example, Mobile IP needs a home agent in a nodes home network and a foreign agent in visited networks (see Section 3.1). Similar, the Mobile TCP Socket needs a Home Portable Support System and Foreign Portable Support Systems (see Section 3.5). Mobile SCTP does not support TCP at all, as it is a transport layer protocol itself (see Section 3.4).

4.1.1 HIP Mobility and Multi-Homing

The basic functionality of HIP is described in Section 3.3. In the following, HIP's mobility and multi-homing extension [NA03] is depicted in more detail.

HIP decouples transport layer protocols from IP addresses. Connection end-points are bound to the HI, not to the IP address of a peer. This way, IP addresses of communicating hosts may change, but the connection end-points remain the same. In the extreme case, hosts may even switch from IPv6 to IPv4 communication or vice-versa and keep transport layer connections open.

However, the mapping from HI to IP address has to be updated each time a host changes its IP address. During an ongoing connection, HI-IP mappings are stored locally on the communicating hosts. If a mobile host changes its IP, it informs each of its known peer hosts about the address change such that they can update their local HI-IP mappings. The packet flow for this so called HIP readdressing is shown in Figure 4.1.

HIP readdressing is a three-way handshake. First, the mobile host (MH) sends a readdressing (REA) packet to a peer host (PH) to announce its new address. In the next step, PH tries to verify whether MH is really reachable at the new address. Therefore, it sends an Address Check (AC) packet to the new IP address. On reception of an AC packet, MH has to confirm its new IP address by replying with an Address Check Reply (ACR) packet. PH does not send any data to the new IP address of MH before receiving the

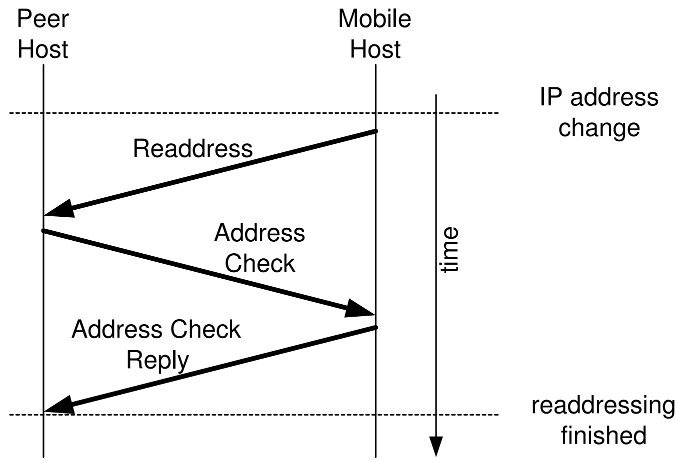


Figure 4.1: Packet flow for HIP readdressing

ACR packet[1].

The address check procedure helps to prevent distributed denial-of-service attacks. If no address check is performed, an IP address X might be attacked in the following way: A set of attackers request data transfers from any hosts. Then, all attackers spoof an IP address change to IP address X at the same time. Consequently X would be flooded, as all ongoing transfers are redirected to X. However, if the sending host first checks the availability of its peers at X, such attacks would not work.

HIP MM supports multi-homing in addition to mobility. A host may store several valid HI-IP mappings per peer host. Therefore, a host that changes an IP address does not only announce its new address but announces all its valid IP addresses. If an IP address becomes unreachable during an ongoing connection, a host can switch to one of the other known IP addresses of its peer.

[1]According to [NA03], an AC packet announces the SPI of a newly created SA. MH then has to create a new SA as well. When data arrives at PH using the new SPI, this is treated as the ACR packet.

4.1.2 Discussion

From all mobility solutions presented in Chapter 3, HIP MM turned out to fit best to the main objective of this thesis. It supports TCP-based communication without changes to applications and incurs only minor additions to network infrastructure. Mobility for ongoing connections is support end-to-end as each host updates its peers about IP address changes.

HIP already ships with an authentication mechanism and uses IPsec to encrypt data. This way, a security architecture is automatically included in this thesis' concept.

Although security is an important aspect, encrypting all data might be unnecessary for some applications. A music or video stream does not necessarily have to be transmitted in a secure manner. Encrypting such high volume data traffic implies high overhead and system load, which might even limit data rate on end-systems with poor CPU power.

4.2 Disconnection Support

In environments with periodic disconnections, TCP-based applications suffer from breaking connections if disconnection periods are too long. The corresponding abort timeouts are not defined by the TCP specification [Pos81]. Each host uses a default value that applies to all of its TCP connections.

This thesis tries to hide these periods of disconnection from applications. As it focuses on supporting TCP-based applications in an end-to-end approach, solution space is mainly reduced to two possibilities.

First, TCP is extended by a mechanism that allows the communicating hosts to keep TCP connections open, even if no physical connection

```
                    +---------+---------+
                    |  Kind=X | Length=6|
+---------+---------+---------+---------+
|             Abort Timeout             |
+---------+---------+---------+---------+
```

Figure 4.2: Format of TCP Abort Timeout Option

is available. The main advantage of this approach is that TCP state information is maintained and buffers are not flushed. Therefore, no additional mechanisms are needed to maintain TCP's reliable transfer semantics. In a second approach, TCP connections may be aborted, but they are transparently re-established if a new end-to-end connection is available (compare Section 3.5). This prevents resource exhaustion during disconnection because no TCP state has to be maintained. However, new mechanisms are needed to keep TCP semantics. I/O calls have to be accepted, although no actual TCP connection is available. Data that is not acknowledged before disconnection has to be retransmitted with a new TCP connection after reconnection. Consequently, the second approach is much more complex. Therefore, this thesis uses a solution following the first approach, the TCP Abort Timeout Option.

4.2.1 TCP Abort Timeout Option

The TCP Abort Timeout Option can be used at connection setup. End-hosts negotiate a duration for abort timeouts on a per-connection basis. This can be used to extend abort timeouts such that longer periods of disconnection do not cause an abortion of TCP connections.

Eggert [Egg04b] already proposed a first version of the TCP Abort Timeout Option. Figure 4.2 shows the format of the TCP Abort Timeout

Option. *Kind* is the option number that has to be assigned by IANA[2]. *Length* specifies an option size of 6 bytes. *Abort Timeout* is 4 byte value representing the proposed abort timeout in seconds.

Hosts that want to negotiate on abort timeouts have to include the TCP Abort Timeout Option with a proposed abort timeout value in their SYN segments during TCP connection setup. A host receiving a TCP segment with an Abort Timeout Option may either accept, shorten or reject the proposed abort timeout.

If it accepts or shortens a proposed abort timeout, then it includes the corresponding timeout value in a TCP Abort Timeout Option with the next segment it sends. From that time on, it has to use exactly this value as the abort timeout for the connection. The host that initiated the abort timeout negotiation has to use the value it receives.

Rejecting an Abort Timeout Option means that a host is not willing to negotiate on abort timeout values. In this case, a host must not include an Abort Timeout Option in the next segment. Both hosts must then use their default abort timeout values.

This version of the TCP Retransmission Trigger has certain drawbacks that could limit the effectiveness in disconnecting environments. First, the initiator of an abort timeout negotiation can only propose a value, but the receiver may arbitrarily shorten this value. If the initiator wants to increase abort timeouts over its default value, it proposes large abort timeout values. However, the peer might shorten this value such that it is even less than the default one. Second, abort timeout negotiation can only be started with SYN segments on connection setup. If networking conditions change during the established state of a TCP connection, abort timeout values cannot

[2] Internet Assigned Numbers Authority

```
                      +----------+----------+
                      |  Kind=X  | Length=6 |
+----------+----------+----------+----------+
| Min. Abort Timeout  | Max. Abort Timeout  |
+----------+----------+----------+----------+
```

Figure 4.3: Format of Extended TCP Abort Timeout Option

be adjusted. The next section proposes two extensions that could help to overcome these drawbacks.

4.2.2 Extensions

The extensions proposed in the following are based on the TCP Abort Timeout Option by Eggert [Egg04b].

Lower and Upper Limits for Abort Timeouts

As a first extension, the initiator does not only propose a single value. Instead, the TCP Abort Timeout Option specifies a desired value range. This is done by including a minimum and a maximum value in the option format. The new extended format is shown in Figure 4.3.

Kind and *Length* have the same meaning as in Section 4.2.1. *Min. Abort Timeout* and *Max. Abort Timeout* define a lower and upper limit for a desired abort timeout. To keep same option size as before, each timeout value is now a 2 Byte value. This limits the maximum abort timeout to 18.2 hours. If it turns out that longer abort timeouts are needed, abort timeouts could be specified in minutes instead of seconds. If using minutes, the 2 byte value limits maximum abort timeout to 45.5 days.

The *minimum abort timeout* defines a minimum value that has to be guaranteed, otherwise a negotiation is needless. This prevents negotiations ending up with undesirable short abort timeout values. If a mobile node knows that it experiences long periods of disconnection very often, then it might consider resetting all TCP connections that won't overcome those periods. Thus, it can save resources for connections, which are probably not aborted.

The *maximum abort timeout* is used to specify a maximum useful value. If a node that opens a connection is short on system resources, it might consider limiting the abort timeout. Another reason for putting an upper limit on an abort timeout could be an application that does not benefit from long abort timeouts. If, for example, a client wants to retrieve up-to-date information on stock exchange rates, there is no need to keep the connection open for hours. In this case, the client could specify a maximum value of a few minutes or even seconds.

A receiver of the extended option may either choose an arbitrary value in the proposed range or reject the negotiation. If it chooses an appropriate value, it includes an extended Abort Timeout Option with minimum and maximum value set to this value. After negotiation, both hosts must use this value for abort timeouts.

Negotiation in Established State

Abort timeout negotiation can only be started during connection setup. One reason is that most TCP implementations drop TCP segments that contain unknown option fields if the connection is in established state. However, if a negotiation has already been executed on connection setup, then both hosts support the TCP Abort Timeout Option. Consequently, negotiation during established state could be allowed in this case.

As described in Section 4.2.1, a host that rejects a negotiation does not include a TCP Abort Timeout Option in the next segment. Unfortunately, the missing option does not indicate whether the peer node rejects the abort timeout option or if it simply does not support it. As a consequence, no other negotiation can be initiated in established state.

This thesis proposes a second method to signal rejection. A host that rejects negotiation may include the TCP Abort Timeout Option in its next segment, but the abort timeout value has to be set to 0. This signals that it does not want to negotiate but still indicates that it supports the TCP Abort Timeout Option.

The same method can be used by the initiator. To signal a peer node that the TCP Abort Timeout Option is supported, even though no negotiation is initiated, the initiator includes an abort timeout value of 0.

With these two modifications, a host can inform its communication partner that negotiation is not wanted now, but may be desired later in the established state.

4.2.3 Discussion

The problem with TCP in disconnecting environments is connection abortion after a certain timeout. The TCP Abort Timeout Option is a mechanism to negotiate on this timeout. Consequently, it can help to prevent unnecessary abortion of TCP connections if hosts agree on sufficiently large timeout values.

However, it does not provide any policy mechanisms that decide which timeout values should be proposed or accepted. These decisions are still local to the hosts and may be based on history information, geographic positions, or availability of system resources.

Although primarily designed for extending abort timeouts, the TCP

Abort Timeout Option can also be used to save system resources. If two hosts use different abort timeout values, one aborts a connection earlier than the other. Thus, the latter wastes system resources by keeping TCP state for an aborted connection.

System resources can also be saved if applications request real-time data such as stock market information. In this case, TCP connections should be aborted as soon as possible in the event of disconnection, because the requested data will be outdated anyway by the time the node reconnects. As a result, hosts may agree on abort timeout values that are shorter than the default ones.

4.3 Performance Enhancements

If a node loses its physical connection, running TCP connections will encounter segment losses. Consequently, TCP stops sending further segments and waits for retransmission timeouts. Only after such a timeouts, it retransmits lost segments. Timeout values grow exponentially with each retransmission attempt. Therefore, retransmission timeout values rapidly reach an order of tens of seconds when a node is physically disconnected. If the node gets reconnected, TCP remains idle and waits until the next retransmission timeout occurs. The time that a physical connection is available, but not used by TCP is called *idle time* in this thesis (see Figure 4.4).

The idle time is one of TCP's major problems in disconnecting environments. When a physical connection is re-established, the transport protocol should immediately resume transmission of data instead of waiting for any timeouts.

The time a physical connection is available during the lifetime of a TCP connection is called the *net connect time* within this thesis (see Figure 4.4). In other words, it is the total connect time of a TCP connection less the physically disconnected time.

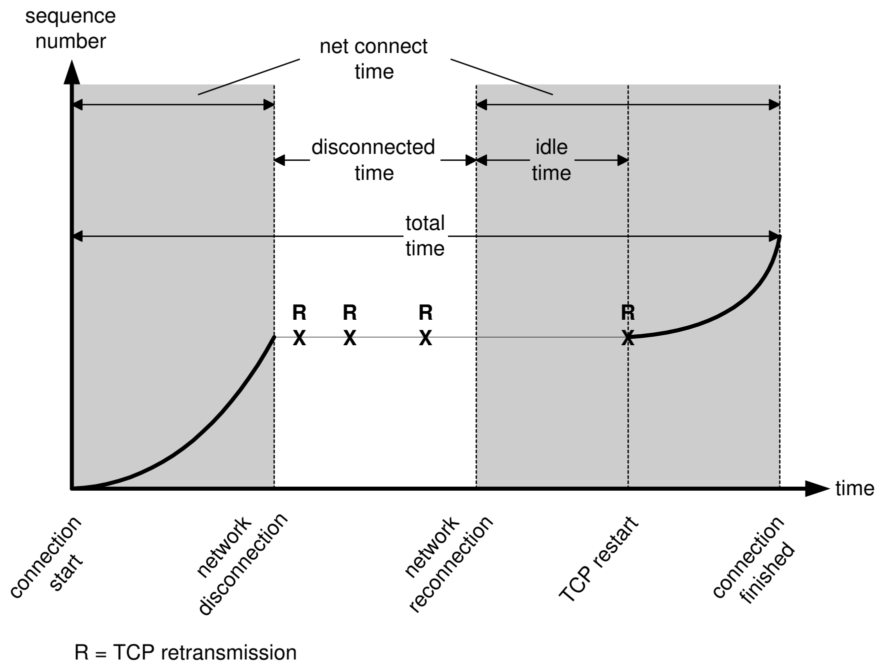


Figure 4.4: Total connect time, net connect time and idle time with one period of disconnection

As this thesis proposes an end-to-end solution, data transfer is only possible during net connect time. Thus, the goal is to optimise usage of net connect time.

This thesis concentrates on minimising TCP's idle times, because reducing idle times implicitly increases usage of the net connect time.

Different approaches to minimise idle times were already presented in Section 2.1.3. Link layer and network layer solutions have certain drawbacks. Buffering segments for a complete period of disconnection could violate

TCP's specification for a maximum segment lifetime[3] (see RFC793 [Pos81]). In addition, these solutions have to analyse the payload of IP packet to interpret the TCP header. First, this is a *layering violation*. Second, interpreting TCP headers is impossible if they are encrypted with IPsec. As HIP is used to provide mobility, TCP segments are transmitted as ESP packets, which can not be read by lower layers. Consequently, this thesis uses a transport layer approach called *TCP Retransmission Trigger* (see Section 4.3.1).

It should be noted here that TCP experiences additional problems if running over wireless connections. However, this thesis focuses on problems that result from disconnection and mobility, which also occur with wired links[4].

4.3.1 TCP Retransmission Trigger

The TCP Retransmission Trigger is a mechanism to reschedule outstanding retransmissions. This can be used to kick-start idle TCP connections immediately after a new link is available.

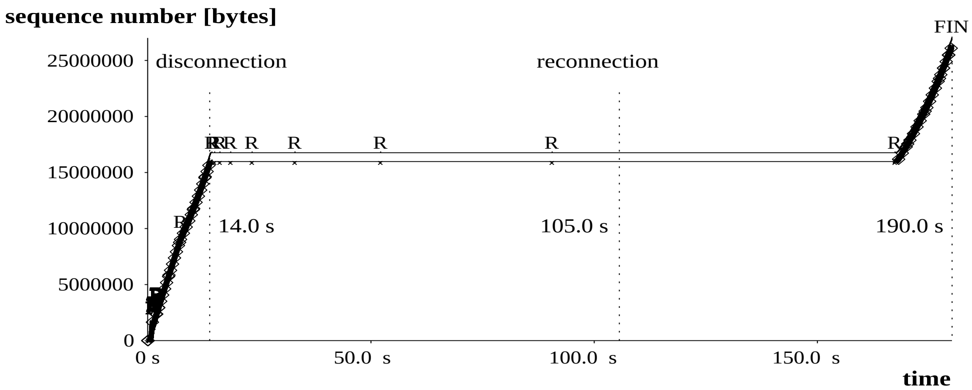
Assuming that retransmitted segments do not get lost, peer nodes will reply by sending acknowledgements for received data. As soon as the acknowledgements arrive, corresponding TCP connections can schedule new data for transmission and data transfer is continued.

Calling such a TCP Retransmission Trigger can reduce net connect time. If a physical connection becomes available, the TCP Retransmission Trigger

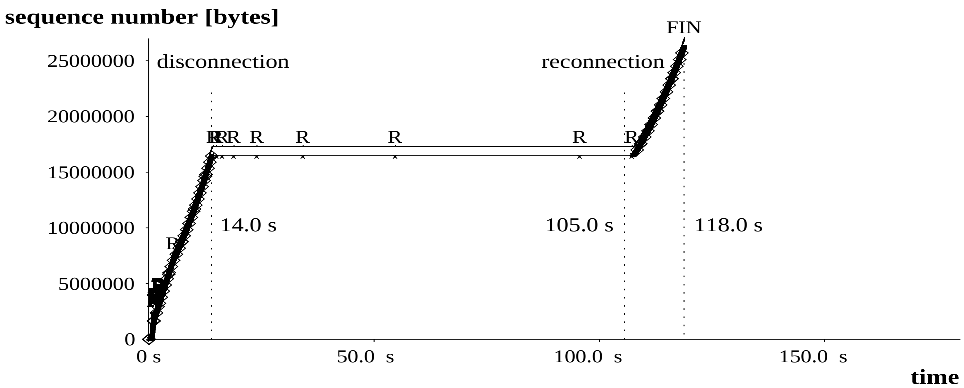
[3] TCP segments must be discarded if they do not reach their destination after 120 seconds.

[4] An employee might change from one office to another, connecting his notebook to Ethernet in both of them.

is used to reschedule planned retransmissions to be sent immediately after such an event.



(a) Standard TCP



(b) TCP with Retransmission Trigger

initial connect time = 14s, disconnected time = 91s

Figure 4.5: Time-Sequence graphs for reconnecting TCP streams

The effects of such a configuration are shown in Figure 4.5. Both TCP traces are captured on the same system with same timing for disconnection and reconnection. While Figure 4.5(a) shows the results of standard TCP, Figure 4.5(b) illustrates the effects of TCP using the retransmission trigger. As the retransmission trigger pre-schedules a retransmission, data transfer restarts than with standard TCP. Total and net connect time drop by the same amount of time as the retransmission is pre-scheduled. Standard TCP finishes data transfer after 190 seconds, but TCP with retransmission trigger already finishes after 118 seconds, thus saving 72 seconds of connect time.

The upper bound for retransmission timeouts on Linux systems is 120 seconds. Theoretically, a retransmission could be pre-scheduled by these 120 second. Therefore, calling a TCP Retransmission Trigger can reduce idle and net connect time by an amount of nearly 120 seconds, in the best case.

The evaluation presented in Chapter 5 was done using only a simple version of the TCP Retransmission Trigger. It tries to trigger all open TCP connections of a node. Rescheduling of retransmissions is performed for each TCP connection that meets two conditions. The first condition is that the TCP connection has to be bound to an IPv6 address or a HIT. It should be noted here that Host Identifier Tags are 128 bit numbers and look like IPv6 addresses. The second condition for rescheduling retransmissions is that a TCP connection has to be in a retransmitting state. This means, that there is unacknowledged data that was already retransmitted at least once.

A TCP Retransmission Trigger meeting these conditions is sufficient for the evaluation scenario. As only one TCP connection is active at a time, it does not have to precisely select the connections that benefit from being triggered. For use in realistic environments, a TCP Retransmission Trigger should be extended by some features described in Section 4.3.3.

4.3.2 Calling the TCP Retransmission Trigger

The TCP Retransmission Trigger offers a mechanism to reschedule TCP's planned retransmissions. Still, it has to be discussed which events should really trigger such retransmissions.

This thesis assumes that network interfaces are reconfigured when a new physical connection gets available. Consequently, the trigger should be called when an IP address is added or changed. This works fine if TCP is used on top of IP and the mobile node is always assigned the same IP. As TCP endpoints are bound to IP addresses, segments can be transmitted as soon as IP connectivity is restored.

Unfortunately, this is not true for TCP over HIP. If a mobile node is assigned a new IP address, HIP first has to update the HI-IP mapping on its peer node before bidirectional communication is possible. Even though it would be possible to send retransmissions to a peer node before readdressing is completed, the peer node might not be able to reply with an appropriate acknowledgement. Retransmissions are then treated as lost by the mobile node, and the TCP Retransmission Trigger fails to restart the data transfer.

Therefore, the TCP Retransmission Trigger cannot schedule retransmissions immediately after a new IP address is configured. Retransmissions have to be delayed by a time that is long enough such that HIP is able to complete its readdressing procedure first. Since this involves a three-way packet exchange, HIP readdressing can take arbitrary time. Its duration depends on the current round-trip time as well as on the system load of involved nodes. Triggering TCP connections after too short delays would fail, because HIP readdressing is not finished before retransmission. Delaying the trigger too long would waste valuable connection time, if HIP readdressing finishes early.

Consequently, the TCP Retransmission Trigger used in this thesis was directly coupled with the HIP readdressing mechanism. TCP connections are triggered immediately after HIP sends out an Address Check Reply packet, the last packet of HIP's readdressing mechanism. No delay is needed in this case because it is very unlikely that a retransmission arrives at the peer node before the Address Check reply packet, e.g. by using another route. When the retransmitted segment arrives at the peer node, HIP readdressing is already finished. Thus, the peer node is able to send an acknowledgement back to the new IP address of the mobile node. This triggers transmission of new data.

4.3.3 Trigger Extensions

Selecting appropriate TCP connections

A problem arises when a mobile node has more than one peer node at the same time. HIP readdressing must then be performed for each of them. In this case, all TCP connections are triggered multiple times, once for each peer node.

As the trigger is called from the HIP layer when an Address Check Reply packet is sent, the HIP layer could pass the HIT of the peer node to the TCP Retransmission Trigger. Instead of triggering all TCP connections, only those that are bound to this HIT should reschedule their retransmissions.

Triggering a peer node

So far, the trigger mechanism only works if a mobile node sends data. If it does not send data, no retransmissions will be scheduled. However, a mobile node's peer node might want to send data but waits for a retransmission timeout.

In conjunction with HIP, the trigger mechanism can also be used on the peer nodes. HIP readdressing is executed when a mobile node moves. Consequently, peer nodes are implicitly notified about reconnection of a mobile node. Peer nodes should call the TCP Retransmission Trigger each time a HIP readdressing is finished.

Without HIP, the TCP Retransmission Trigger has to be extended, because a peer node is not notified about reconnection of a mobile node. The TCP Retransmission Trigger should be extended to send at least four segments that acknowledge the last data received from the peer node. Thus, the peer node receives four acknowledgements for the same segment, such that the fourth one is a triple duplicate acknowledgment. This activates fast retransmit algorithm [Ste96, p. 312]. The four acknowledgements may either be sent piggy-bagged on data segments or as pure acknowledgements.

4.3.4 Trigger Characteristics

The TCP Retransmission Trigger does not change the basic congestion control algorithms of TCP. Therefore, TCP is not getting more aggressive to the network when the retransmission trigger is used.

Segment losses still cause the same recalculations for slow start thresholds and congestion windows. The only modification is rescheduling of single retransmissions on the mobile node in response to an address change. Even if the mobile node has no data to send, it injects only four acknowledgements that carry no data. Consequently, if the TCP Retransmission Trigger fails, then it transmits one additional maximum segment[5] in the worst case. If the TCP Retransmission succeeds, then it only sends data that has to be transferred anyway.

The TCP Retransmission Trigger modifies TCP to use more physically connected time, but it does not change its behaviour during an ongoing

[5]Inserting one retransmission corresponds to a maximum of one segment size. Pure Acknowledgements carry only header information for TCP and IP packets which is in the order of 40 Bytes (excluding option headers).

transfer. TCP implementations with a TCP Retransmission Trigger are fair to all other TCP implementations.

4.3.5 Trigger Implementation

The TCP Retransmission Trigger has been implemented for the Linux 2.4.20 Kernel as part of this work. The current implementation does not include the extensions of Section 4.3.3. The source code can be found in Appendix A.

Background

The Linux TCP stack holds a hash table `tcp_ehash`. For all TCP connections in established state, this table contains a pointer to a corresponding `sock` struct that stores information about the corresponding sockets. Among other information, a `sock` struct contains the connection type, protocol family and the addresses of both connection end-points. For TCP connections, it also stores pointers to a `tcp_opt` struct.

The `tcp_opt` struct holds status information about a TCP connection. Most important for the TCP Retransmission Trigger, it has an element called `retransmit_timer`. This is the timer that schedules retransmissions.

In Linux, TCP timers are implemented as `timer_list` structs. Among other elements, the `timer_list` struct contains a variable `expires` and a pointer to a function that has to be called when the timer expires. A timer expires when the system time passes the value defined by variable `expires`[6].

In particular, the `retransmit_timer` points to function `tcp_write_timer`, which is responsible for retransmitting lost TCP

[6] In Linux, timers do not operate on milliseconds but on *jiffies*. That is the number of clock ticks since last reboot. On an i386 platform clock frequency is 100 Hz by default.

segments. Thus, unacknowledged TCP segments are retransmitted as soon as the `retransmit_timer` expires.

The TCP stack offers a function `tcp_reset_xmit_timer`. It is used to change expiration times for TCP timers. A caller has to pass the sock struct of a TCP connection and the expiration time as a time offset from the current time.

Implementation

The TCP Retransmission Trigger is implemented in one function (see Appendix A):

```
void tcp_trigger_retransmit_timers(unsigned int when)
```

It steps through the hash table `tcp_ehash` to retrieve all appropriate TCP sockets. For these sockets, it overrides the `expires` value of the corresponding `retransmit_timer` by calls to `tcp_reset_xmit_timer`. The new timeout value is set to the current time plus the delay `when` that is requested by the caller of the TCP Retransmission Trigger.

Note, the TCP Retransmission Trigger does not override the retransmission timeout values for all TCP connections. It checks the address family of a TCP socket. Sockets that are not bound to IPv6 addresses are skipped[7]. The function also checks whether a TCP connection is in a retransmitting state. This means that there is at least one TCP segment, which is unacknowledged although it was already retransmitted before. In addition, retransmission timeout values are only overridden if the new value is lower than the old one. Therefore, retransmission timeouts are either pre-scheduled or kept unmodified, but they are never delayed.

[7]As already mentioned in Section 4.3.1, HITs are treated as IPv6 addresses.

4.3.6 Discussion

The TCP Retransmission Trigger provides a mechanism to restart TCP data transfer if a physical connection is re-established. Although it optimises TCP's usage of available bandwidth, it does not change its basic retransmission and congestion avoidance algorithms. Thus, it provides a performance enhancement without adding significant traffic to the network.

There are two main advantages over existing link-layer approaches. It operates directly on transport layer, such that lower layers do not have to read the TCP segments. This eliminates the overhead for reading these segments remains working even when segments are encrypted with IPsec.

4.4 Summary

The system is designed as an end-to-end approach. Thus, major changes are only applied to end-hosts. For the intermediate nodes, all traffic is seen as pure IP traffic. Consequently, the system provides a network solution that operates on current network infrastructure. It consists of the three elements HIP MM for mobility support, TCP Abort Timeout Option for disconnection support and TCP Retransmission Trigger to enhance TCP performance. Together, they form a complete solution to efficiently support TCP-based applications for intermittently connected nodes.

However, the three elements have an orthogonal design. This means that the elements do not affect each other. Each of them might be exchanged by another solution while keeping the whole system operational. For example, it should be possible to replace HIP with Mobile IP and leave disconnection support unchanged[8]. Similarly, the TCP Abort Timeout

[8]Only a call to the TCP Retransmission Trigger after an IP address change has to be

Option and TCP Retransmission Trigger can be exchanged without affecting the other elements.

Although the system is designed as an end-host approach, it is conceivable to combine it with a middlebox solution as described in Section 2.2. In this case, mobile node and middlebox could communicate using enhanced network support while peer node and middlebox use standard protocols.

implemented in MobileIP.

Chapter 5

Experimental Evaluation

This chapter provides an experimental evaluation of the system proposed in chapter 4. It is based on a simplified version of the scenario in Section 1.1. This experimental scenario is described in Section 5.1. Sections 5.2 and 5.3 present the test network and how it simulates the scenario. Three cases are evaluated. First is the baseline case, where only HIP's mobility and multi-homing extension is used (see Section 5.5). The second case adds the TCP Abort Timeout Option on top of HIP (see Section 5.6). The third case measures the complete system. HIP, TCP Abort Timeout Option and TCP Retransmission Trigger are enabled in this case (see Section 5.7).

5.1 Experimental Scenario

The experimental scenario is very similar to the base scenario described in Section 1.1. However, only one period of disconnection occurs in the evaluation scenario. Consequently, the experimental scenario has three distinct phases. In the *initial connection phase*, the mobile node (MN) is connected to the Internet via access point 1 (see Figure 5.1). MN opens a

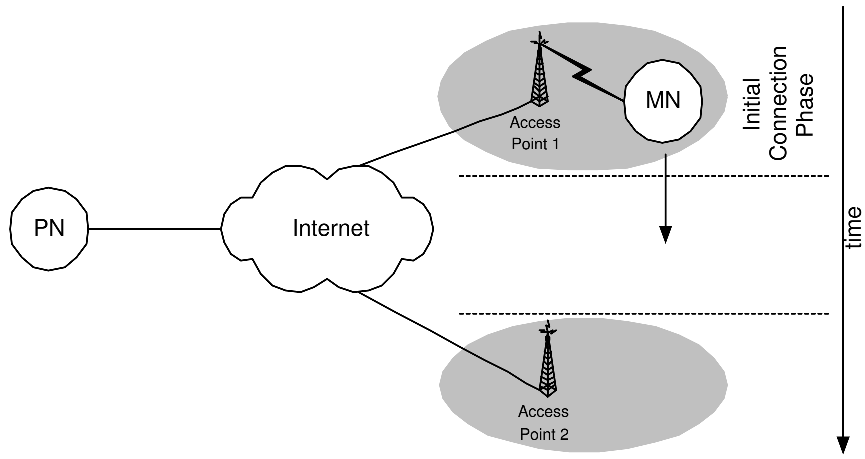


Figure 5.1: Initial connection phase

TCP connection to the peer node (PN) and starts a data transfer. After a certain amount of time, MN moves out of the coverage area of access point 1 and looses connectivity to the Internet, and consequently to PN.

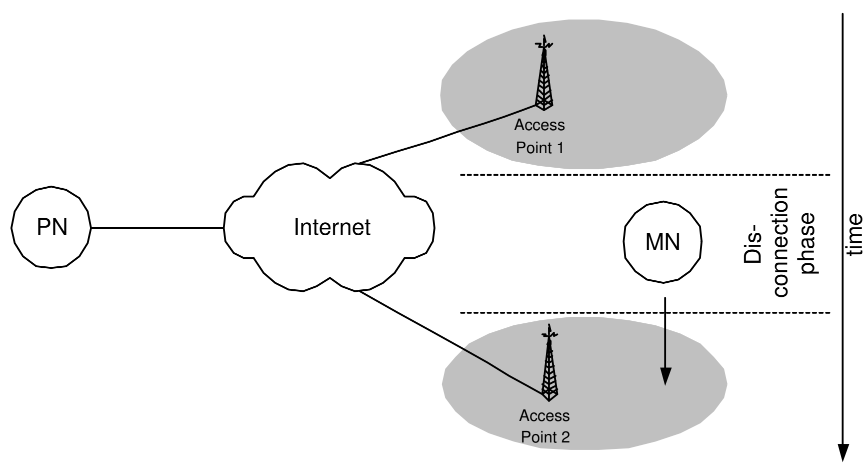


Figure 5.2: Disconnection phase

This is the beginning of the second phase, the *disconnection phase* (see Figure 5.2). During the disconnection phase, no network connectivity is available. MN keeps moving and reaches coverage area of access point 2 after some time. Network connectivity again becomes available starting the

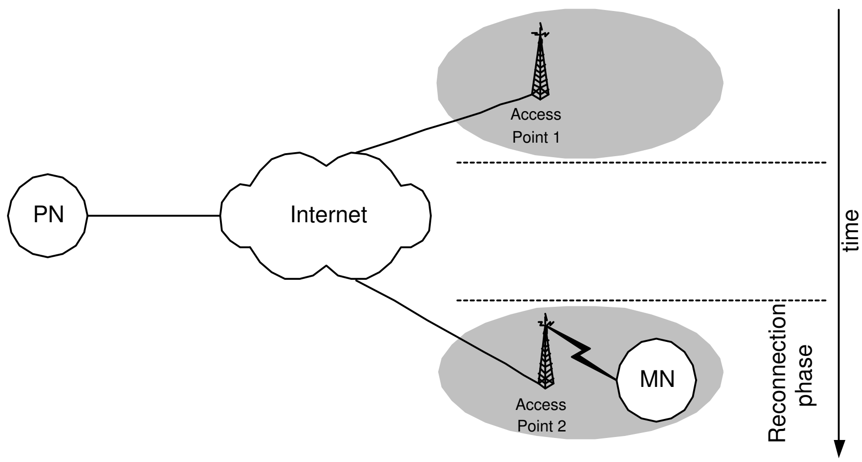


Figure 5.3: Reconnection phase

third and last phase, called the *reconnection phase*. In this phase, MN stops moving and remains in the coverage area of access point 2 until data transfer is complete.

Both access points provide the same quality of network connections. Bandwidth is limited to a maximum of 10 Mb/s. This is more than the Internet currently offers, but it is reasonable that a bandwidth of 10 Mb/s will be offered by next generation Internet in the future. One-way delay from MN to PN is set to about 50ms. This results in a round-trip time of 100ms. It should be noted here, that bandwidth and round-trip delay have only minor influence on the general effects inspected by this thesis. They only limit maximum throughput.

A network with these properties incurs a relatively high bandwidth-delay product. In order to eliminate the possibility of TCP socket buffers to limit throughput in such environments, they are set to 1 MB, which empirically proved to be sufficiently large during some preliminary testings.

5.2 Network Configuration

The test network consists of three identical PCs. Each is equipped with a Pentium IV 2.8 GHz and 256 MB DDR memory. Four network interfaces are available:

- 1x Intel 82557/8/9 Ethernet Pro 100, 100Mb/s
- 1x Intel 82540 Gigabit Ethernet Controller, 1Gb/s
- 2x Intel 1000MT Pro, 1Gb/s

The topology of the test network is shown in Figure 5.4. The PCs play the roles of mobile node MN, peer node PN and an intermediate router R.

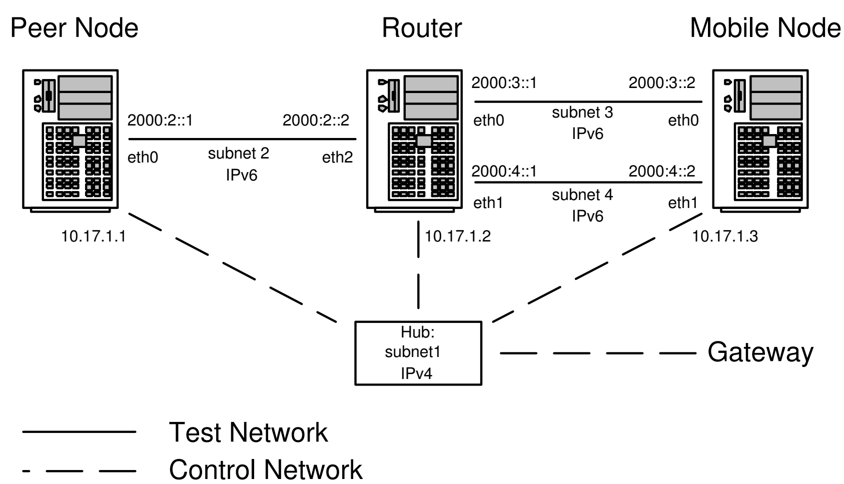


Figure 5.4: Network topology

PN connects to R via subnetwork 2. MN has two connections to R, either via subnetwork 3 or via subnetwork 4. These subnetworks are configured as IPv6 networks and use Gigabit Ethernet cards. Subnetwork 1 is an IPv4 control network, which is only used for executing commands on remote machines. The 100 Megabit Ethernet cards are used to connect to the control network.

5.3 Simulating the Scenario

The test network uses three stationary nodes with wired connections. Therefore, mobility and disconnection will be simulated.

As shown in Figure 5.4, two network interfaces connect MN to R. Configuration settings described in the following apply for MN only. R and PN remain unchanged during simulation. In initial connection phase, only interface eth0 is configured with an IPv6 address. The default route for IPv6 traffic is set to interface eth0 with R as a gateway. This provides an initial connection to PN. At the end of initial connection phase, the IPv6 address and as well as the default route is removed from eth0. This simulates a disconnection, because MN cannot communicate with R and PN anymore. After disconnection phase has elapsed, a reconnection at a different access point has to be simulated. Therefore, interface eth1 is configured. As eth1 belongs to a different subnetwork than eth0 (see Figure 5.4), it is assigned another IPv6 address than the one used in initial connection phase. Again, R is the gateway for IPv6 traffic, but the default route uses eth1 instead of eth0. Consequently, MN reappears in the network using a different link and subnetwork than before, just like it had moved to another location.

Interfaces and routes are configured manually with bash scripts. It would be easier and more realistic to use IPv6 auto-configuration, instead. In this case, interfaces eth0 and eth1 only have to be brought up or shut down at the appropriate times. Addresses and default routes would be set automatically.

Unfortunately, the HIP implementation used for the evaluation is still in experimental state and crashes in conjunction with IPv6 auto-configuration.

Mobility and disconnection are not the only features that have to be simulated. Network properties like bandwidth and round-trip time have to be simulated as well. Otherwise, the evaluation network provides a bandwidth of 1 Gb/s and a round-trip time of less than one millisecond, which is not comparable to Internet communication. Running a *Modular Click Router* on R allows modelling of these properties. The Modular Click Router [KMC+00] is a software router that is able to simulate various router behaviours like queuing, dropping and scheduling packets.

Wireless link characteristics are not simulated. As already mentioned in Section 4.3, this thesis focuses more on the impacts of disconnection and not on those of wireless links.

5.4 Parameters and Metrics

The evaluation scenario defines a mobile node that wants to transfer data to a peer node. Performance of the transfer can be measured in two ways. Either, the mobile node is sending data for a predefined duration and the amount of transferred data is measured. Or, the amount of data is predefined and performance is measured by the duration of the transfer. The second approach is nearer to a real use case. In general, a certain amount of data, like a file, has to be transferred. Another advantage is, that a broken TCP connection can easily be pointed out. If a TCP connection breaks before data transfer is complete, then the received amount of data is less than expected. Therefore, experiments are designed to transfer a certain amount of data and measure the duration of the transfer.

5.4.1 Parameters

The concept is evaluated for different movement characteristics of the mobile node. These are represented by two parameters, the duration of initial connection phase as well as the duration of disconnection phase. Therefore, a complete set of experiments for all possible parameter combinations has to be executed.

The duration of a transfer should be long enough, such that - for some settings of initial connection phase - TCP reaches a steady state before and after disconnection. Slow start effects are already finished after a few seconds, such that TCP should be in a steady state after 10 seconds at the latest. Thus, a data transfer should last at least 20 seconds, such that an initial connection phase of about 10 seconds splits up the data transfer into two halves of 10 seconds each. Preliminary tests have shown that 25 MB of data is an appropriate value, as it is transferred in about 20 to 22 seconds, if no disconnection occurs.

As data transfer lasts for about 20 to 22 seconds, initial connection phase was parameterized by values from 2 to 26 seconds. A value of 2 seconds corresponds to a scenario where the initial connection breaks very soon after TCP connection setup. On the other hand, an upper bound of 26 seconds should be long enough, such that data transfer is always completed before disconnection phase is entered. Consequently, a parameter range from 2 to 26 seconds will cover all possible scenarios. Choosing a resolution of 2 seconds, the parameter space for initial connection phase is

$$2, 4, 6, 8, 10, \ldots, 22, 24, 26.$$

Parameter space for disconnection space should start with 0 seconds of disconnection. This corresponds to a scenario, where the mobile node can immediately switch from one access point to another without experiencing disconnection in between. The upper limit and resolution can be arbitrarily chosen. It is only restricted by the time that is available for running experiments. This thesis uses durations of disconnection phase from 0 to 208 seconds with a resolution of 13 seconds. Therefore, parameter space for disconnection phase is

$$0, 13, 26, 39, ..., 195, 208.$$

One set of experiments consists of experiments for all possible parameter combination. This means that 13 parameter values for initial connection phase have to be combined with 17 parameter values for disconnection phase. Therefore, one set of experiments consists of $13 * 17 = 221$ single experiments.

Each experiment underlies certain deviations. Several runs for each experiment are executed to get more reliable results. The number of runs is only restricted by time. For the evaluation, 10 runs are executed which takes about 5 days to complete.

5.4.2 Metrics

As the concept is an end-to-end approach, the total connect time for a data transfer is not an appropriate performance metric to evaluate the concept. During a disconnection phase, data cannot be transferred end-to-end because no connected path is available. Consequently, a data transfer takes at least as long as the disconnection phase. Therefore, duration of disconnection phase should not be included in the performance metric.

Net connect time meets this condition. As already defined in Section 4.3, net connect time is calculated by subtracting the disconnected time from the total connect time of a data transfer.

$$\textit{net connect time} = \textit{total connect time} - \textit{disconnected time}$$

Net connect time only measures that part of a data transfer, that can be influenced by an end-to-end approach. System performance is better if net connect time is shorter. The best end-to-end solution leads to a minimal net connect time.

As already mentioned in Section 5.4.1, each experiment is repeated 10 times. The results presented later, show the median that is calculated over these 10 runs. The median is preferred over the better known mean because results showed to be heavy tailed.

Deviations are usually reflected by the standard deviation if mean values are used to aggregate data. The inter-quartile gap is the corresponding metric for median values. Narrow inter-quartile gaps indicate that deviation is low.

Median values do not reflect whether the data transfer was completed successfully for all runs or only for some of them. It just aggregates those that succeeded. Therefore, median values may indicate good results even if the experiment fails to complete the transfer in some cases. To avoid this misrepresentation, no values are presented for experiments where at least one run failed to complete the data transfer.

5.5 Baseline Case: Simple HIP

For evaluation of the baseline case, MN and PN use HIP's mobility and multi-homing extension to provide mobility support[1]. Unmodified TCP connections are used to transfer data. This is called the baseline case, because no measurements are possible without HIP or another mobility solution. TCP connections would break after initial connection phase, because MN

[1] Appendix B presents the implementation used for this thesis.

changes its IP address in the reconnection phase.

On Linux systems, TCP aborts a connection if the last retransmission is lost. For the baseline case, TCP is configured to use a maximum of 9 retransmissions, which corresponds to an abort timeout of 2 to 3 minutes, depending on the actual retransmission timeout values. Some other TCP implementations use an abort timeout of 2 minutes by default (see [Ste96, p. 299]).

5.5.1 Expected Behaviour

Changing the input parameters of the experiments can have two effects. First, experiments could fail to deliver the data for some parameter combinations. Second, it can influence net connect time of those that succeed. In the following, the expected behaviour is discussed.

Increasing duration of disconnection phase

With short disconnection phases, all runs of the experiments should complete and transfer the full amount of data. As long as disconnection phases are shorter than TCP's abort timeout, MN retransmits a segment any time after reconnection. Retransmitted segments are acknowledged by PN and MN can send new segments, thus restarting data transfer.

As durations of disconnection phases approach the abort timeouts, some experiment runs will fail. Section 4.3 already discussed that TCP retransmission timeouts do not occur at specific times, but underlie certain deviations. Therefore, in some runs the last retransmission will be scheduled before reconnection phase and for others in the reconnection phase. Only those runs where it is scheduled in reconnection phase can restart and complete data transfer.

Further increasing the duration of the disconnection phase should cause all runs of the experiments to fail. This is the case, when the disconnection phase always exceeds TCP's abort time.

So far, the impact of disconnection phase on the success of a data transfer was discussed. However, it influences TCP performance as well. An increase of the duration of disconnection phase can influence net connect time, either increasing or decreasing it.

If duration of the disconnection phase is longer, TCP's retransmission timeout periods are longer as well. Therefore, idle time can be longer if disconnected phase is longer. As a tendency, idle time should increase with longer disconnection phases. As idle time directly influences net connect time, net connect time should increase with longer disconnection phases as well.

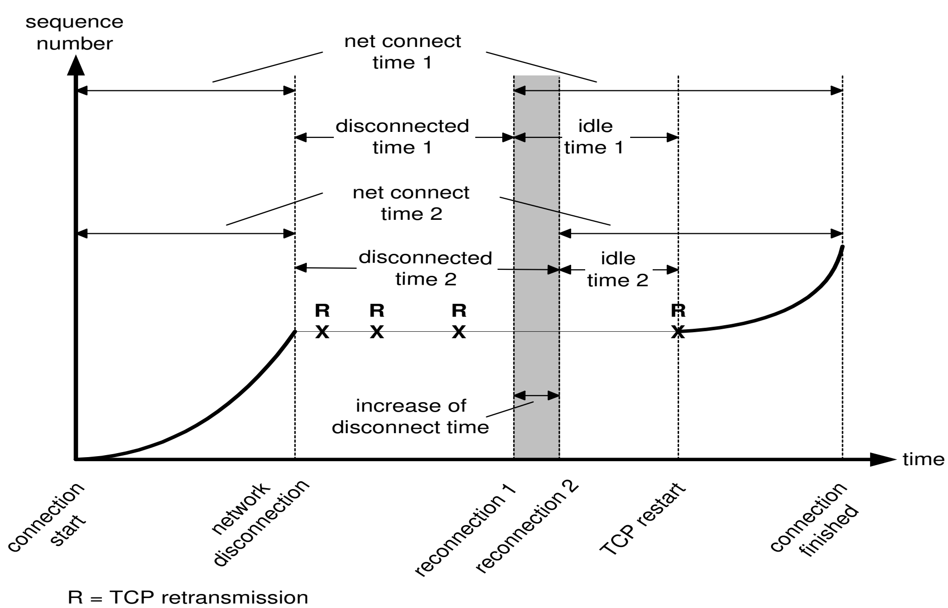


Figure 5.5: Idle time and net connect time with different disconnect times

However, if the increase of disconnection phase is less than the idle time, it only reduces the duration of idle time (see Figure 5.5). Total connect time will remain the same, as the same number of retransmissions is needed to restart data transfer. Consequently, net connect time is reduced by same amount as disconnection phase was increased.

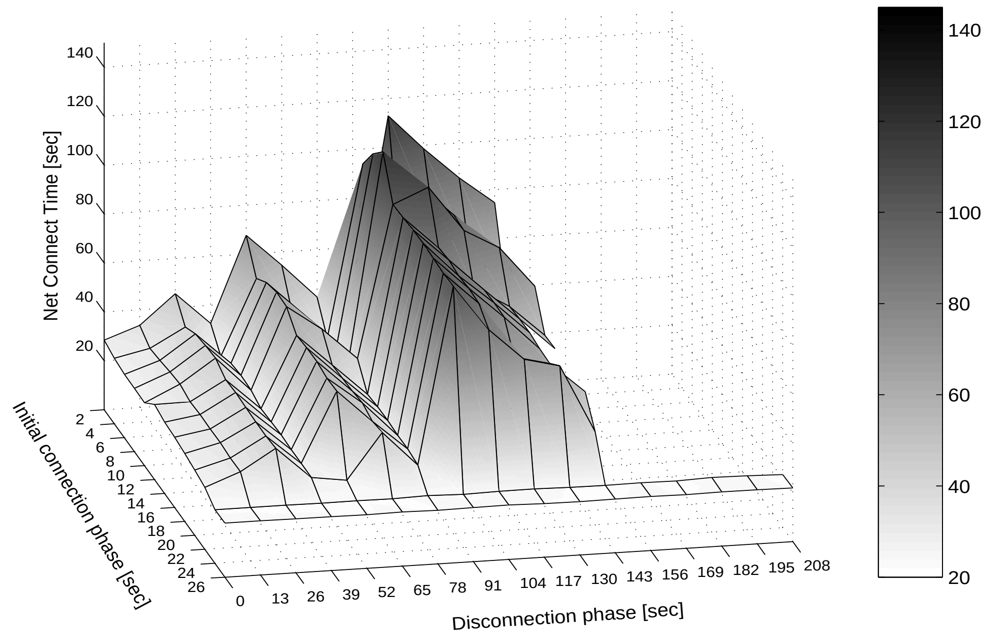
Altogether, it is expected that net connect time follows a zigzag course with increasing disconnection phases. If the increase of disconnection phase is less than current idle time, then net connect time decreases with the same amount as disconnection phase is increased. If increase of disconnection phase is larger than current idle time, then TCP needs additional retransmissions to restart data transfer. Net connect time grows by a value near the next TCP timeout value. As the TCP timeout value grows exponentially with the number of retransmissions, peaks of the zigzag course should grow exponentially as well.

Increasing duration of initial connection phase

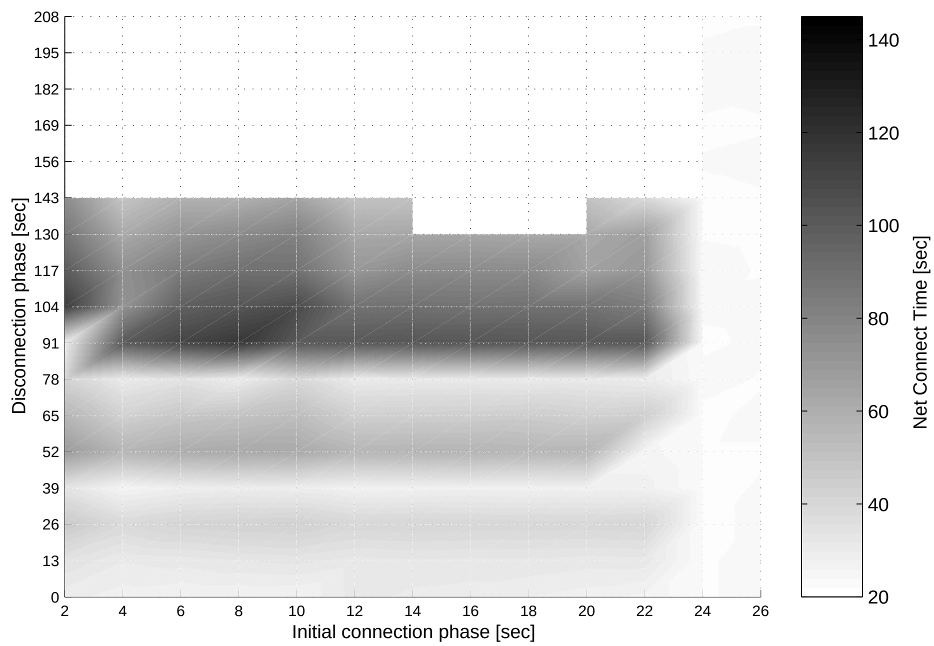
In the best case, data transfer already completes in initial connection phase. Net connect time cannot be lower than in this case, because TCP will reach steady state and send at maximum rate until data transfer is complete. Transfer of the 25 MB takes about 20 to 22 seconds (see Section 5.4.1). Consequently, when initial connection time exceeds 22 seconds, all connections should finish in the initial connection phase.

5.5.2 Measurement Results

Figure 5.6 shows the results for the baseline case. Results are presented as a surface plot and as a density plot. Both plots are based on the same data.



(a) Median net connect time (surface plot)



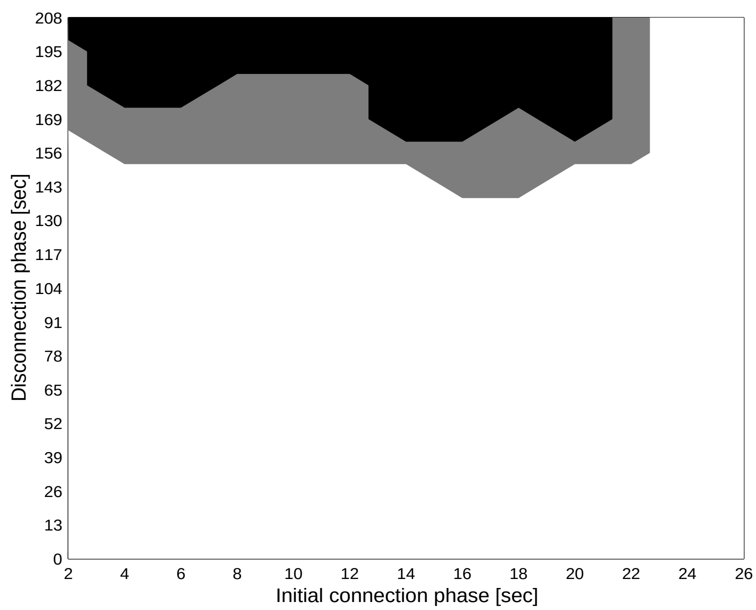
(b) Median net connect time (density plot)

Figure 5.6: Measurement results for Simple HIP

Increasing duration of disconnection phase

Up to a disconnection time of 130 seconds, data is available for all experiments. All runs for those experiments completed their data transfers and TCP connections were not aborted. This shows, that HIP is working as a mobility solution because TCP connections are able to deliver data after an IP address change in reconnection phase.

However, no data is available for disconnection phases that are longer than 143 seconds. This means, that at least one run per experiment failed to transfer the full amount of data. As expected, TCP connections are aborted if disconnection phase gets to too long. Figure 5.7 shows whether only some runs or all runs failed for an experiment. At the latest, all experiment runs



white: no run failed

grey: some runs failed

black: all runs failed

Figure 5.7: Failed experiment runs due to long disconnection phases

failed for disconnection phases of 195 seconds and more. This means, that no TCP connection survives a disconnection period that is longer than 195 seconds if it is using a maximum of 9 retransmissions.

Performance is generally bad with a net connect time of more than 100 seconds in the worst case. As expected, net connect time follows a zigzag course with increasing disconnection phase. Peaks are observed at disconnection phases of 39, 65 and 91 seconds. Peak values are increasing with longer disconnection phases up to values of more than 100 seconds, which corresponds to a network throughput of $\frac{25MB}{100s} = 2.0Mb/s$. This is only 20% of available bandwidth.

Increasing duration of initial connection phase

Experiments with an initial connection phase of 24 or 26 seconds finish their data transfer before disconnection. Otherwise, data transfer cannot complete as TCP connections are aborted on long disconnection phases. As already mentioned in Section 5.5.1, this kind of experiments highlights the best case, where net connect times are measured with about 22 seconds.

Influence of initial connection phase is very low when increasing from 12 to 20 seconds. Net connect time differs only about 1 second. In the evaluation network, only one TCP connection is active at a time. Thus, segment losses only occur if the data rate of this connection exceeds network capacity. Therefore, TCP raises its data rate until it reaches maximum data rate and then enters steady state. During the experiments, TCP reaches a steady state already after about 3 seconds. Consequently, TCP connections reach a steady state before and after disconnection phase for most of the experiments such that the duration of initial connection phase has only minor influences on net connect time.

However, some jitter is visible for lower initial connection times. This thesis does not investigate this effect further, but one explanation could be that TCP's round-trip time estimations are not very accurate at the beginning of a connection. If disconnection occurs in an early phase of a TCP connection, then retransmission timeouts will be based on inaccurate round-trip time estimations. Therefore, retransmission timeouts have a higher jitter if initial connection phase is shorter. This jitter is magnified by each retransmission attempt due to exponential backoff. Consequently, the net connect time is different for short initial connection phases, as it is dependent on retransmission timeout values.

Inter-Quartile gaps

High inter-quartile gaps of more than 40 seconds are mainly observed when initial connection phase is 22 seconds. This is due to the fact that some runs complete data transfer before disconnection phase and others have to complete in reconnection phase. For other initial connection phases, inter-quartile gaps are lower with a maximum of about 18 seconds.

Generally, inter-quartile gaps are higher near the peaks of net connect time. As described in Section 5.5.1, peak values are generated by delaying reconnection after a TCP retransmission. Consequently, reconnection and TCP retransmission are close to each other near peak values. Then, for some runs reconnection might occur before and for others after that retransmission. In the latter case, TCP needs an additional retransmission attempt. This leads to a higher net connect time. Therefore, net connect time shows higher differences for the two cases.

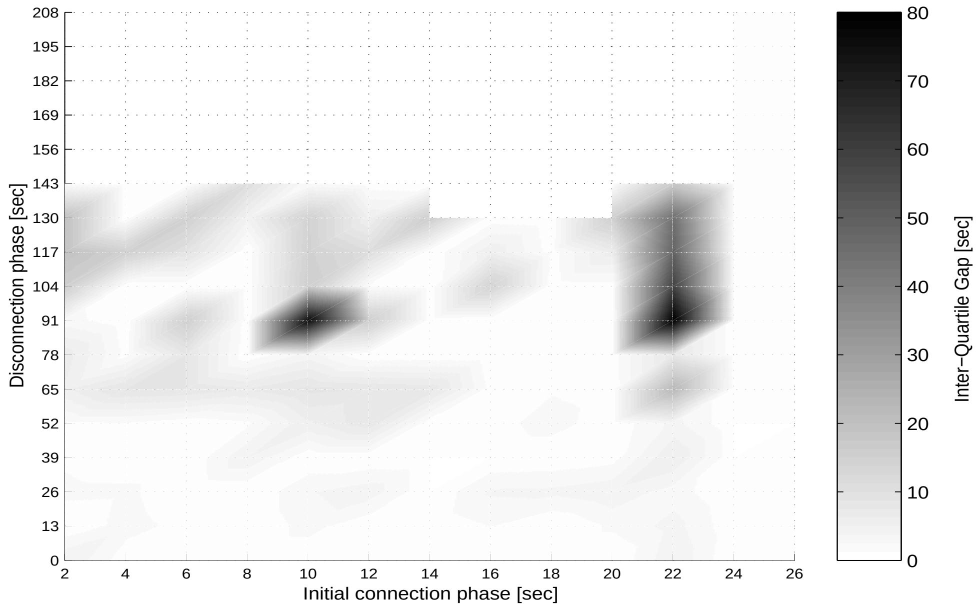


Figure 5.8: Simple HIP: inter-quartile gaps for net connect time

5.5.3 Summary

HIP proves to be a working solution to provide mobility support. Data transfer continues after reconnection even though IP addresses change. However, if disconnection phase is too long, TCP connections abort and data transfers do not complete. Net connect time is generally bad. In the worst case, it is more than 100 seconds, which is already more than four times the optimal value of 22 seconds.

As deviation is high for some experiments, results should be treated as a global picture. They do not reflect exact values, but show the general behaviour of TCP in disconnecting environments.

5.6 HIP with TCP ATO

Measurements presented in this section are based on systems using HIP with additional support for the TCP Abort Timeout Option. MN and PN can negotiate appropriate abort timeout values on connection setup. It is assumed that MN requests an abort timeout value that is larger than any disconnection that occurs in the scenario. In addition, PN is assumed to accept all timeout values proposed by MN. Both assumptions are reasonable because the longest disconnection phase is only 208 seconds.

The TCP Abort Timeout Option will be simulated because no implementation exists at this time. This is done by increasing the maximum number of TCP's retransmission attempts to 15 on both nodes. In this case, TCP abort timeout is about 15 to 16 minutes, depending on the first retransmission timeout value. Therefore, TCP connections use an abort timeout that is longer than the maximal disconnection phase of 208 seconds.

5.6.1 Expected Behaviour

Introducing the TCP Abort Timeout Option should allow all experiment runs to complete their data transfer. TCP retransmits data up to a duration of 15 or 16 minutes, but reconnection already happens after 208 seconds at the latest. Consequently, at least one retransmission should arrive at PN before the timeout. PN acknowledges reception and MN can restart sending data.

Changing the abort timeout of TCP does not influence its behaviour while the connection is in an established state. Thus, net connect time for experiments that succeed without a TCP Abort Timeout Option should be the same as before.

However, some experiments with long disconnection phases should result in net connect times that are even longer than in Section 5.5.2. As longer abort timeouts allow more retransmissions, the retransmission timeout values grow to larger values as well[2]. Consequently, TCP's idle time can be longer, too, which results in longer net connect time.

5.6.2 Measurement Results

Results are presented similar to Section 5.5.2. Same axis scaling and same greyscales are used to allow easy comparison of the plots.

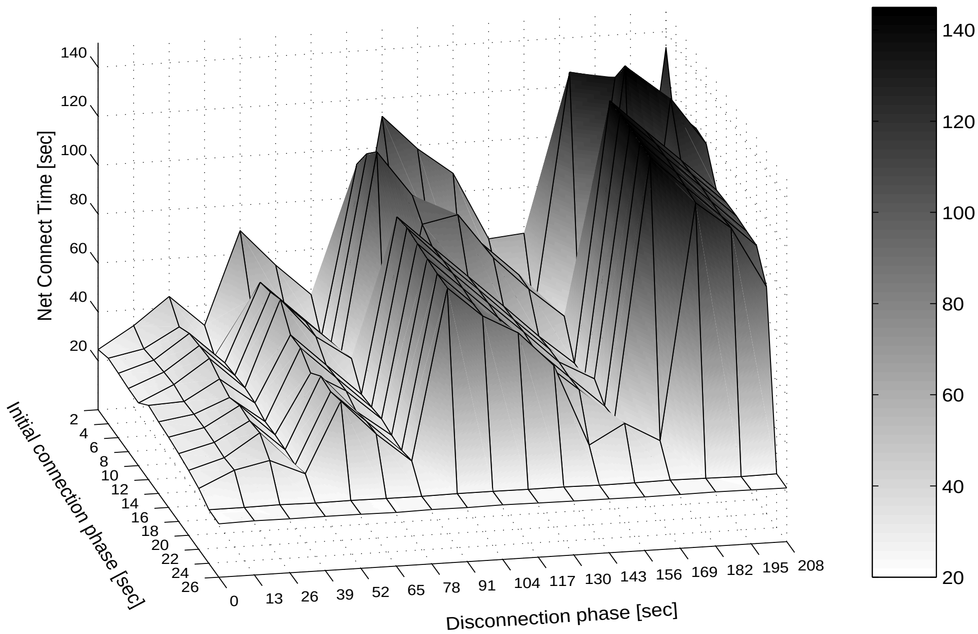
Figure 5.9 shows, that all runs complete now. Negotiating appropriate abort timeout values at connection setup allows connections to tolerate long periods of disconnection. If MN and PN request or accept those values, TCP connections are not aborted in disconnecting environments.

In addition, Figure 5.9 shows that net connect time is very high for disconnection phases of 169 seconds or more. Net connect time is then measured with values about 105 to 140 seconds in most of the cases. The worst case of 140 seconds is more than six times the optimum value of 22 seconds. The corresponding network throughput computes as $\frac{25MB}{140s} = 1.4Mb/s$, which is only 14% of the available bandwidth.

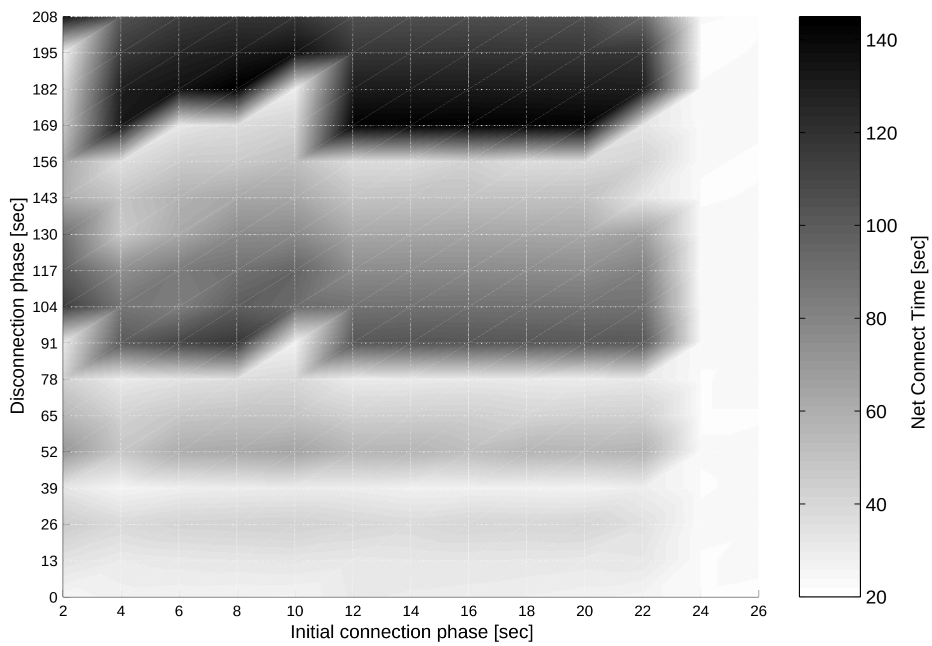
As expected, net connect time is similar to the results without a TCP Abort Timeout Option for shorter disconnection phases. Differences are less than 5% for most of the experiments.

Inter-quartile gaps are again measured with very high values of up to 70 seconds for initial connection times of 22 seconds (see Figure 5.10). For other parameter settings, inter-quartile gaps are measured with less than 15 seconds. Again, these deviations can be explained by the dynamics of

[2]The upper limit for retransmission timeouts is 120 seconds on Linux systems



(a) Median net connect time (surface plot)



(b) Median net connect time (density plot)

Figure 5.9: Measurement results for HIP with TCP Abort Timeout Option

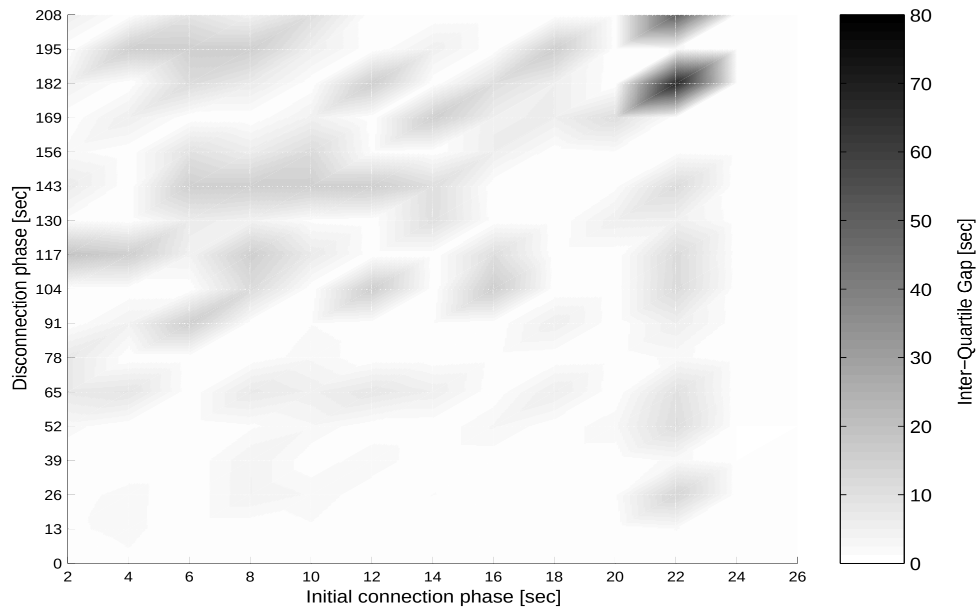


Figure 5.10: HIP with TCP ATO: inter-quartile gaps

TCP's retransmission timeout calculation and TCP's congestion avoidance algorithms. Results may not show exact values, but they still show the general characteristics of TCP in disconnecting environments.

5.6.3 Summary

Measurement results indicate that a TCP Abort Timeout Option helps to prevent TCP connections from aborting in disconnecting environments. However, performance is still very bad. In the worst case, TCP connections only use 14% of available bandwidth.

5.7 HIP with TCP ATO and Retransmission Trigger

For the last set of experiments, the TCP Retransmission Trigger was enabled in addition to HIP and TCP Abort Timeout Option. The HIP layer triggers TCP connections immediately after it completes a readdressing. Thereupon, TCP reschedules outstanding retransmissions to be sent at once.

5.7.1 Expected Behaviour

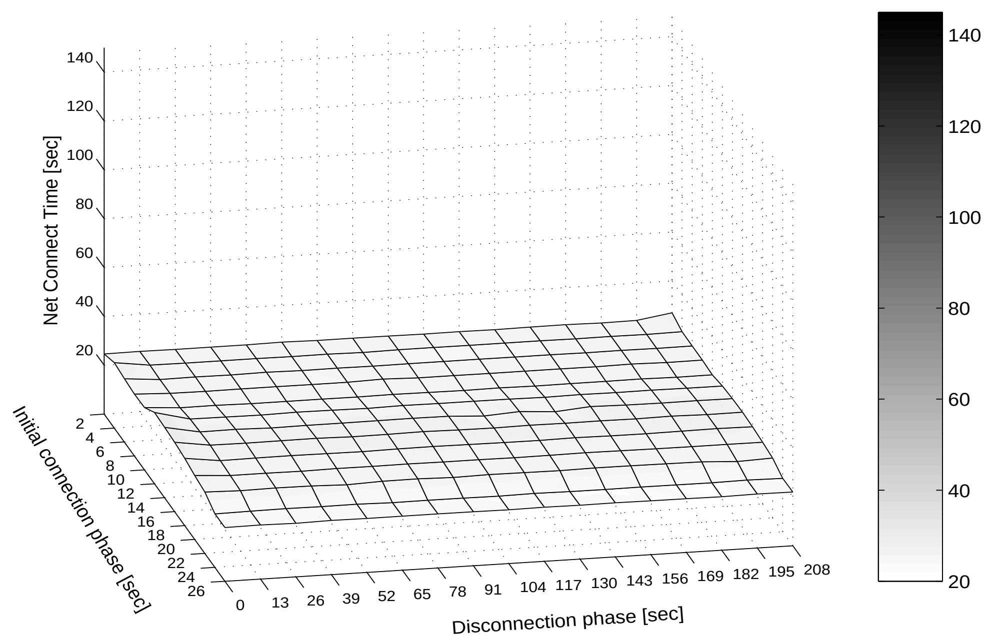
The TCP Retransmission Trigger causes a retransmission of lost segments immediately after HIP completes its readdressing handshake. HIP readdressing is executed as soon as a new connection is available. Consequently, TCP's idle time should be very short. If HIP readdressing always takes the same time, then idle time should not change for different disconnection phases. As net connect time is mainly influenced by idle time, it is expected to be rather independent from disconnection phase. Its value should be near to the sum of the optimum value of 22 seconds plus the time needed for HIP readdressing.

As a TCP Abort Timeout Option is used, all experiments should succeed in delivering the full amount of data.

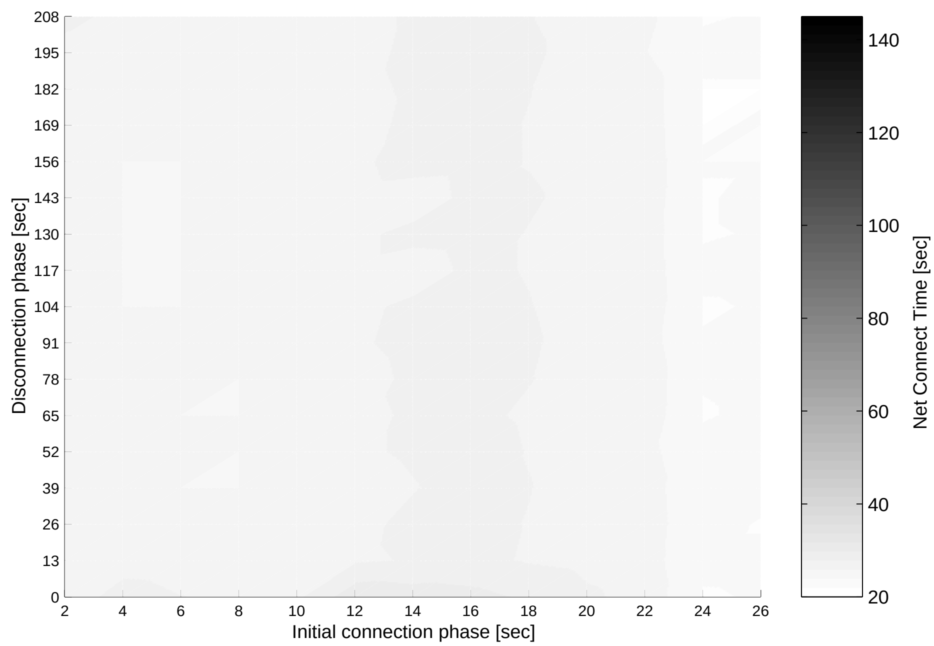
5.7.2 Measurement Results

Measurement results are presented similar to the previous results. Again, same axis scaling and same greyscales are used to simplify comparison of the results.

Figure 5.11(a) shows an almost flat surface plot for net connect time.



(a) Median net connect time (surface plot)



(b) Median net connect time (density plot)

Figure 5.11: Measurement results for HIP with TCP ATO and Retransmission Trigger

Accordingly, only few differences for grey-values are visible in the density plot (see Figure 5.11(b)). This shows that net connect time is now nearly independent of the durations of initial connection phase and disconnection phase. The TCP Retransmission Trigger seems to restart data transfer in a very reliable and homogenous manner.

Most experiments completed after about 24.5 to 27 seconds. This corresponds to a network throughput of 7.4 to 8.1 Mb/s and a usage of 74-81% of available bandwidth.

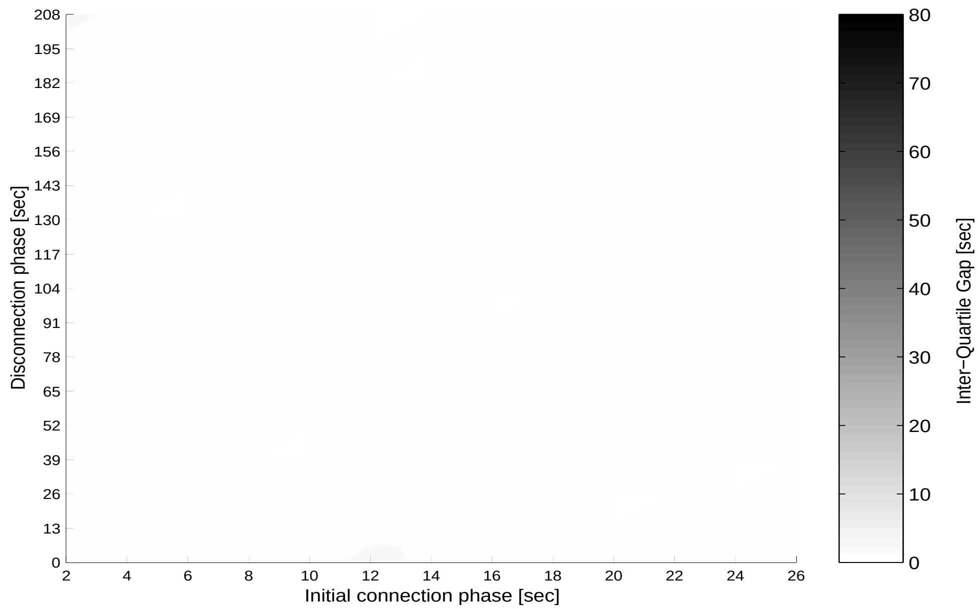


Figure 5.12: HIP with TCP ATO and Retransmission Trigger: inter-quartile gaps

Inter-Quartile gaps are very low with a maximum of only 2.5 seconds (see Figure 5.12). Again, this shows the reliability of the TCP Retransmission Trigger.

An additional analysis of packet flow was accomplished to measure the time that is needed for a HIP readdressing. HIP readdressing is a three-way packet exchange. First, MN informs PN about its new IP address. Then, PN sends an Address Check packet to MN. Last, MN confirms its new address by sending an Address Check Reply packet to PN.

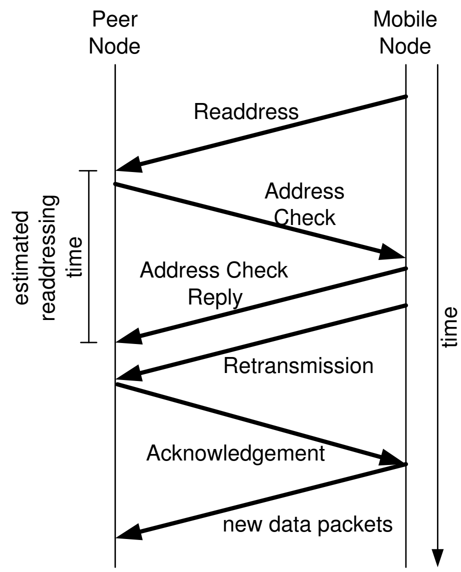


Figure 5.13: Packet flow after reconnection

The packet flow after reconnection is shown in Figure 5.13. The time for a HIP readdressing was estimated by measuring the time slip between reception of the first readdressing packet and the address check reply packet. This time difference is about 2.15 seconds in the mean.

5.7.3 Summary

The TCP Retransmission Trigger seems to work very well for the evaluation scenario. Net connection time is mostly reduced to values of 24.5 to 27 seconds. These values already include the time that is needed for the HIP readdressing. During readdressing, no data is sent to PN. Consequently, TCP remains idle for at least 2.15 seconds after reconnection. Still, 24.5 to

27 seconds is already quite near to the optimum value of 22 seconds. It has to be considered, that the optimum value is based on a data transfer without readdressing. Thus, it does not experience any idle time.

5.8 Discussion

Measurement results indicate that all three parts of the proposed system improve different aspects of the scenario. Together they let TCP tolerate mobility and long periods of disconnection and reduce net connect time by nearly 120 seconds in the best case. HIP provides mobility support, such that IP address changes are transparent to TCP connections. Modifying TCP abort timeouts with the TCP Abort Timeout Option helps TCP connections to survive long periods of disconnection. Last, the TCP Retransmission Trigger proved to restart data transfer from mobile node to peer node in a reliable manner. Net connect time is reduced to a value near the optimum in this case.

22 seconds seem to be the lower limit for net connect time to transfer 25 MB of data with TCP over HIP. Some additional measurements showed that transferring this amount of data with TCP over pure IP connections takes about 21 seconds in the same network. The overhead that HIP introduces can be estimated as only $\frac{22s-21s}{22s} = 4.5\%$. However, this overhead can increase in faster networks. As all traffic running on HIP connections is encrypted by IPsec, CPU power is a potential bottleneck[3] for network throughput.

As already mentioned before, the mobile node does not use IPv6 autoconfiguration. In reality, most IP address changes after a movement are

[3]On the machines used for the evaluation, maximal throughput on HIP connection was limited with 70 Mb/s. In this case, CPU usage was 100%.

managed by DHCP or IPv6 auto-configuration. This would add some additional idle time after reconnection. First, the IP address has to be configured automatically. Only when this is finished, HIP will start its readdressing. In the evaluation, this additional idle time is not included, as reconnection is simulated by configuring the IP address manually. Thus, reconnection and IP address happen at the same time.

5.9 Summary

This chapter evaluated different parts of the concept by measuring net connect time for a data transfer from a mobile node to a peer node. Measurements in Section 5.5 show, that HIP solves the mobility problem. TCP connections can survive IP address changes if they bind their end-points to Host Identity Tags instead of IP addresses. Section 5.6 tested the TCP Abort Timeout Option. If this option is used to extend TCP abort timeout values, then TCP connections are able to survive long periods of disconnection. Last, measurements of Section 5.7 show, that the TCP Retransmission Trigger dramatically decreases net connect time for the evaluation scenario.

Chapter 6

Conclusion and Outlook

6.1 Summary

In the basic scenario of this thesis, mobile users are travelling while using mobile devices such as notebooks, PDAs or mobile phones to connect to the Internet. When entering areas with bad network coverage, they temporarily loose connection. They change access networks when moving to a different location, which implies IP address changes. Ongoing connections are disrupted in such a scenario, because connections are bound to fixed IP addresses. TCP connections are also aborted if disconnection periods are too long. Disconnections decrease TCP performance because data transfer is not immediately resumed when network connectivity is re-established. Instead, TCP waits for retransmission timeouts that grow exponentially during periods of disconnection.

This thesis proposes a system that enables TCP connections to efficiently transfer data in the presence of IP address changes and periods of disconnection. It consists of three complementary parts. HIP is the first part. It decouples transport layer connections from IP addresses. Consequently, connection end-points remain valid even if communicating nodes change IP addresses due to mobility or temporary disconnection. The second part is the TCP Abort Timeout Option, which allows communicating nodes to negotiate and extend abort timeout values for TCP connections. Thus, connections can tolerate arbitrary-long periods of disconnection if communicating nodes negotiate sufficiently large abort timeout values. The third and last part of the system is a TCP Retransmission Trigger. It increases TCP performance in disconnecting environments. The retransmission trigger schedules retransmissions immediately after network connectivity is re-established, thus restarting idle TCP connections earlier than the usual retransmission timeouts.

The measurements presented in Section 5 indicate that the proposed system is very effective. TCP connections tolerate IP address changes as well as periods of disconnection. Net connect time is independent of the duration of disconnection periods and reduced by a maximum of nearly 120 seconds and by 43 seconds on average in the presence of one intermediate disconnection.

Finally, the proposed system is well supported by current Internet infrastructure, because all three parts of the system operate on an end-to-end bases. Mobility and disconnection support is entirely implemented on end-hosts such that data is transferred as standard IP traffic, which does not rely on any modifications on intermediate routers.

6.2 Future Work

6.2.1 Abort Timeout Policies

The TCP Abort Timeout Option offers a mechanism to negotiate on abort timeout values. However, the negotiating nodes still have to choose appropriate values. This thesis does not address such selection policies. During evaluation, all nodes were assumed to propose and accept abort timeout values that are sufficiently large. Adaptive selection policies should be developed to choose timeout values that are large enough to tolerate disconnection and short enough to prevent resource exhaustion. For example, policies could be based on information about geographic location of nodes and access points, movement directions and speed or current resource utilisation of the communicating hosts.

6.2.2 Connection Selection for TCP Retransmission Trigger

The TCP Retransmission Trigger is designed to work well in conjunction with HIP. TCP connections that should be triggered are selected on the basis of Host Identifiers (see Section 4.3.3). Another selection mechanism has to be introduced, such that nodes can benefit of a TCP Retransmission Trigger even if they are not using HIP. One possible approach could be that a caller passes a pointer to a evaluation function to the TCP Retransmission Trigger. This function must be able to conclude whether a TCP connection can benefit from rescheduling of retransmissions. This decision can be based on TCP socket information. The TCP Retransmission Trigger should only reschedule retransmissions for those TCP connections that are selected by the evaluation function.

6.2.3 HIP Extensions

The HIP protocol architecture proposes the use of rendezvous servers for frequently moving nodes. However, no exact specification of a rendezvous server is currently available. A common standard has to be defined. Eggert proposes the use of *rendezvous brokers* [Egg04a]. A rendezvous broker is a rendezvous servers that provides bridging functionality for communication between HIP and non-HIP nodes.

HIP communication is always encrypted which is unnecessary for many applications. In such cases, encryption produces a high overhead on end-hosts such that CPU power might become the limiting factor. Therefore, applications could benefit from a new HIP option that allows non-encrypted data transfer.

6.2.4 Optimisation of Throughput during Connected Time

This thesis concentrates on maximizing TCP's usage of net connect time by minimizing idle time. In this way, TCP restarts data transfer earlier after a reconnection.

No changes are applied to TCP's congestion control algorithms. TCP still performs slow start at connection setup and after each reconnection even if network conditions do not change. Thus, TCP throughput during connected time could be further optimised if data rates would be adapted to current network conditions faster than with current congestion control algorithms. XCP [KHR02] is one example for such an approach.

6.2.5 Experimental Evaluation for Interactive Applications

This thesis evaluated the proposed system based on bulk data transfer from a mobile node to a peer node. The results show that the system is very effective within the experimental scenario. Further evaluations could be carried out for interactive communications. The TCP Retransmission Trigger should improve responsiveness of interactive applications, because it decreases TCP's idle time after reconnection.

6.2.6 Disconnection Tolerance for Other Mobility Solutions

In this thesis, the TCP Abort Timeout Option extends TCP to deal with periods of disconnection. This kind of disconnection tolerance works well as long as TCP is used as the transport protocol. Therefore, the proposed disconnection tolerance solution is expected to function in conjunction with mobility solutions that operate only on layers lower than transport layer (e.g. Mobile IP). Further evaluations could be carried out for mobility solutions other than HIP to verify this assumption.

Appendix A

Source Code

The implementation of the TCP Retransmission Trigger is described in Section 4.3.5. The corresponding source code of function `tcp_trigger_retransmit_timers` is presented in the following.

```
void tcp_trigger_retransmit_timers(unsigned int when) {
  struct tcp_opt *tp = NULL;
  struct sock *sk = NULL;
  struct tcp_ehash_bucket *ebucket = NULL;

    /* loop over all known open tcp sockets */
  int i;
  /* for each bucket in hash table */
  for (i = 0; i < tcp_ehash_size; i++) {
    ebucket = &tcp_ehash[i];
    read_lock(&ebucket->lock);
    /* for each socket in bucket */
    for (sk = ebucket->chain; sk != NULL; sk = sk->next) {
```

```
      /* check for inet6 family */
      if (sk->family != PF_INET6)
        continue;
      tp = &sk->tp_pinfo.af_tcp;
      /* check whether retransmissions scheduled */
      if (!tp->retransmits) {
        continue;
      }

      /* override retransmission timer if retransmission */
      /* gets pre-scheduled */
      if (jiffies+when < tp->retransmit_timer.expires) {
        tcp_reset_xmit_timer(sk,TCP_TIME_RETRANS,when);
      }
      /* decrease number of retransmits and backoffs, */
      /* because this is an additional retransmit*/
      if (tp->backoff > 0)
        tp->backoff--;
      tp->retransmits--;
    }
    read_unlock(&ebucket->lock);
  }
}
```

Appendix B

HIP for Linux

Helsinki Institute for Information Technology and Helsinki University of Technology are developing a HIP implementation for Linux (HIPL)[1]. Current Implementation supports IPv6 and includes the mobility and multi-homing extension. HIPL is based on Linux 2.4.20 kernel. The experimental evaluation presented in chapter 5 uses HIPL patch version 60.

HIPL provides a new libinet6 library. Applications either have to link dynamically to the new libinet6 library or they have to be recompiled if they link statically. The library offers a new version of function `getaddrinfo`, which is used to lookup an IP address of a host. If a HIT is registered for this host further communication is run over HIP. Otherwise, communication uses standard IP.

[1]HIPL is available at http://gaijin.iki.fi/hipl

Bibliography

[AS03] Ilknur Aydin and Chien-Chung Shen. Cellular SCTP: A Transport-Layer Approach to Internet Mobility, October 2003. Internet Draft, work in progress.

[DW03] Spencer Dawkins and Carl Williams. End-to-End, Implicit "Link-Up" Notification, October 2003. Internet Draft, work in progress.

[Egg04a] Lars Eggert. Host Identity Protocol (HIP) Rendezvous Mechanisms, February 2004. Internet Draft, work in progress.

[Egg04b] Lars Eggert. TCP Abort Timeout Option, April 2004. Internet Draft, work in progress.

[FS00] Paul Ferguson and Daniel Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing, May 2000. RFC 2827.

[JPA03] David Johnson, Charles Perkins, and Jari Arkko. Mobility Support in IPv6, June 2003. Internet Draft, work in progress.

[KA98a] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol, November 1998. RFC2401.

[KA98b] Stephen Kent and Randall Atkinson. IP Encapsulating Security Payload (ESP), November 1998. RFC2406.

[KHR02] Dina Katabi, Mark Handley, and Charlie Rohrs. Internet Congestion Control for Future High Bandwidth-Delay Product Environments, August 2002. ACM Sigcomm 2002.

[KMC⁺00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[MN03] Robert Moskowitz and Pekka Nikander. Host Identity Protocol Architecture, September 2003. Internet Draft, work in progress.

[MNJ03] Robert Moskowitz, Pekka Nikander, and Petri Jokela. Host Identity Protocol, June 2003. Internet Draft, work in progress.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[NA03] Pekka Nikander and Jari Arkko. End-Host Mobility and Multi-Homing with Host Identity Protocol, December 2003. Internet Draft, work in progress.

[Per02] Charles Perkins. IP Mobility Support for IPv4, August 2002. RFC3344.

[Pos81] Jon Postel. Transmission Control Protocol, September 1981. RFC793.

[QYB97] Xun Qu, Jeffrey Xu Yu, and Richard P. Brent. A Mobile TCP Socket. Technical Report TR-CS-97-08, Department of

Computer Science, Australian National University, Canberra, April 1997.

[RT03] Maximilian Riegel and Michael Tuexen. Mobile SCTP, August 2003. Internet Draft, work in progress.

[SB03] Keith Scott and Scott Burleigh. Bundle Protocol Specification, October 2003. Internet Draft, work in progress.

[SM03] James Scott and Glenford Mapp. Link Layer-Based TCP Optimisation for Disconnecting Networks. *ACM SIGCOMM Computer Communications Review*, 33(5):31–42, October 2003.

[Sno03] Mark Alexander Connell Snoeren. *A Session-Based Architecture for Internet Mobility*. PhD thesis, Massachusetts Institute of Technology, February 2003.

[SRX+03] Randall R. Stewart, Michael A Ramalho, Qiaobing Xie, Michael Tuexen, Ian Rytina, Maria-Carmen Belinchon, and Phillip T. Conrad. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration, September 2003. Internet Draft, work in progress.

[Ste96] W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley Longman Inc., October 1996.

[SXM+00] Randall R. Stewart, Qiaobing Xie, Ken Morneault, Chip Sharp, Hanns Juergen Schwarzbauer, Tom Taylor, Ian Rytina, Malleswar Kalla, Lixia Zhang, and Paxson Vern. Stream Control Transmission Protocol, October 2000. RFC2960.

[Tan96] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, Inc., third edition, 1996.

[XKWM02] Wei Xing, Holger Karl, Adam Wolisz, and Harald Müller. M-SCTP: Design and Prototypical Implementation of an End-To-End Mobility Concept. In *Proc. 5th Intl. Workshop, The Internet Challenge: Technology and Applications*, Berlin, Germany, October 2002.

[YGD+01] Praveen Yalagandula, Amit Garg, Mike Dahlin, Lorenzo Alvisi, and Harrick Vin. Transparent Mobility with Minimal Infrastructure. Technical Report TR01-30, University of Texas, Austin, July 2001.

[ZD95] Yongguang Zhang and Son Dao. A "Persistent Connection" Model for Mobile and Distributed Systems. In *Proceedings of the 4th International Conference on Computer Communications and Networks*, pages 300–305, Las Vegas, NV, September 1995.

Index

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mannheim, den   Simon Schütz