Vincent Jacquot

Test Bed for Multipath TCP

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 23.05.2012

Thesis supervisor:

Prof. Jukka Manner

Thesis advisor:

M.Sc. (Tech.) Lennart Schulte

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Vincent Jacquot

Title: Test Bed for Multipath TCP

Date: 23.05.2012 Language: English Number of pages:7+57

Department of Communications and Networking

Professorship: Networking technology Code: S-38

Supervisor: Prof. Jukka Manner

Advisor: M.Sc. (Tech.) Lennart Schulte

It was a common assumption on the Internet that endpoints access the network through only one interface. But now, most mobile devices support several access technologies and can have several IP interfaces. This diversity can be used to improve network performance. Traditional protocols do not support multi-access thus new solutions have to be deployed.

This thesis explores the use of MultiPath TCP as a solution to take advantage of multiple interfaces on mobile endpoints. It considers more specifically the case of devices that support several 3G subscriptions. MPTCP can use several interfaces concurrently but it raises a problem of power consumption. Dynamic interface selection is proposed as a way to make a tradeoff between performance and battery life.

To evaluate the performance of MPTCP and the different path selection algorithms in real world conditions, this work included the design and implementation of a test bed based on a laptop and three mobile phones. The results showed that MPTCP with or without interface selection can improve the performance of TCP connections on multihomed endpoints.

Keywords: MultiPath TCP, Mobile devices, Multihoming, Interface selection, Test bed

Acknowledgements

I want to express my gratitude to Prof. Jukka Manner for giving me the opportunity to conduct my master's thesis at the department of Communications and Networking of Aalto Univesity School of Electrical Engineering.

I would also like to thank my instructor Lennart Schulte for all the valuable advices and comments on this work.

Otaniemi, 22.5.2013

Vincent Jacquot

Contents

Abbreviations

API	Application Programming Interface
BSD	Berkeley Software Distribution
CMT	Concurrent Multipath Transfer
CPU	Central Processing Unit
CSV	Comma Separated Values
DAR	Dynamic Address Reconfiguration
DSS	Data Sequence Signal
GPRS	General Packet Radio Service
GPS	Global Positioning System
GUI	Graphical User Interface
HMAC	Hash Message Authentication Code
HTTP	Hyper Text Transfer Protocol
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
LAN	Local Area Network
LTE	Long Term Evolution
MADM	Multi Attribute Decision Making
MCTCP	Multi-Connection Transmission Control Protocol
MPTCP	MultiPath Transmission Control Protocol
MSS	Maximum Segment Size
MVNO	Mobile Virtual Network Operator
NAT	Network Address Translator
nmcli	NetworkManager Command Line Interface
PEAP	Protected Extensible Authentication Protocol
PIN	Personal Identification Number
PLMN	Public Land Mobile Network
PLMT	PayLoad Multi-connection Transport
PPP	Point-to-Point Protocol
pTCP	Parallel TCP
QoS	Quality of Service
RFC	Request For Comments

RTT	Round-Trip Time
RTP	Real Time Protocol
MPRTP	MultiPath Real Time Protocol
SCTP	Stream Control Transport Protocol
SHA	Secure Hash Algorithm
SIM	Subscriber Identification Module
SIP	Session Initiation Protocol
SQL	Standard Query Language
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TLV	Type-Length-Value
UDP	User Datagram Protocol
UMA	Unlicensed Mobile Access
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
WLAN	Wireless Local Area Network
3GPP	Third Generation Partnership Project

1 Introduction

In recent years, the success of mobile devices such as smartphones and tablet computers changed the common assumptions about how users will access the Internet. Those devices represent a more important part of the total traffic and that part is still expected to increase [1]. In Europe, the number of mobile subscriptions per capita is now higher than one [2]. At the same time, applications available on mobile devices become closer to those traditionally reserved to laptops and desktop computers. Thus, users may expect the same level of responsiveness from their network applications regardless of the kind of device. In addition, due to the power and CPU limitations of mobile devices, the trend is to rely on cloud computing to bring heavy applications to mobile environments. Hence, the quality of the network access will be even more important for mobile applications.

This brings forth new challenges as mobile devices do not always have access to the reliable and efficient wired connections available to non-mobile computers in home or corporate networks. To access the Internet, mobile endpoints typically use a variety of mobile networks whose capacity and reliability depend on time and location. This results in frequent changes in throughput or latency, especially while moving fast. Some applications, such as video or audio streaming, may fail to handle those events.

1.1 Initial motivation

Mobile devices may lack a reliable connection but they could take advantage of the diversity of the networks to which they have access. Recent mobile endpoints usually support Wireless LAN and different cellular network technologies such as UMTS or GPRS. According to the resource pooling principle introduced by Whischik et al. [3], using different paths concurrently between two endpoints would increase the throughput, improve the response to burst traffic and strengthen the robustness of the link.

Mobile endpoints would greatly benefit from such improvements. Availability and capacity of wireless networks can largely vary depending on the location, the time of day, and, of course, the operator. At the same time, due to both user and application behavior, incoming traffic is bursty. Pooling the capacity of the different available networks would help to handle traffic surges and to improve connection reliability.

Whischik et al. [3] also suggest the use Multipath TCP as a way to achieve resource pooling. These extensions to TCP support the use of several IP addresses by a single TCP connection. As only the endpoints would need to be updated, it could be easily deployed. Furthermore, MPTCP can be transparent to the application layer. This advantage is not negligible as most common network applications use TCP as their transport protocol.

1.2 Interface selection

The results of this work will confirm that, if several wireless networks are simultaneously available, using MPTCP can considerably improve the reliability and the throughput of TCP connections. However, using a radio interface requires power, and using all of them at the same time may not be the best option in terms of energy efficiency. This is a real problem since power consumption is a key concern on mobile devices. Thus trying to use the available throughput on every path may come at too high a price for the user that favors battery life over network performance.

Nevertheless, as MPTCP is able to recover from interface failures by switching to backup interfaces, it still improves reliability. Thanks to its make-before-break design, it can also move the traffic from one interface to another without interrupting the TCP connection. Thus, even though only one interface is active, the mobile device will still benefit from its other interfaces in case of failure, improving the robustness of the connection. Furthermore, by providing a way for TCP connections to be moved from one interface to another without any interruption, MPTCP allows dynamic interface selection. It means that it can be operated along an interface scheduler that dynamically chooses the best active interface in order to increase performance without sacrificing energy-efficiency.

1.3 Objectives

There could be many different ways to design such interface schedulers. If some of them have the ambition to be largely deployed on mobile endpoints, they will eventually need to be tested in real conditions. The main goal of this work was to implement a test bed to evaluate the performance of MPTCP, along different path selection algorithms, in real-world situations. This thesis considers the case of endpoints that can use several 3G connections simultaneously.

The problem of the routing configuration for MPTCP has to be solved by the test bed. Tools to easily test different dynamic interface selection algorithms have to be developed. Then, the performance of MPTCP and of different interface selection methods has to be evaluated.

1.4 Results

A test bed for MPTCP was developed in the Python programming language. It provides tools for implementing and evaluate path selection algorithms for MPTCP. A generic solution to automatically configure routing tables on MPTCP-enabled Linux systems was developed.

The results of real-world experiments proved that MPTCP performs well over several 3G networks and in a mobile environment, considerably improving bandwidth and latency. Measurements also showed that some simple dynamic interface selection processes can constitute a reasonable compromise between performance and power consumption.

1.5 Structure

The first chapters will present related works and background elements relevant to the main issues of the topic. That is to say:

- Multi-access and resource pooling on mobile devices
- MPTCP operation
- Network interface selection for mobile devices

Chapter 2 will review the multi-access solutions available on mobile endpoints, that is to say protocols that could take advantage of multiple interfaces or achieve some degree of resource pooling. Chapter 3 will focus on MPTCP concepts and mechanisms. Chapter 4 will present the issue of interface selection for mobile devices and will introduce the interface selection processes that has been experimented for this thesis.

The design of the test bed and the implementation of its different components will be described in Chapter 5. Chapter 6 will present the experiments and analyze their results. The last chapter concludes the work and mention possible applications and future works.

2 Multi-access on mobile hosts

Most recent mobile devices support several access technologies, such as UMTS and WLAN. Taken separately, these technologies may have significant shortcomings compared with a wired connection (lower throughput and intermittent interruptions). However, mobile endpoints could be able to use these interfaces simultaneously or sequentially to improve the overall quality of the connection. Unfortunately, none of the widely-deployed protocols on the Internet supports multihomed endpoints at the moment. This feature has to be added by a new protocol or an extension of an existing one.

Two key terms have to be defined before presenting the issue.

Multihoming
: A multihomed device or endpoint is as a host with several active IP interfaces. For instance, a laptop that maintains a wireless connection via WLAN and a wired connection via Ethernet is multihomed.

Vertical handover
: Vertical handover is the ability to switch between two different access technologies without interrupting the ongoing connections. As an example, a smartphone would perform a vertical handover if it moved from an UMTS cellular connection to WLAN without interrupting ongoing connections. Handover capabilities can ensure the continuity of the connections provided that the network coverage of different access points or operators overlaps as in Figure 1.

Schmidt et al. [4] listed the main multi-access solutions for mobile endpoints and classified them according to the layer at which they operate. Since different layers tend to provide different functionalities, this is reasonable to wonder at which layer multi-access should be supported. Other criteria are also taken into account, such as concurrent multipath transfer and flow scheduling capabilities. It is also crucial to consider potential obstacles to deployment and adoption.

Different protocols that support multiple interfaces will be presented and their advantages and drawbacks will be discussed. The conclusion will explain why TCP-based solutions are considered as the easier to deploy.

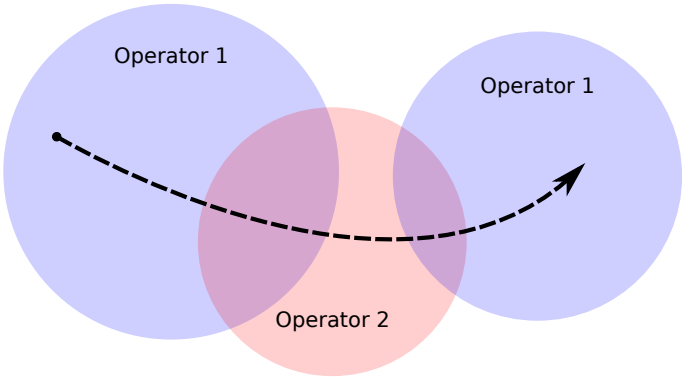

Figure 1: Vertical handover allows reliability pooling between different networks

2.1 Offloading

Operators have already considered using the WLAN capability of mobile devices to reduce the load on their cellular networks [5, 6]. A mobile device that primarily uses a 3G or GPRS access can use opportunistic connections with other networks (e.g. WLAN access points) to increase the available throughput and move the traffic away from the cellular network. This is called offloading. The 3GPP developed its own offloading technologies. Schmidt et al. [4] give a short description of some of these solutions. Modifications and additional equipment in the network are necessary to deploy these technologies. When an endpoint has access to a wireless LAN, it needs to connect to a specific gateway that provides access to the core of the 3G network of its service provider [7]. That is why these solutions cannot provide resource pooling between cellular networks of different operators.

2.2 Mobile IP

Mobile IP adds mobility support at the network layer [8]. It is developed by the IETF and supports both IPv4 and IPv6 [9, 10]. It requires an additional node with a public address in the network, the *home agent*, that acts as a relay by receiving the packets on behalf of the mobile host before forwarding them through an IP tunnel. The home agent is informed of the current IP address of the mobile device either by additional nodes, called *foreign agents*, or by the mobile endpoint itself.

This mobility mechanism supports vertical handover thus provides some reliability pooling. Nevertheless, it does not allow the endpoints to concurrently use multiple interfaces to improve throughput since multihomed devices can only use one of their IP interfaces at a time. In addition, the presence of a relay (the home agent) between the two endpoints may increase the latency, especially if both hosts are far from the relay. Finally, mobile IP is in competition with the 3GPP solutions. As a result, it has not been widely adopted as it was expected [11].

2.3 SCTP

SCTP is a transport layer, connection oriented protocol originally designed for signaling specified by the IETF [12]. One of the main strengths of this protocol is its ability to support several streams per association [1]. As a stream is established between two IP interfaces, a multihomed endpoint that uses SCTP would be able to create different streams over different paths and achieve resource pooling at the transport layer.

However, it does not natively support vertical handover. Indeed, SCTP allows endpoints to exchange their available IP addresses only during the establishment of the connection. Furthermore, SCTP uses only one stream at a time for data transfers and cannot take advantage of the pooled capacity of the interfaces to increase throughput. But some extensions were designed to answer these shortcomings.

The Dynamic Address Reconfiguration extension (DAR) [13] permits an SCTP connection to dynamically change the set of active IP addresses. DAR can be used to add the support of vertical handover to SCTP [14, 15]. An extension adds the support of Concurrent Multipath Transfer (CMT) to SCTP [16] to increase the throughput of the association through capacity pooling [17, 18]. SCTP implementations are available for Linux and BSD and DAR is supported on these platforms.

As opposed to TCP and UDP, it natively supports multihoming, thus it could have been a straightforward solution to achieve resource pooling on multihomed mobile devices. Unfortunately, several obstacles hinder its adoption. Firstly, many network devices, such as NATs and firewalls, do not support SCTP [19]. Even if some endpoints support SCTP, it would not be functional for general use with mobile devices since NATs are omnipresent. One solution would be to operate SCTP on top of UDP [20] but it would require further modifications of the current implementations. Finally, SCTP would have to be supported by applications. Software and application-level protocols would need to be modified so as to use that new transport protocol instead of the widely used and known TCP and UDP.

2.4 TCP-based solutions

In 2002, Hsieh and Sivakumar proposed the pTCP protocol [21]. It has been designed to allow throughput aggregation between different interfaces and already considered multihomed mobile devices as a potential use case. They provided simulation results but their protocol was never implemented or specified.

An IETF draft [22] published in 2009 describes a set of TCP options to simultaneously use different interfaces at the sender side. However, in the case of mobile devices, this is of limited interest. Indeed, since users tend to download more content than they upload, the sender side is not the mobile endpoint but some server on the public Internet. Thus, it does not allow the mobile device to concurrently receive data on its several radio interfaces. Even if the server has several interfaces, it may not help to improve throughput or reliability significantly as the bottleneck

[1] SCTP uses the term association instead of connection.

of the connection is more likely to be the radio interface of the mobile system and not the typically fast and reliable Internet access on the server side.

The IETF Multipath TCP working group chose MPTCP as a solution to add multihoming support to TCP. RFC 6182 [23] depicts the MPTCP architecture. It works on both sender and receiver sides. RFC 6824 [24] specifies the new TCP options MPTCP needs for signaling. It is transparent to applications, transparent to most NATs, supports subflow priority and can fall back to regular TCP in case of failure. It can add new interfaces to the connection after the initial establishment thus it supports vertical handover. Last but not least, an implementation of MPTCP in the Linux kernel is available [25, 26].

Nevertheless, some middleboxes that strip TCP options or perform packet inspection to check the payload may prevent MPTCP from working. Moreover, the limited size of 40 bytes for TCP options may be limiting for additional features such as authentication algorithms that would require long keys. That is why other proposals suggested to limit or give up the use of TCP options and embed the signaling in the TCP payload.

Multi-Connection TCP (MCTCP) [27, 28] is similar to MPTCP but only a few TCP options are exchanged during the establishment of TCP flows. The other signaling messages are carried in the payload. As long as only one interface is active, MCTCP behaves like a normal TCP connection. When a new path is available, MCTCP encloses both connection data and signaling messages in TLV fields.

Payload Multi-connection Transport (PLMT) [29] does not use any TCP options for signaling. Instead, it opens a second TCP connection dedicated to signaling on a predefined port. Then, all data or signaling messages are sent in the TCP payload in TLV fields. That method was successfully used by TLS [30] before. Since there is no need to modify the TCP stack, PLMN can be implemented in user space.

2.5 Application layer solutions

Schmidt et al. [4] also consider some application layer solutions. To implement multi-access protocols at the application layer, one can simply use the socket API and bind different sockets to different interfaces. If routing is appropriately configured, data written on different sockets will take distinct paths in the network. Moreover, the protocol can be tuned to fit the exact needs of the application.

On the downside, the application developer would need to implement himself congestion control, subflow management, security and packet scheduling. This represents a lot of work and cannot reasonably be done for each network application. It may be preferable to implement multi-access capabilities at lower layers so a large number of applications can benefit from multi-access.

Still, it may be interesting to add multi-access support to some generic or very popular application layer protocols. In those cases, implementations of these protocols may be available in popular libraries or frameworks. If the new multi-access features are included in those libraries, existing applications could be easily updated.

For instance, HTTP is by far the most popular protocol. Countless applications use HTTP because they want to be accessible in a web browser or be able to go

through NATs and firewalls. If a popular web browser or a generic HTTP implementation was able to send different requests through different interfaces, many web-based applications would benefit from resource pooling. Because of the connectionless nature of HTTP, no additional mechanism is needed. However, it does not really support vertical handover and is somehow unfair to HTTP persistent connections since multiple TCP connections will get more throughput together at bottlenecks than a single TCP connection. The latter problem is discussed in Section 3.2.

The file sharing protocol Bittorent would also be able to use several interfaces for downloading a file since it is able to retrieve different pieces of a same file from different sources using different connections [31].

RTP is a generic protocol for real-time applications developed by the IETF [32]. In most cases, RTP is operated on top of UDP but TCP is supported as well. If it were able to support multi-access to increase the available throughput, many real-time applications, such as IP telephony and video streaming could improve the quality of the streamed media. Multipath RTP (MPRTP) [33, 34] is a solution to achieve resource pooling for real-time multimedia applications by using several interfaces simultaneously.

RTP is often used along SIP for signalization. It is possible to use SIP to perform vertical handovers [35, 36] and achieve reliability pooling. However, that solution does not consider concurrent transfers on multiple interfaces. Like MPRTP, this protocol is somehow specific to real-time applications and is not suited for reliable data transfers.

2.6 Conclusion

Different protocols and mechanisms could allow mobile endpoints to take advantage of several interfaces, each one having advantages and drawbacks. However, apart from some application layer solutions, TCP-based solutions are probably the easiest and fastest to deploy. They have two key advantages:

- Multi-access is entirely supported by the endpoints. There is no need for heavy modifications in the network. Even though some NATs or firewalls may cause some problems by stripping unknown TCP options, most middleboxes are expected to see multi-access enabled TCP traffic as normal TCP traffic. Furthermore, there is no need to buy and deploy dedicated network equipment such as relays or gateways.

- All already existing TCP applications could benefit from multi-access without any modification if the operating system supports the new variant of TCP. This is a strong argument in favor of such solutions since most popular application protocols run on top of TCP. As shown by Falaki et al. [37], on smartphones, common TCP applications are responsible for most of the traffic.

Other solutions, depending on the layer at which they operate, may require more efforts in order to be deployed:

- The network layer solutions proposed by the 3GPP are specific to cellular data networks and would not allow users to access the Internet via different operators. Both mobile IP or 3GPP solutions require specific relays and/or gateways [7, 9, 10].

- A completely new protocol at the transport layer would encounter problems with NATs and other middleboxes. SCTP is a good example. This protocol is available on endpoints as it is implemented in recent versions of open-source operating systems. Nevertheless, it is still very difficult to operate SCTP because of the omnipresence of NATs that do not support it [19]. Encapsulation in UDP is often the only solution. In addition, applications have to be modified to support the new protocol.

- Though no modification of the network is necessary, application layer solutions require a lot of work on the endpoints. Indeed, all existing applications have to be modified or rewritten to benefit from multi-access. Unless it affects a very popular protocol such as HTTP or address a specific issue like real-time applications, adding multi-access support at lower layers is probably a more economical solution.

3 Multipath TCP

Multipath TCP is a set of TCP extensions defined in RFC 6824 [24] that allows a single TCP connection to send and receive data using different IP addresses simultaneously. It is necessary to present its mechanics before explaining the design of the test bed and the results of the experiments. Thus, this chapter will present important MPTCP concepts and will provide an outline of its operation. Some predictions on the impact of MPTCP on performance will be given. The existing works on the application of MPTCP to mobile devices will be reviewed at the end of the chapter.

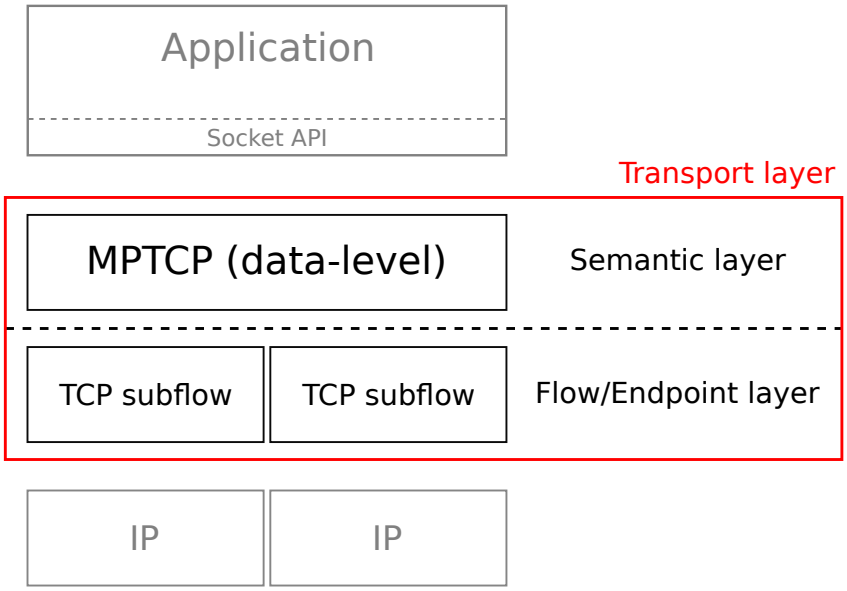

Figure 2: MPTCP protocol stack, adapted from RFC 6128

3.1 Specific concepts

Before going into the details of MPTCP operation, some concepts that are specific to this protocol need to be defined and explained. This section will follow the terminology section of RFC 6824 but will provide additional examples and considerations.

Subflows

With normal TCP, there is only one data flow per connection, between to pairs of IP addresses and ports. With MPTCP, data belonging to a same connection can be transmitted with different source and destination IP addresses and ports. Packets having the same source and destination IP addresses and ports belong to a same subflow. An MPTCP can be made of several subflows.

To MPTCP-unaware middleboxes, MPTCP subflows are like normal TCP connections since they have their own sequence number space, are opened with a three-

way handshake and terminated either by a four-way handshake or a reset packet. However, as long as there is more than one subflow, subflows only carry a fraction of the application level data. The payload is not meaningful on its own and must be reordered and combined with pieces of data carried by other subflows before being delivered to the application socket. That is why it may confuse network devices that perform packet inspection.

Connection level

The concept of connection is not the same for normal TCP and MPTCP. Indeed, with normal TCP, a connection is limited to a single address and a single port on each endpoint. With MPTCP, a connection can use several subflows and is not identified by the source and destination addresses and ports. As visible on Figure 2, which presents the high-level architecture of MPTCP, the transport layer is now divided in two parts. RFC 6182 [23] makes the difference between the application-oriented *semantic* layer and the network-oriented *flow and endpoint* layer. The semantic layer ensures the reliable data transmission between the two distant applications and the flow and endpoint layer focuses on congestion control and endpoint identification.

At the MPTCP connection level, data is reordered and ready to be delivered to the application socket. According to RFC 6824, *data-level* can be used as a synonym of *connection level*. Figure 2 shows that the connection level is an abstract layer between the transport layer and the application layer.

Paths

The path taken by packets is defined by the successive hops between routers before they arrive at the destination host. To achieve resource pooling, an important assumption is that packets belonging to different subflows can take different paths to reach their destination. This depends on routing configuration and network architecture. Paths may have links in common, they do not need to be disjoint; as long as they differ from one link, there are still some benefits from resource pooling, at least for reliability.

3.2 Operation

This section explains the different mechanisms that allow MPTCP to create, maintain and close a connection made of several subflows. It does not mention corner cases. The related drafts and RFCs [24, 38, 39] provide a more detailed functional specification.

Table 1: MPTCP options subtypes as assigned by IANA

Value	Symbol	Description
0x0	MP_CAPABLE	Multipath Capable
0x1	MP_JOIN	Join Connection
0x2	DATA_SEQUENCE_SIGNAL	Data ACK and Data Sequence Mapping
0x3	ADD_ADDR	Add Address
0x4	REMOVE_ADDR	Remove Address
0x5	MP_PRIO	Change Subflow Priority
0x6	MP_FAIL	Fallback
0x7	MP_FASTCLOSE	Fast Close
0x8-0xe	Unassigned	
0xf	Reserved for private use	

MPTCP options

In order to support multipath-specific functionalities, MPTCP uses TCP options for signaling. As specified by RFC 793 [40], a new TCP option kind is necessary. In additions, MPTCP options contain a subtype field. The MPTCP option kind and its subtypes are defined by RFC 6824 and assigned by a specific IANA registry [41]. The subtypes listed in Table 1 will be encountered in the following description of MPTCP operation.

Establishing connection

An MPTCP-enabled host opens a TCP connection by sending a `SYN` packet that contains the `MP_CAPABLE` option. If the other endpoint supports MPTCP, then the `SYN/ACK` response will contain the `MP_CAPABLE` option as well. The first host completes the three-way handshake with an `ACK` packet that also carries the `MP_CAPABLE` option. If the other endpoint does not support MPTCP, or if middleboxes on the path stripped MPTCP options, the connection falls back to normal TCP. Figure 3 illustrates a successful MPTCP connection establishment.

The hosts will also exchange keys and agree on a cryptographic algorithm during the three-way handshake. For now, the only algorithm that has been specified by RFC 6824 is `HMAC-SHA1`. The keys are also used to generate two tokens, that uniquely identifies the connection on a host. The tokens, shared keys and chosen algorithm will be used by the endpoints to identify the connection and authenticate each other when new subflows will be added.

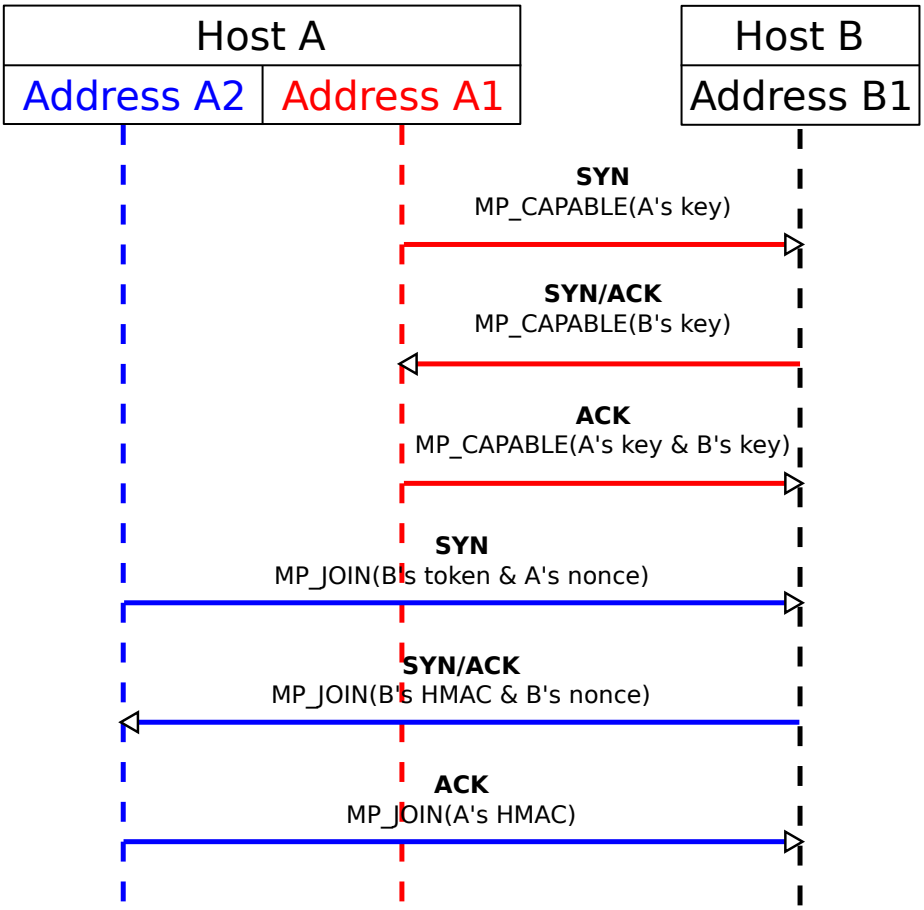

Figure 3: Initial handshake and subflow opening with MPTCP options, adapted from RFC 6824

Opening subflows

To open a new subflow, one of the endpoints has to send a new `SYN` packet with the `MP_JOIN` option. As the option includes the token of the connection, the receiver can associate the new subflow to the appropriate MPTCP connection. The `SYN` and `SYN/ACK` packets also carry an address identifier. It may be used later to refer to that address on another subflow, even if one of the hosts is behind a NAT.

The hosts will also use the method they chose during the initial connection establishment to authenticate each-other. If they use the `HMAC-SHA1` algorithm, which is the only available method for now, they will exchange randomly-generated numbers (nonces), combine them with the keys, generate a SHA1 hash from the result, and sending it back to the other endpoint. This ensures that an attacker cannot open a new subflow with one of the host unless he managed to sniff the initial handshake. The exchange process is depicted by Figure 3.

A multihomed host may not be able to open new subflows with the other endpoint if the latter is behind a NAT. The multihomed endpoint can announce the existence of supplementary interfaces on an already established subflow with the `ADD_ADDR` option. The other endpoint may then decide to open a new subflow to the announced address.

Closing connection

As the data-level is independent of the subflow level, the `FIN` and `RST` flags cannot be used to cleanly terminate an MPTCP connection. These are reserved for closing a single subflow. The four-way handshake triggered by the `FIN` flag is used to normally terminate a subflow while the `RST` flag is used to abort operations on a subflow in case of error (e.g. inconsistent data-level to subflow-level mapping). There are two ways to close an MPTCP connection at the data-level. The equivalent of the `RST` packet at the connection-level is the `MP_FASTCLOSE` option and the normal way to close the connection is the `DATA_FIN` flag of the `DATA_SEQUENCE_SIGNAL` option.

When a host wants to abort the connection, it sends `RST` packets on all subflows except one. On the remaining subflow, it sends an `ACK` packet containing the `MP_FASTCLOSE` option. The connection can be terminated when a `RST` packet from other endpoint is received on that subflow. To normally terminate a connection, a host will send a `DATA_SEQUENCE_SIGNAL` option with the `DATA_FIN` flag set to one. Each subflow has then to be terminated with a `FIN` packet, including the subflow on which the `DATA_SEQUENCE_SIGNAL` option has been sent, so as to ensure that all data has been received.

Data reordering

MPTCP subflows are like normal TCP connections. Hence, they have their own 32-bits sequence number space. Yet, as MPTCP splits the data between the different subflows, a second sequence number space is necessary to reorder the data at the connection level. An MPTCP connection has then its own 62-bits sequence number space.

This data-level sequence number does not have to be sent with every TCP packet. Instead, packets sometimes contains a mapping between the subflow-level sequence numbers and the data-level sequence numbers in the `DATA_SEQUENCE_SIGNAL` option. This method decreases overhead as there is no need to resend the mapping as long as the mapping of the last acknowledged `DATA_SEQUENCE_SIGNAL` is still valid.

RFC 6824 does not specify how the data should be divided between the different subflows at the sender side. This aspect is implementation-specific. However, it may have an important impact on performance, especially latency. The method adopted by the reference Linux implementation is outlined in Section 5.1.

Receive window

All the subflows of an MPTCP connection use the same receive window size. Nevertheless, the subflow-level acknowledgements cannot be used to get the lower bound of

the receive window. Indeed, different subflows may have different RTTs. The result is that even though all data on each subflow may be acknowledged, there may still be holes in data-level sequence numbers. Then a data-level acknowledgement is necessary for the sender to know when it can free data from its buffer. That is why the `DATA_SEQUENCE_SIGNAL` option carries a cumulative data-level acknowledgement.

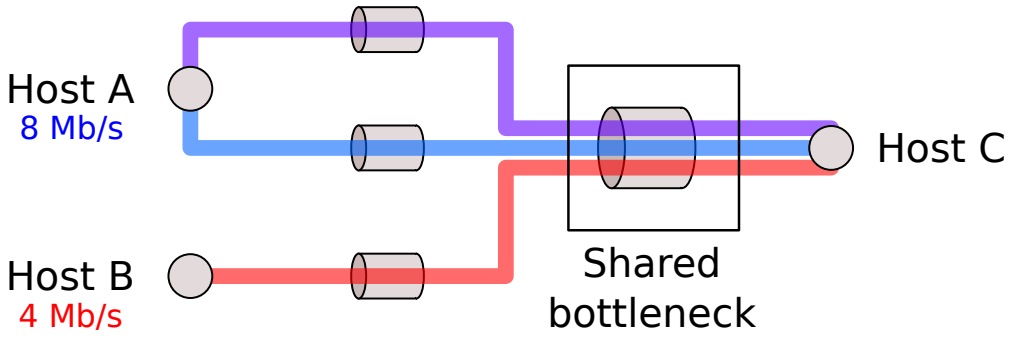

Figure 4: Multipath with per-flow congestion control is unfair to isolated flows at the bottleneck

Congestion control

Multipath TCP cannot use a normal TCP congestion control algorithms on each subflow. Per-flow congestion control mechanisms such as defined by RFC 5681 [42] only guarantees fairness between flows. Now, the problem is that MPTCP can use several subflows per connection. It is possible to encounter bottleneck links where several subflows of the same MPTCP connection compete against regular TCP flows or isolated subflows of other MPTCP connections. These so-called shared bottlenecks would probably be a common case as they may appear when paths are not disjoint, which is the case when one of the endpoints has only one interface. With normal congestion control, each of these subflow would get as much throughput as a normal TCP connection as in the scenario of Figure 4, which would be unfair to isolated TCP or MPTCP flows at the bottleneck. Therefore, congestion control on one subflow cannot be independent of other subflows.

The MPTCP reference implementation uses the *coupled* congestion control algorithm described by RFC 6356 [38]. This window-based method takes into account the congestion windows of other subflows of the same connection. Its design has been explained by Wischik et al. [43] [2]. This algorithm has been designed to comply with the three following rules.

[2] In their paper, the algorithm of RFC 6356 is called *semicoupled* instead of *coupled* since *coupled* names another, simpler, congestion control mechanism.

- An MPTCP connection should be at least as efficient, in terms of throughput, as a normal TCP connection.
- On a given path, a multipath flow should not take more resources than a normal TCP flow.
- MPTCP should move the traffic from congested paths to uncongested paths.

This algorithm is flawed, as Khalili et al. [44] prove that it may harm other TCP flows. They also suggest a better alternative as do Hassayoun et al. [45]. However, as *coupled* is the default algorithm for the available Linux implementation [25] as well as the only one specified by an RFC, this work will consider that MPTCP uses the *coupled* congestion control algorithm. The basics of this algorithm are introduced below.

In case of packet loss, coupled congestion control behaves like normal TCP for each subflow. The congestion window of the subflow is halved and the slow start algorithm is triggered. But the congestion avoidance phase is different.

Formula (1) comes from RFC 6356. It computes the increment ΔCW of the congestion window in bytes when an acknowledgement is received on the subflow i. L is the size, in bytes, of the acknowledged data; MSS_i is the maximum segment size on the path of the flow i; CW_i is the congestion window of the flow i; CW_{total} is the sum of the congestion windows of the flows that belong to the same MPTCP connection.

$$\Delta CW_i = \min\left(\frac{\alpha \cdot L \cdot MSS_i}{CW_{total}}, \frac{L \cdot MSS_i}{CW_i}\right) \tag{1}$$

We observe that the second term guarantees that the subflow will not increase its congestion window faster than a regular TCP flow. The important part of the formula is the parameter α, which is chosen such as the MPTCP connection respects the two following constraints.

- The connection should get as much throughput as would get a normal TCP connection on the best path.
- To be fair at the bottleneck, the connection should not get more throughput as would get a normal TCP connection on the best path.

$$\alpha = CW_{total} \cdot \frac{\max_i \left(\frac{CW_i}{RTT_i^2}\right)}{\left(\sum_i \frac{CW_i}{RTT_i}\right)^2} \tag{2}$$

Wischik et al. explain how to calculate α. The final expression is Formula (2) where RTT_i is the measured RTT on the subflow i. The window increase is then

lower than one MSS, which is intended. An MPTCP connection will take more time to reach maximum utilization than separate TCP flows on separate interfaces.

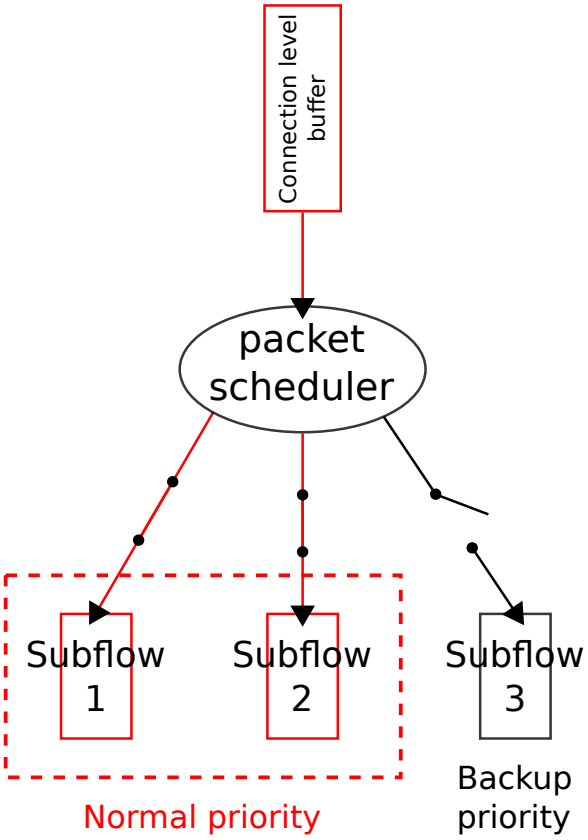

Figure 5: MPTCP subflow priority system

Subflow priority

The packet scheduler is implemented in kernel space and cannot be configured by the applications. Hence, the subflow priority system of MPTCP is an important feature. Indeed, dynamic interface selection will need a way to move traffic from one interface to another and this can only be done in user space by using the priority system. It can also be useful for applications that need some degree of responsiveness to set lower priorities to a path with a high RTT. MPTCP supports two priority levels: a normal priority and a backup priority. A subflow can be set as a *backup* subflow that will not be used to send data unless other subflows fail. The packet scheduler will then share the traffic only between the normal subflows as illustrated by Figure 5.

The priority of a subflow can be notified during the handshake, in the `MP_JOIN` option, or after the subflow has been established, using the `MP_PRIO` option. That second option allows the receiver endpoint to change the priority of a subflow dynamically. If an endpoint receives a message that sets the priority of a subflow to *backup*, it will avoid sending data on that subflow as long as other subflows are

available. However, data can still be received from backup subflows.

If an interface is temporary unavailable due to problems on lower layers it is still possible to change its priority by sending the `MP_PRIO` option on another subflow. The option will then include the address ID that refers to the source address of the unavailable subflow.

3.3 Expected behavior

Before taking into account the results of various experiments, it is interesting to look at the expected performance of MPTCP to check later if the TCP extension really met its objectives.

Reliability

By its very design, MPTCP helps to recover from link failures if several interfaces are available on one of the endpoints. As an example, Figure 1 shows that a mobile device that has two radio interfaces should be able to maintain an MPTCP connection as long as one interface is available.

Throughput

The global throughput is expected to be at least as good as the available throughput on the best path, as the congestion control mechanism is designed with this goal. At the same time, in order to ensure fairness at the bottleneck, the coupled congestion control algorithm is less aggressive than individual TCP flows. The total throughput that an MPTCP connection can achieve would be lower than the sum of the throughputs of individual TCP flows. Whischik et al. [43] explain that the connection can actually reach maximum utilization on uncongested paths.

Latency

It is difficult to estimate the impact of MPTCP on latency. The exact performance of MPTCP regarding latency depends on how the sender splits the data between the different established subflows, which is implementation-specific. At that point, it is unclear whether the observed RTT will be lower or higher.

On the one hand, as data can come from different sources, it is necessary to wait for missing packets coming from all networks. Thus, if the traffic is shared between all interfaces, the lower bound for the overall latency is the RTT on the slower path. Then it would be unsuitable to use MPTCP if the endpoint is running an application that favors responsiveness over throughput.

On the other hand, if an application generates a lot of traffic, bufferbloat must be taken into account. MPTCP could be able to share the load between the buffers of different paths and decrease latency. Indeed, the congestion control algorithm does not ensure that every path is used at its full capacity.

3.4 Application to mobile endpoints

The advantages of MPTCP on mobile devices are presented by Raiciu et al. [46]. They explain that the make-before-break capabilities of MPTCP offer smoother handover than network layer solutions such as mobile IP. The problem of energy consumption is mentioned and a simple scheduler similar to the one defined in Section 4.4 is tested. Results of simulations and indoors experiments show the benefits of MPTCP in terms of throughput and energy-efficiency.

Paasch et al. experiment vertical handover between WLAN and 3G [47]. They consider different ways to use the handover capabilities MPTCP. For instance, if the user prefers battery life over performance, WLAN and 3G will not be used concurrently. Short experiments are done in real networks and show the delay necessary for MPTCP to perform the handover. In some cases, the handover is too long due to the loss of signaling packets. The authors suggest a modification of MPTCP that solves the latter problem.

The issue of transmitting multimedia streams to mobile devices over MPTCP is explored by Diop et al. [48]. Like TCP, MPTCP ensures a reliable data transfer. But real-time applications such as video streaming may accept some packet losses and reliability mechanisms may create unnecessary overhead and delay. The authors of simulated a video streaming over MPTCP (with two interfaces used concurrently), TCP and UDP using the ns-2 simulator. The results show that MPTCP transport can deliver a better video quality than streaming over UDP. Additional QoS mechanisms to improve video streaming quality with MPTCP are explained and evaluated in the simulator.

Pluntke et al. address the problem of MPTCP power consumption on mobile devices [49]. They use MPTCP vertical handover capabilities along with path schedulers in order to improve energy-efficiency. Power consumption models for WLAN and 3G interfaces are provided. The proposed path selection algorithm is evaluated with different types of applications. Application models are generated from traffic measurements done on a smartphone.

Most of these papers consider the case of a mobile host with one 3G access and one WLAN interface. It is, indeed, the case of most smartphones nowadays. But one could wonder how MPTCP performs on a mobile endpoint that has more than two active interfaces. This could be the case, for example, of smartphones with active dual-SIM capabilities and a WLAN interface. This thesis will cover such a case since the test bed will use three 3G interfaces.

Furthermore, WLAN usually offers better performances than UMTS regarding throughput and energy consumption. Thus, interface selection may be simple as WLAN is assumed to be the best choice in most cases. On the contrary, when available interfaces are of the same kind, interface selection becomes difficult as the choice of the best interface will vary depending on network load and radio coverage.

3.5 Conclusion

This chapter presented the key elements of MPTCP design and operation. Subflow management, data reordering and flow and congestion control with MPTCP have been explained. Expectations about MPTCP performance and previous works on that experimented MPTCP on mobile devices were reviewed. Still, it is necessary to make long real-world experiments to confirm previous theoretical results before planning any deployment. Real-world measurements could reveal effects that were not predicted or included in simulations.

4 Interface selection

Interface selection is the process of choosing one network among several available interfaces. Many aspects of this issue are not specific to multi-access enabled endpoints. For instance, when a mobile device with only one WLAN interface connects to the Internet, it often has to choose among several WLAN access points. Even a host that supports concurrent transfer over different interfaces may prefer to choose only one path so as to decrease energy consumption or latency (that case is the one considered with MPTCP on mobile devices).

The user can, of course, select the interface himself, but it does not guarantee that the choice is judicious. In addition, the characteristics of the available networks may be constantly changing and the interface selection redone regularly. Thus, it is necessary to automate that process. When the mobile device supports vertical handover, the benefits of an automated and efficient algorithm for interface selection are higher since switching from one path to another does not interrupt the ongoing data transfers.

This chapter will explain the issue of interface selection and present some related works. The interface selection algorithms that were implemented for this thesis will be defined.

4.1 Objectives

The objective of the interface selection process is to select the *best* available network. Before trying to determine which interface is the best, it is necessary to define the criteria that are used to define a good or a bad network. The objective can vary depending of the definition of a good connection.

Throughput and latency are obvious characteristics that an interface selection process could try to optimize. Reliability could be also important, for instance for long bulk data transfers, and jitter may be considered by real-time applications. There are other path characteristics that are not related to performance or reliability but may be considered by an interface selection algorithm. On mobile devices, data transfer price and energy consumption would probably be very important criteria for the user.

Now, this may not be possible to find a single interface that optimizes all these criteria simultaneously. Furthermore, different applications may run at the same time on the same device and have different requirements. That is why tradeoffs are inevitable while defining what is the *best* network. Each criterion has to be combined with the others but not all of them are of equal importance.

Among methods to make decisions based on multiple parameters are the use of utility functions [50, 51] and MADM algorithms [52, 53, 54]. Mohamed and al. [55] state that giving the right weight to each parameter is still an issue and compares different methods intended to solve the problem for MADM methods. Nevertheless, for some parameters such as energy consumption or the price of the transfer, a completely automated process cannot make an objective decision.

4.2 Obtaining information

Making a good decision requires having information about the possible choices, that is to say, about the characteristics of the available interfaces. This is not a trivial problem but most of the papers cited in Section 4.1 do not address the issue. Before selecting an interface and starting a transfer, a mobile endpoint has no idea of the characteristics of the path. Here are several ways to obtain information about the available networks:

1. Predict the network characteristics from local characteristics
2. Using the interface or probing the interface before the selection
3. Retrieving information from device history or some database
4. Receiving information from the network

Method 1 is probably the simplest and the most common. The device can select an interface with only basic information about the interface, such as its type (GPRS, 3G, WLAN, Ethernet) or its signal strength. This is what most smartphones are doing. The interface selection process assumes that a WLAN connection will usually provide a better throughput than a 3G connection and uses the WLAN access points when they are available. This may change with the deployment of LTE and LTE Advanced as cellular networks will be able to provide higher throughputs than WLAN connections [56]. Hardware information can be used to predict energy consumption as proposed by Pluntke et al. [49]. But these are only rough approximations as it is impossible to predict the exact RTT or available throughput. Moreover, those methods do not work when the system has to choose among several connections of the same type with similar characteristics, such as different 3G connections with different operators.

Method 2 has the advantage to provide accurate and up-to-date information about the path. Unfortunately, it implies using, at least for short periods of time, interfaces that would not be used otherwise. By temporary moving the traffic to another interface whose characteristics are potentially very bad, the selection process could deteriorate the connection performance during the probing time.

To avoid that, one could do active measurements by sending dummy traffic to probe unused interfaces [57, 58]. This could be useful method to get information about RTT and loss rate but it would imply consuming a lot of bandwidth with useless traffic to get information about the available throughput. Olvera-Irigoyen et al. [58] suggest such a method but limit the scope its to home networks. Indeed, on a mobile endpoint connected to public networks, flooding an interface for the sole purpose of measuring the available throughput is highly questionable.

However, this is less of an issue on hosts that support concurrent transfers on multiple interfaces, e.g. MPTCP-enabled devices. In order to probe the different paths, such a device can simply spread the traffic over the available interfaces and use that real and useful traffic to get the characteristics of the different paths. Moreover, if the multipath protocol implementation automatically shares the traffic

between the different paths according to their capacity, it does not take the risk to decrease the performance of the connection during the probing time. Once the measurements are over, it can starts a selection algorithm and switch back to a single-interface mode. Raiciu et al. [46] employ this process to measure energy-efficiency with MPTCP traffic.

Method 3 collects then exploits information about the context to get an estimation of the current characteristics of an interface. In the case of mobile hosts, these characteristics depend on radio coverage and network load. The history could, for instance, include the day of the week, the time of day, the location of the device and various performance criteria. As an example, Hyeyeon Kwon and al. [59] use a history database containing information about cells and access points to perform handovers from WLAN to UMTS. Such a history may provide useful estimates when the user often open connections with his device in the same areas. Yet, if the user is frequently traveling and access the Internet in new areas, the history will be of little help.

To solve that problem, several users could share their history and create a global database. Then, the user would need to connect to a third-party server to query the database or update its local copy. This leads to method 4. That is to say, the information about the available interfaces comes from the network. Method 4 is also considered by Ericsson [60]. The network itself would directly inform the endpoint about the load on the WLAN and 3G access points and the device would exploit that information to make a decision about a potential handover (yet, such a system is limited to handover between the access points of a same operator).

4.3 Performing vertical handover

Once the device collected enough information about the interfaces and applied some method to select the one that is considered the best, it can switch to the new interface. However, that action may have undesirable effects on the ongoing connections. In the simplest case, when the host do not support vertical handover, the selection process will interrupt the ongoing connections. The handover process may otherwise create additional latency as with mobile IPv6 [61]. The handover may also have a cost in energy, which is the case with a 3G interface, as it will remain active and consumes power for a few seconds after the handover. That is why it is not always optimal to always switch to the best interface.

However, if the device supports multi-access and concurrent transfer on different interfaces, as in the case of MPTCP, the delay problems could disappear. Indeed, it is possible to keep sending data on the previous interface while beginning to send data on the new one. This make-before-break ability allows not only to maintain the active connections but also to avoid handover delay. As a consequence, interface selection can be done much more often without taking the risk of degrading the performance of the ongoing connections.

Still, it does no mean, of course, that the sender will immediately send the packets to the newly selected interface. The mobile endpoint needs to inform the sender side of the modification, it will require at least one RTT before the first

packets are received on the new interfaces. Thus, it should be assumed that the previously selected interface will remain active during at least one RTT. If it is not the case, some packets will be lost. However, a reliable protocol such as MPTCP should manage to do the handover by retransmitting the lost data.

4.4 Chosen algorithms

This section presents the different interface selection processes that have been implemented and tested for this project. When the available interfaces do not use the same technology, the selection process can rely on the fact that one of the paths (e.g. through a WLAN interface) has usually better characteristics than the other (e.g. a UMTS access). Unfortunately, in the case of this work, three 3G interfaces are considered. Thus, it is necessary to collect information about the available paths before the selection.

This thesis tests different ways to obtain information about the available networks rather than methods exploiting the available information. The objective is to outline the behavior, advantages and drawbacks of MPTCP on a mobile endpoint with a few basic interface selection methods and not tuning the decision process to achieve maximal performance or energy-efficiency.

The definition of performance can vary depending on the use cases. Nevertheless, throughput, packet loss, RTT and jitter are the basic values on which are built other performance criteria. There are other parameters that a path selection algorithm could take into account but that are out of the scope of this work. Energy consumption itself could be predicted or measured in real time to affect the path selection. The price of data transfers is also a very important criterion for the user. A selection algorithm could try to minimize costs instead of energy consumption and select the cheapest operator, switching to the other ones only in case of failure.

The objective is to make a tradeoff between the performance offered by MPTCP and energy consumption. Thus, there are two basic requirements that an interface selection algorithm has to meet.

1. It should not decrease performance compared with a normal device using a single interface.

2. In order to improve energy efficiency, it should avoid using interfaces that provide bad performance. Appendix A shows that keeping only the best interface increases energy-efficiency.

Throughput probing

The objective of this scheduler is to find and select the interface with the highest available throughput. Appendix A explains why selecting the interface with the highest bandwidth increases energy-efficiency compared with normal MPTCP. As the available bandwidth changes over time, the simplest way to get accurate information is to probe the network before making a decision. As explained in Section 4.2,

probing the available bandwidth with MPTCP is relatively safe for performance and does not require to flood the network with useless packets.

This scheduler refreshes its interface selection every ΔT seconds. This interval is split into the probing part and the path selection part. During the probing time $P < \Delta T$, MPTCP uses all interfaces simultaneously. At the end of the probing time, the path with the higher throughput is selected.

P should be large enough to let the MPTCP subflows reach their maximum throughput. But the device will also consume more energy as all interfaces are active during the probing time. Optimizing P is then determining the exact amount of time that MPTCP needs to download at full speed, which depends on the congestion control algorithm, the RTT and the network capacity. These last two parameters will unfortunately vary over time or as the device moves. As a result, an arbitrary value will be chosen.

In order to react quickly when the active path is no longer the better one, ΔT should not be too large. If ΔT is too large, then a path can remain active even though the conditions changed. The worst case being when network characteristics change just after the probing time. However, if ΔT is too small and the available throughput on one or more of the available interfaces is oscillating, then the device might constantly change its active interface to little or no benefit, or worse. Indeed, 3G interfaces still consume energy for a few seconds after the end of the transmission.

Location and history-based selection

This algorithm also tries to select the interface that provides the highest throughput but uses a database and the location of the device to obtain an estimate for the bandwidth instead of probing the interfaces. The current location is obtained via a GPS chip. The algorithm queries a database [62] that contains records of previous measurements. A typical entry of this database contains GPS coordinates, PLMN identifier and various measurements such as average throughput, latency or jitter. The scheduler selects the entries whose coordinates are close to its current location and it compares the average throughput provided by different operators. The 3G interface that uses the best operator is then selected.

Figure 6 describes the decision process. It takes into account the entries whose coordinates are within a disc centered on the current position. The initial radius of the disc is 100 meters. If there is no entries whose coordinates are close to the current position, the algorithm doubles the radius until if finds enough entries. This is not a very accurate solution since small structures may modify the quality of the radio link, especially inside buildings. Moreover, if the measurements are only a few in the area, the operator selection may be based on measurements that are too far from the current position and then are meaningless, even though there is a maximum value for the radius of the disc.

It is also assumed that a 3G connection cannot give as much throughput as WLAN or Ethernet connections. Therefore, if a WLAN or Ethernet wired connection is available, that scheduler does not query the database and automatically uses these connections instead of the 3G interfaces.

This method avoids actively probing the different networks to get some information about them. Thus, only one interface is used at once, which is better for energy consumption. Furthermore, it allows the selection process to get information on a given network even if it is currently in use. For instance, comparing the RTT on the active interface with the RTT on other interfaces is meaningless, as the traffic on the active interface may artificially increase the latency because of bufferbloat.

If such a solution were deployed, the database would need to be updated frequently as network coverage and performance may evolve as operators update base stations and network devices or if new structures modify shadowing. Moreover, such a database could only provide trends and does not guarantee that the selected network has the best characteristics at the moment. Indeed, as opposed to probing, this method does no take into account temporary modifications of the path properties, due to the random nature of user behavior. For instance, if several users in the same area uses the same scheduler with the same database, they will all use the same operator, potentially overloading the base station and missing the opportunity to share the traffic between different networks.

Best RTT

The objective of this scheduler is to make a tradeoff between power consumption and latency. The RTT is obtained by probing the available interfaces with a ping-like application. Two approaches are combined to avoid high latencies. Firstly, the scheduler frequently changes the active interface. Indeed, the active interface is more likely to be the one with the higher RTT, because of bufferbloat. When the active subflow changes regularly, the traffic is shared between the buffers of different paths, avoiding congestion, bufferbloat and decreasing latency. Secondly, when multiple other interfaces are available, the one with the lowest latency is selected.

Avoiding paths with high latency may also increase throughput. Indeed, the Mathis formula [63] gives an upper bound for the TCP throughput that increases as the RTT decreases. Moreover, a high latency may indicate that the path is congested and the available throughput potentially lower than other paths. Though, this does not ensure that the average throughput will be better, as some paths with higher available bandwidth may also have a higher latency.

4.5 Conclusion

Interface selection is a decision making problem that involves different parameters. Some of these parameters are non-technical and the user would have to define himself the choices and tradeoffs between, for instance, battery life and network performance. Many works propose MADM algorithms as a solution for interface selection. Nevertheless, most of them do not broach the problem of gathering the required information about the available networks. Thanks to its make-before-break design, MPTCP can make an extensive use of vertical handovers and interface selection without much performance drawbacks. The three interface selection algorithms that has been tested along MPTCP in this thesis have been presented.

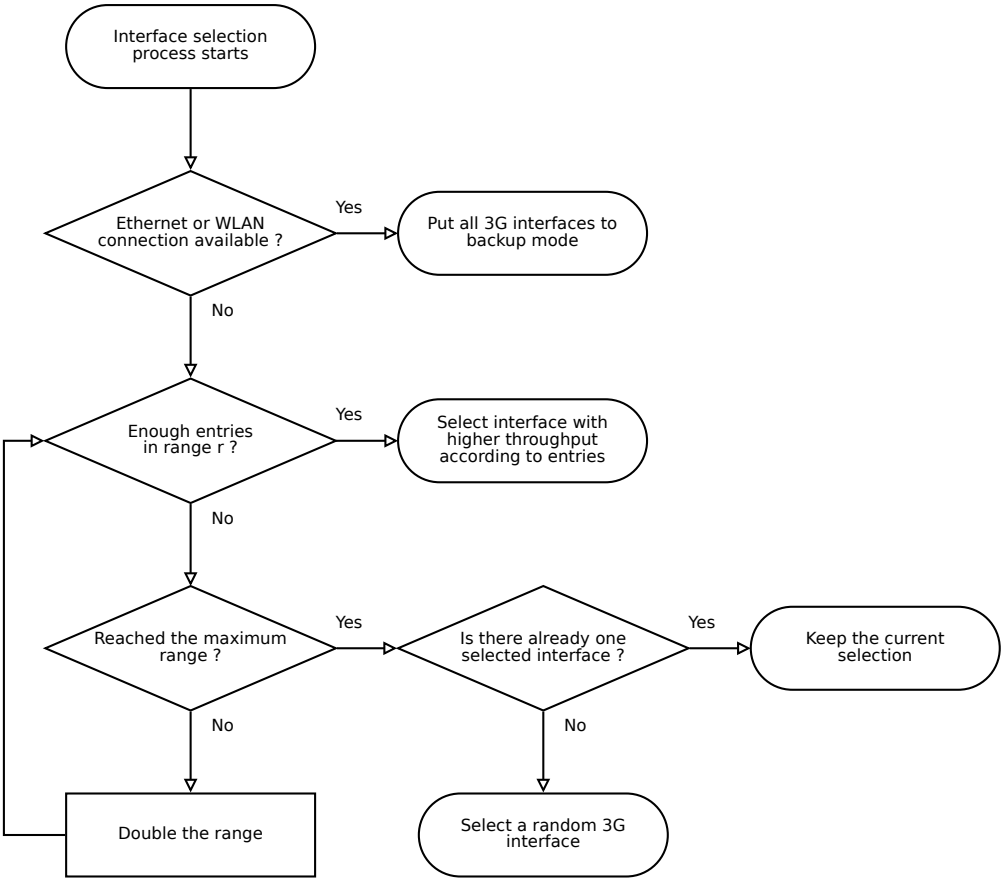

Figure 6: Location and history-based selection decision process

5 Test bed

In order to evaluate the performance of MPTCP, at test bed has been designed and implemented. The test environment had to recreate conditions that are close to those of a mobile device user interacting with some MPTCP-enabled server on the Internet. Tools for measuring performance and implementing the interface selection algorithms had to be developed.

This chapter will describe the main software and hardware components on which the test bed relies. The testbed is made of a mobile client and a server that has a public IP address. Both use the reference MPTCP Linux implementation [25, 26]. Section 5.1 gives more details about the Linux implementation. The mobile client is a laptop that accesses the Internet using three Nokia N9 smartphones and the NetworkManager configuration utility [64]. These are presented in Section 5.2. A Python script automatically configures routing. Indeed, with MPTCP, routing configuration is specific. This issue is addressed in Section 5.3. To simulate traffic and make measurements, a client and a server written in Python are running respectively on the laptop and on the server. They are presented in Section 5.4. Interface selection algorithms are implemented using a specific framework whose implementation is detailed in Section 5.5.

5.1 Linux kernel implementation

Being the most common open source operating system, Linux seemed to be a straightforward choice for both the server and the client. Above all, an implementation of MPTCP for the Linux kernel is available [25, 26]. Some details about this Linux implementation can be found in an IETF draft [39] although it is not up to date. It is necessary to know how the implementation shares the load between the different subflows and how MPTCP configuration can be modified from user space.

Packet scheduler

An important aspect of an implementation is the packet scheduling method. It defines how the traffic will be shared between different subflows at the sender side. As explained in Section 3.3, it may have an important impact on the latency of the MPTCP connection.

When there is room in the congestion windows of all subflows, this implementation sends the packet to the interface whose RTT is the lowest. By preferring the *fastest* path, that implementation ensures that an application that does not send enough data to be limited by the network bandwidth, will benefit from a low RTT.

The mentioned draft specified that flow priority cannot be changed by the `MP_PRIO` presented in Section 3.2 This option is essential to transmit the dynamic interface selection information to the sender side. Fortunately, it is now supported. At the server (sender) side, the packet scheduler takes the subflow priority into account as illustrated by Figure 5.

MPTCP configuration

The modified `ip-link` program shipped with that implementation allows the user to modify how MPTCP will use an interface. The four following options are available:

multipath on
: For each MPTCP connection, one subflow will be established and used on the interface.

multipath off
: MPTCP will not use this interface.

multipath backup
: For each MPTCP connection, one backup subflow will be established on the interface. It will only be used if other subflows fail.

multipath handover
: MPTCP will not use this interface unless it is the only available one.

If the priority of an interface is modified after the MPTCP connection establishment, an `MP_PRIO` option will be sent to all subflows that operate through this interface so as to notify the other endpoints about the priority modification.

5.2 Internet connection

In terms of hardware, the test environment will use a laptop and three mobile phones. The laptop is multihomed and establishes MPTCP connections with the server over three different cellular networks. Figure 7 illustrates the setup. The Internet connection is established by NetworkManager and a Python script automates the process.

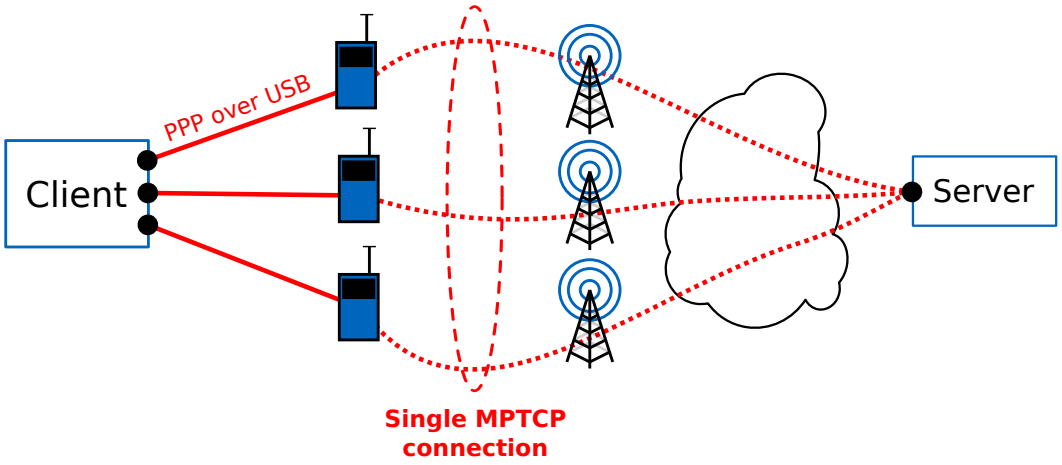

Figure 7: Connection between the multihomed client and the server over 3G networks

Mobile phones

Mobile phones are Nokia N9 running Meego Harmattan. Each phone has an Internet access provided by a specific operator. Operators are different and, in the case of an MVNO, its carrier is different from the two others. The laptop can connect to the Internet by establishing PPP connections over USB with the phones, using them as modems.

An information server is launched before the tests. It collects information from the phone, such as the GPS location, the operator's PLMN identifier or information about the selected cell. The laptop is able to establish a TCP connection with the phones to retrieve the data on demand. This is used to collect statistics about operator selection and, as the laptop has no GPS chip, is required if any interface selection program chooses the operator depending on the current location of the device.

NetworkManager

NetworkManager is a network configuration utility available with most Linux distributions. Some components of the test bench rely on that software and use the same concepts as those internally used by NetworkManager.

Two important types of objects are used by the test bed: devices and connections. A device object represents a physical device. Devices may be of different types, such as `wifi`, `gsm` or `ethernet`. When connected, a device is bound to an IP interface, an object that holds its current IP configuration. A connection object is a predefined configuration that NetworkManager can apply to an interface to establish a connection. It holds, for instance, static IP configuration, certificates and credentials for PEAP or PIN code if connecting using a SIM card. A connection has a unique identifier and is related to a specific device type.

NetworkManager Command Line Interface (`nmcli`) allows to communicate with NetworkManager using a bash shell. A D-Bus interface is also available. Thus, measurement and configurations scripts can access and manipulate NetworkManager objects.

Automatic connection

Before the routing can be configured, the three phones have to be connected to the laptop. The operating system uses the phones as if they were modems so as to connect to the Internet. The connection with the phone (or the modem) must be appropriately configured. That configuration depends on the operator. NetworkManager stores the connection settings into a connection object. Then, each device must be connected with the right connection object. Normally, the user can do the operation with the GUI. Both connections and devices can be identified using their name. Unfortunately, device names only describe the model of the phone and no unique identifier is easily available. It is a problem since the phones that will be connected simultaneously will be Nokia N9; then all three will be visible under that name in the GUI.

That is why an additional script was needed for that operation. To uniquely identify devices, the script uses their USB serial number and links them to the right NetworkManager device object using the symbolic links between serial devices and network devices that are fortunately present in the `/dev/serial/by-id/` directory. The association between the USB serial number of the device and the unique identifier of the appropriate connections is stored in a dictionary in a Python script. Using the dictionary, the script is able to connect each plugged phone to the right operator.

5.3 Routing

If MPTCP is able to give different source or destination addresses to different subflows, it does not ensure that the packets belonging these distinct subflows will take distinct paths. It only depends on routing. In order to effectively use separate paths, the multihomed host has to use a separate interface and gateway for each subflow. Otherwise, the different subflows will not only probably take the same path but may also be blocked because of ingress filtering. A generic MPTCP-compatible routing process is described in an IETF draft [65]. Yet, in the case of Linux, there is no need to fundamentally change the routing process.

The reference Linux implementation does not modify the default routing configuration and routing tables need to be manually modified to support MPTCP. Indeed, the Linux kernel assumes that the host will use only one default gateway and one default interface. All packets, regardless of their source address, are sent through the same interface to the same router. In order to send the packets through the appropriate interface and to the appropriate gateway, routing must take the source address into account.

This can be achieved by using Linux routing policies. Before routing a packet, the kernel will first look into the policy database and, only then, choose the appropriate routing table. Linux routing policies can be modified to use a specific routing table depending on the source address. There should be one rule in the policy database and one associated routing table for the source address of each interface supported by multipath TCP.

As an example, the routing policy database of an MPTCP-enabled host that has three interfaces `eth0`, `wlan0` and `ppp0` should be similar to Listing 1. The `local`, `main` and `default` tables are the three default routing tables used by the Linux kernel. The `local` routing table is automatically generated from other routing tables by the kernel. The `main` and `default` tables can be modified by the user. However, these are generic routing tables and it would not be clever to reserve them to a specific source address. Instead, new routing tables are created: these are tables 1, 2 and 3. The contents of these tables should be similar to Listing 2, Listing 3 and Listing 4. The new routing tables contain two entries: a default route that uses the appropriate device and the appropriate gateway, an entry for destination hosts that are in the same local network.

Listing 1: Routing policy database

```
0:  from all lookup local
32763:  from 192.168.1.2 lookup 1
32764:  from 192.168.2.2 lookup 2
32765:  from 192.168.3.2 lookup 3
32766:  from all lookup main
32767:  from all lookup default
```

Listing 2: Routing table 1

```
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0  scope link
```

Listing 3: Routing table 2

```
default via 192.168.2.1 dev wlan0
192.168.2.0/24 dev wlan0  scope link
```

Listing 4: Routing table 3

```
default via 192.168.3.1 dev ppp0
192.168.3.0/24 dev ppp0  scope link
```

Manual configuration for MPTCP

The `ip-rule` command is used to modify the source-based routing policies and the `ip-route` command is used to create new routing tables and add entries. Listing 5 shows the configuration of a new interface on device `eth0`. The first two lines create the entry of the new routing table and the third one adds the corresponding routing policy. The configuration requires the device name, the IP address of the interface, the IP address of the gateway and the address of the network.

Listing 5: Configuring routing for a new interface

```
# ip route add 192.168.1.0/24 dev eth1 scope link table 1
# ip route add default via 192.168.1.1 dev eth0 table 1
# ip rule add from 192.168.1.2 table 1
```

This must be done whenever a new interface is activated or if its IP address changes. Moreover, if the device is unplugged or the interface disabled, the configuration must be removed with the appropriate commands to delete the routing policy rule and flush the associated routing table. Hence, doing the configuration manually takes a lot of time if devices are constantly plugged and unplugged, which may be the case during some tests.

Automatic configuration

As some tests will require to often plug or unplug devices, the first new necessary component of the test bench is a tool that automatically overrides the default routing configuration by MPTCP-ready routing tables. Such a tool would need to be executed whenever the state of an interface changes (such as a disconnection or a new DHCP lease). NetworkManager detects such events and executes the scripts present in the `dispatcher.d` directory when they happen. Furthermore, NetworkManager passes information to such script as environment variables. This includes device name, IP addresses of the interface, network prefix length, gateway, connection identifier and the type of event that triggered the script. All the information required for routing configuration is thus available.

A Python script parses the information provided by NetworkManager and automatically adds the right routing policy and creates a new routing table when an interface is connected. When the interface is disabled, the configuration is automatically removed. This script supports devices of all kinds as long as they have an IPv4 address. It could easily be extended to support IPv6.

Associations between devices and routing tables are kept in an SQL database. This method has two advantages. Firstly, it allows the tool to directly know which routing table must be removed when an interface is disconnected, avoiding calling the IP route command variants and parsing their results to get that information. Secondly, as the database only contains routing tables generated by the script, it avoids accidental deletions of manually-defined routing tables.

5.4 Performance measurements

The test bed had to include components that measure the performance of the network connection in order to evaluate MPTCP and the different interface selection algorithms. It can be evaluated using some of the following criteria: connection availability, round-trip time, jitter, throughput and energy consumption.

Availability, latency and throughput can easily be obtained by establishing TCP connections between the laptop and an MPTCP-enabled server. Unfortunately, measuring energy consumption is quite difficult as the test bench is made of four different devices, each one having its own battery and activity. For instance, communication between the phones and the laptop over the USB cables may also increase consumption, as well as various the processes running on the laptop. Moreover, the laptop is charging the phones through the USB interface. As energy consumption measurements may not be meaningful, the scripts focuses on the other criteria.

In order to measure throughput, a simple Python TCP download server sends dummy data on request of the client. The client can request bulk transfers of various sizes, including infinity.

A TCP echo server allows to get RTT and availability. It sends a new request only when it got the response to the previous one, so the impact of latency measurements on throughput measurement and their contribution to bufferbloat are as small as possible. If the latency reaches a threshold of several seconds, the connection is

considered lost and the client has to reconnect.

The MPTCP-enabled server keeps track of the TCP connections with the download server by recording `tcpdump` captures during the measurements. Time, throughput, latency, availability and the currently active interface are recorded in `.csv` format on the client side. A Python script that parses the CSV data and generates statistics is available but plots can also be generated directly from the output files.

5.5 Interface selection

The test bed has to support interface selection algorithms. It should be easy to implement and test new path schedulers. This section explains how the test bed supports the interface selection and presents the implemented interface selection algorithms.

Framework

A Python framework was written in order to facilitate the implementation and the evaluation of new interface selection decision processes. Such a framework should facilitate the three following steps of the decision process.

1. Gather information about the interfaces.
2. Run some algorithm at precise intervals.
3. Change the priority of the interfaces (then of the subflows).

The first aim of this framework is then to gather all available information about a connected device and make it easily accessible in the path selection code. A specific module, *NetworkManager utilies*, contains different functions and classes that handle interfacing with NetworkManager but also with kernel network input/output counters and the remote information server on the phones. Thus, it can gather the necessary information about the interfaces. Some classes are close to their NetworkManager counterparts since the framework heavily rely on `nmcli`. Figure 8 is a simplified version of the class diagram of the implementation of these utilities.

The code that communicates with the phones is in the `Phone` class. Its instances represent connected Nokia N9 or N950 devices. These can be use to get information about a phone, its IP interface and the associated NetworkManager connection. Data is collected from different sources, such as NetworkManager, the Meego information server running on the phone or from the kernel network I/O counters.

The `Device` class is more generic. It can be any kind of device such as an Ethernet or WLAN interface. It provides methods to connect a device, change its multipath settings or get statistics about the interface. The `Connection` class is a simple interface with NetworkManager connection object. Its functionalities are limited but it could be extended to provide some sort of advanced automatic configuration.

The `SelectorAlgorithm` class provides a structure to quickly implement a new path selection decision process. To do so, one can subclass it, reimplement its

different methods and change the default value of its attributes. The scheduling algorithm can be chosen before starting the tests. Most other components of the test scripts are independent of the path selection process.

It could be interesting for the scheduler to know the RTT on a specific interface. Unfortunately, TCP cannot be used to get that information, since, as we use MPTCP, a single TCP socket may use different subflows at once. An MPTCP-aware socket API considered by RFC 6897 [66] would allow application developers to limit a TCP connection to a given interface. However, such an API is not available at the moment. That is why the test bed uses UDP to get the latency on different paths. Through a set of UDP sockets, each one bound to a specific interface, the laptop communicates with an UDP echo server located on the MPTCP-enabled server, retrieving the latency on the different paths. This information may not be useful for all decision processes, thus, this feature is optional and UDP packets are sent only if the chose path selection algorithm needs RTT measurements.

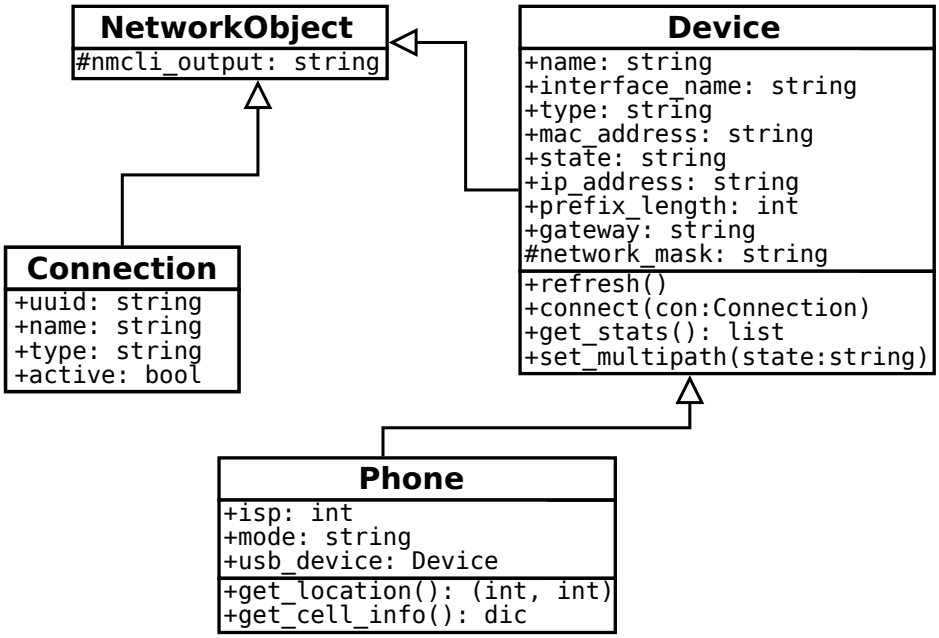

Figure 8: Class diagram of the *NetworkManager utilities* module

Simple interface selection

Some very simple network selection strategies can be considered edge cases of path schedulers. Strictly speaking, these are not schedulers. However, from the test bed point of view, these selection processes are implemented as *selection algorithms*.

A mobile device can use only one interface as long as it is available and use the others in case of failure. It performs as least as well as a single interface, fulfilling the first requirement mentioned in Section 4.4, and does not use all interfaces simultaneously, fulfilling the second requirement. It does not exploit the ability of MPTCP to move the traffic from one interface to another. However, it benefits from the pooling of reliability between available networks, improving robustness.

Another simple strategy could be to randomly choose a new path at regular intervals. It may seem pointless but if the period is short enough, it helps to prevent bufferbloat. Indeed, the data would be shared between the buffers of the available networks, resulting in a shorter RTT.

A scheduler could also select the best interface depending on its type. For instance, a wireless LAN is more likely to perform better than a UMTS access. So the scheduler can set the WLAN interface as the default path whenever it is available. Yet, this is not always true as shadowing or temporary congestion may change the situation. Moreover, if all available interfaces use the same technology, as in the case of this work, this strategy is useless.

There is also a theoretical edge case, a scheduler that could know in advance which interface will be the most efficient. Such a scheduler is called an oracle. This scheduler could only exist in virtual test environment, where network characteristics and user behavior are deterministic. In such a case, the oracle can be considered as the optimal scheduler and provide an optimal boundary with which other algorithms can be compared. As this thesis aimed to do real-world experiments, this algorithm was not implemented in any simulation environment.

5.6 Conclusion

This chapter detailed the different components of the test bed. The software and hardware bases on which the test bed has been built were presented. A solution for the generic issue of routing configuration with MPTCP was proposed. The interface selection framework designed to allow experiments to implement and test interface selection algorithms was described. The experiments done with the test bed and their results are presented in the next chapter.

6 Results

This chapter will present the experiments and analyze their results. Section 6.1 will present the context of the experiments. Section 6.2 will provide an overview of the results. The following sections will focus on specific configurations and algorithms. For each of them, the results will be presented and explained.

6.1 Experiments

Measurements were done in two different contexts: static indoor experiments and mobile outdoor experiments. They respond to different needs and have specific advantages and drawbacks.

Indoor experiments

Indoors experiments have the advantage to provide a relatively stable environment that permits to compare the performance of different configurations and path selection algorithms. Experiment conditions are easily reproducible as the device does not move and the radio environment remains the same. This does not ensure that the network characteristics will not vary. That is why the performance of the connection through each operator was checked before each measurement in order to ensure the conditions of the experiments were the same for each tested configuration.

As the network characteristics remain more or less stable, there is no need to record measurements over long periods of time. Hence, these experiments were quite short, typically a few minutes. It also allows to quickly confirm that a selection algorithm or a specific configuration behaves as indented before testing it with longer mobile experiments.

Outdoor experiments

In order to observe the behavior of MPTCP and the different path selection algorithms in a real mobile environment and with fast-changing network characteristics, tests were made in public transportation between Helsinki and Espoo. These are longer measurements that take into account variations in network characteristics.

These experiments were done in a bus between Otaniemi in Espoo and Kannelmäki in Helsinki. The total distance is about 11 kilometers and the measurements last 40 minutes. That itinerary includes different phases. Depending on these phases, the mobile node was moving slowly, fast or was immobile.

Measurements can be done in two directions at different hours on different days. The link quality may vary as the network load changes depending on the hour of the day or the day of the week due to general trends in user behavior. That is why measurements were made four times for a given configuration, on different days and at different hours. In order to measure the throughput during the entire journey, the client constantly downloads dummy data from the server.

Test bed configuration

The test bes was used to do experiments with different configurations. The simplest was a single phone connected to the laptop. This is the basic case that does not use the capabilities of MPTCP. The mobile device will open a single TCP flow with the download server using the cellular network of one operator. The TCP packets still contain the MPTCP options and an MPTCP connection will be established during the initial handshake. However, as only one interface is available at each end, only one TCP flow will be opened. In that case MPTCP behaves like TCP. The overhead resulting from MPTCP options will be considered negligible. Before the experiment, one of the three operators will be selected: DNA, Elisa or Tele Finland [3]. It is expected that, due to different cellular network coverages, the results differ depending on the chosen operator. Thus, measurements have to be done for each operator and the average performance will be a reference against which other configurations can be compared.

The other configuration uses the three phones at the same time. The laptop will open three MPTCP subflows. Depending on the interface selection process, they will be used simultaneously or sequentially. UDP connections can also be used to probe the path through the different interfaces.

The interval between two executions of the algorithm is 10 seconds in the case of the RTT-based algorithm and the location and history based selection. For throughput probing, the interval between two selections is one minute. The problem is that the probing time should be long enough to let TCP reach its maximum throughput on every interface but still negligible in comparison with the non-probing time. Before the measurements, it is not possible to know the optimal probing time. A quick indoor experiment showed that at least 5.5 seconds were necessary for TCP to reach the maximum throughput on all interfaces. The probing time for longer experiments was arbitrarily set to 10 seconds and the non-probing time to 50 seconds.

6.2 Results overview

Table 2 compares the performance of the different configurations. Results are expressed as a fraction of the value obtained by the average single-path connection in the same conditions. As expected, concurrent multipath provides the highest throughput for both indoor and outdoor measurements. Interface selection algorithms obtain more throughput than the average single-path connection but not as much as MPTCP, which is normal. Concurrent multipath and all interface selection algorithm improve latency.

Figure 9 shows the non-relative results for different configurations in the case of outdoor experiments. These are average values of several measurements of approximately 40 minutes. The conditions of outdoor experiments are unfortunately not exactly reproducible. As a result, it is difficult to compare the different configurations in the case of outdoor measurements. However, obvious trends can be commented and explained.

[3] Tele Finland is an MVNO that operates on the network of Sonera.

Table 2: Comparison of throughput and latency for different configurations

Test bed configuration	Throughput	Latency
Indoor experiments		
1 interface, average	100%	100%
1 interface, DNA	80%	67%
1 interface, Elisa	65%	159%
1 interface, Sonera	155%	75%
3 interfaces, concurrent multipath	247%	13%
3 interfaces, Throughput probing + selection	182%	35%
3 interfaces, RTT probing + selection	128%	23%
Outdoor experiments		
1 interface, average	100%	100%
1 interface, DNA	113%	111%
1 interface, Elisa	81%	78%
1 interface, Sonera	105%	113%
3 interfaces, concurrent multipath	249%	24%
3 interfaces, Throughput probing + selection	119%	76%
3 interfaces, Database query (throughput) + selection	109%	81%
3 interfaces, RTT probing + selection	100%	20%
3 interfaces, Random interface selection	97%	22%

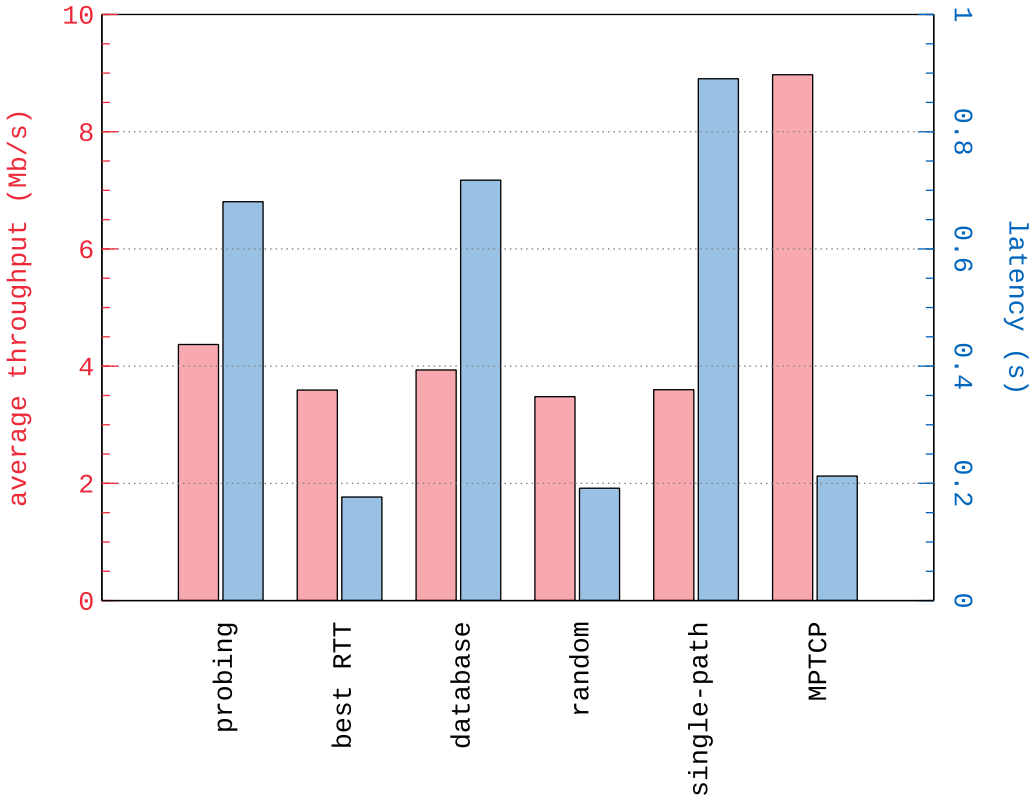

Figure 9: Outdoor experiments: average throughput and latency for different path selection methods

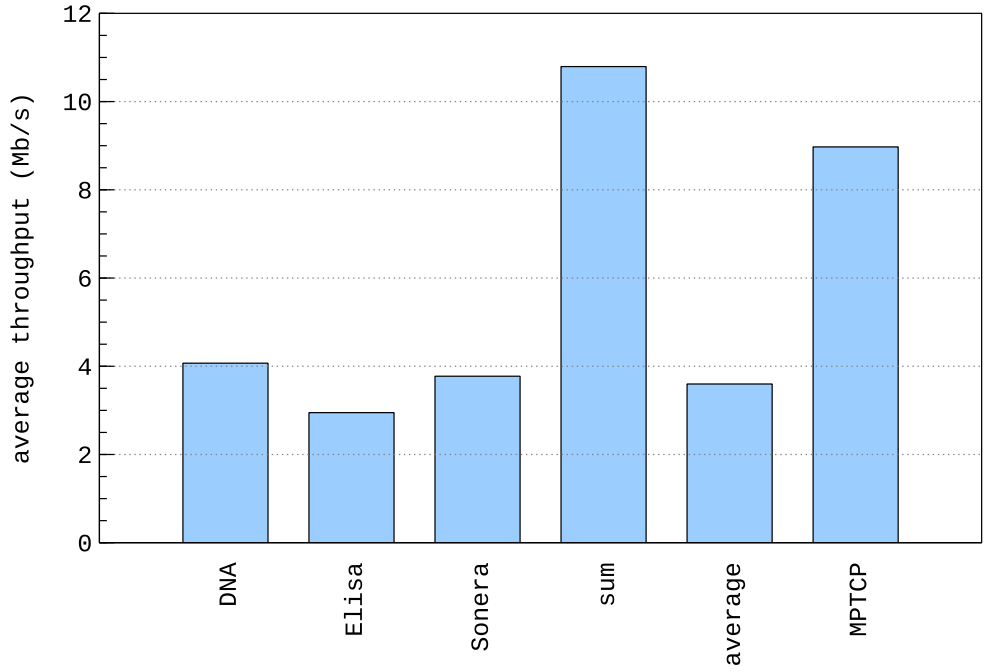

Figure 10: Outdoor experiments: MPTCP throughput compared with normal TCP connections

6.3 Concurrent multipath

This configuration measures the performance of multipath TCP using the three operators simultaneously. The three phones are connected to the laptop and all interfaces are active. There is no interface selection process going on. This configuration was expected to provide the highest throughput. Aside from possible power consumption problems, Multipath TCP with three active interfaces improves performance. It has beneficial effects on both throughput and latency.

Throughput

Figure 10 shows that using MPTCP considerably increases throughput. However, it seems that MPTCP does not manage to reach full throughput utilization on the three interfaces. If it were the case, the throughput obtained by MPTCP would have been close to the sum of the available throughputs of the three interfaces, but results on Figure 10 show that it is not the case. The available throughput with concurrent multipath is only 83% of the sum of the average measured single-path throughput.

There are at least two possible explanations for this apparent underutilization. The first one is the presence of a bottleneck shared by the different subflows, the second one comes from the coupled congestion control algorithm.

When MPTCP concurrently uses the three interfaces, the laptop opens three subflows with the remote server and starts receiving data through them. Each

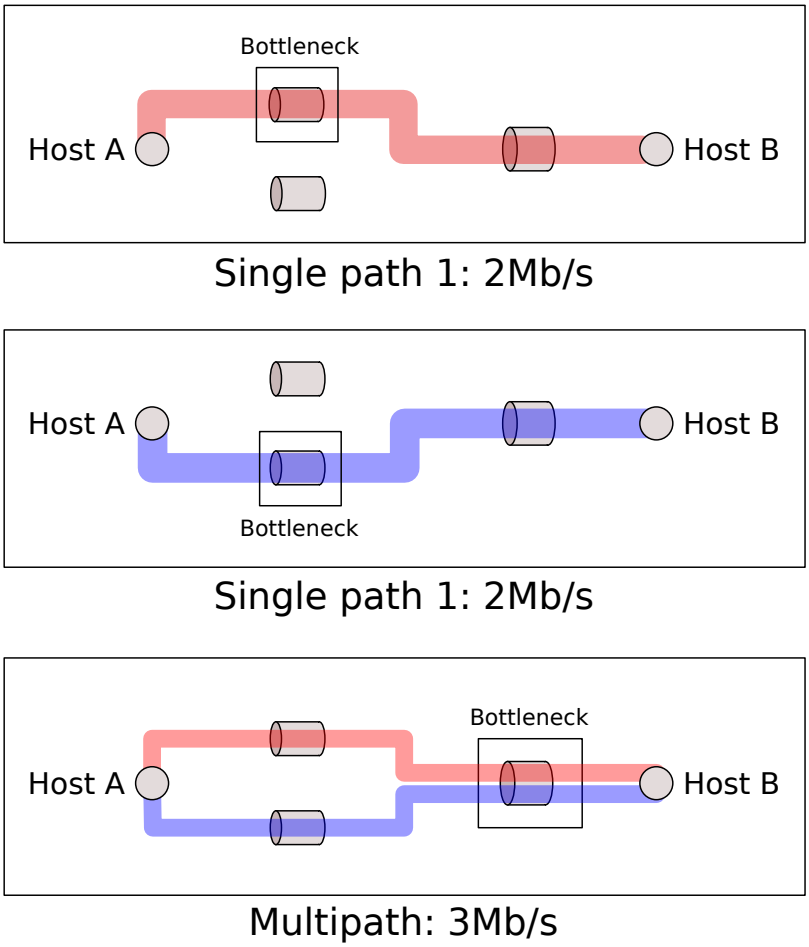

Figure 11: Possible effect of a shared bottleneck on total throughput

subflow will go through a different interface then its path will be different from other subflows. However, different paths are not always disjoint and the paths of the subflows may share one or more common links. If this link is a bottleneck, the total throughput will be lower than the sum of single-path throughputs. Figure 11 illustrates such a case. Since the MPTCP-enabled server of the test bed is not multihomed, it is certain that the different paths share, at least, one common link.

The coupled congestion control algorithm may explain why MPTCP cannot get as much throughput as three separate TCP connections. As explained in Section 3.2, the coupled congestion control only ensures that MPTCP gets as much throughput as TCP would get on the best path. It is, of course, possible to get more throughput but it is not guaranteed and it takes some time. The congestion window increase of MPTCP subflows is slower than normal TCP flows. Figure 12 shows that it takes more time for MPTCP to reach the maximum throughput than three individual TCP flows. This behavior is normal and necessary to ensure fairness at shared bottlenecks but it may explain bandwidth underutilization when network character-

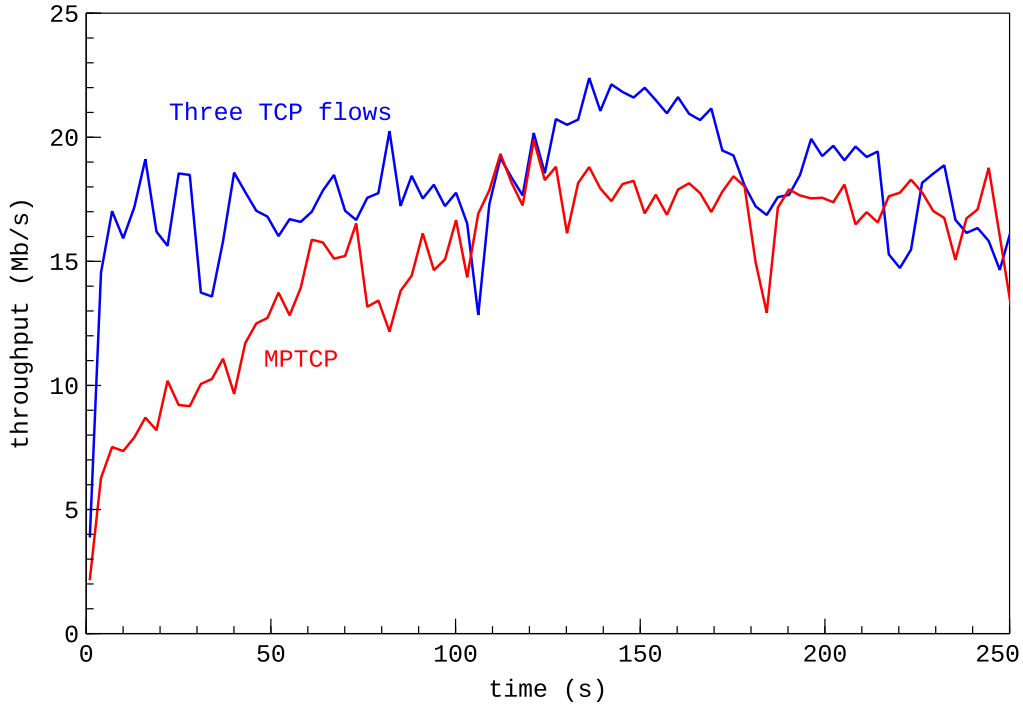

Figure 12: Indoor experiment: MPTCP throughput compared with the sum of normal TCP connections

istics are frequently changing, as shown on Figure 10 in the case of outdoor mobile experiments. However, it means that MPTCP is not suited for short transfers. Indeed, the connection will not have time to reach a high throughput and the different interfaces will be activated and consume energy for little or no benefit.

Latency

Figure 9 shows that using MPTCP with three active interfaces simultaneously decreases latency compared with normal TCP. This is not a trivial result. As explained in Section 3.3, MPTCP could have been limited by the path with the highest RTT thus would have increased the latency of the connection. Instead, using several interfaces simultaneously seems to improve latency. This could be explained by the packet scheduling policy of the reference Linux implementation or by a lower impact of bufferbloat on a multipath connection.

The packet scheduler of the Linux implementation described in Section 5.1 sends the packets on the interface with the lowest RTT when it is possible. It means that an application that is not limited by the bandwidth, such as the TCP ping client used to record the latency on the client side, will not use the subflows with high RTTs. Even better, the scheduler automatically optimizes responsiveness. An application that is limited by the bandwidth, such as the download client used to record the throughput on the client side, will use the available throughput of the three subflows and get the requested data faster at the connection level, improving its responsiveness as well.

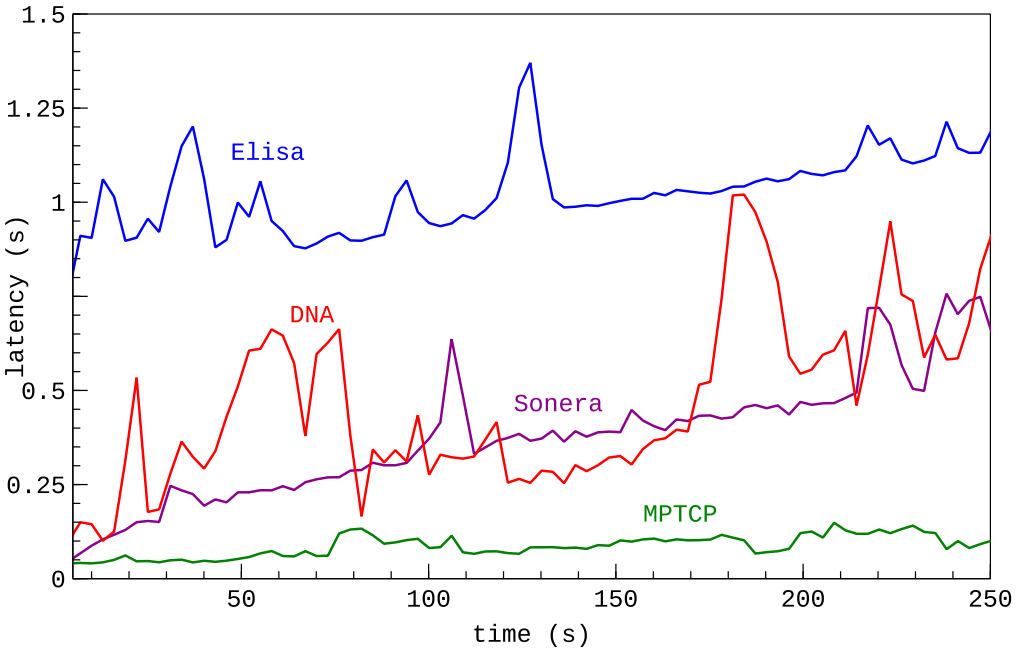

Figure 13: Indoor experiment: MPTCP latency compared with normal TCP connections

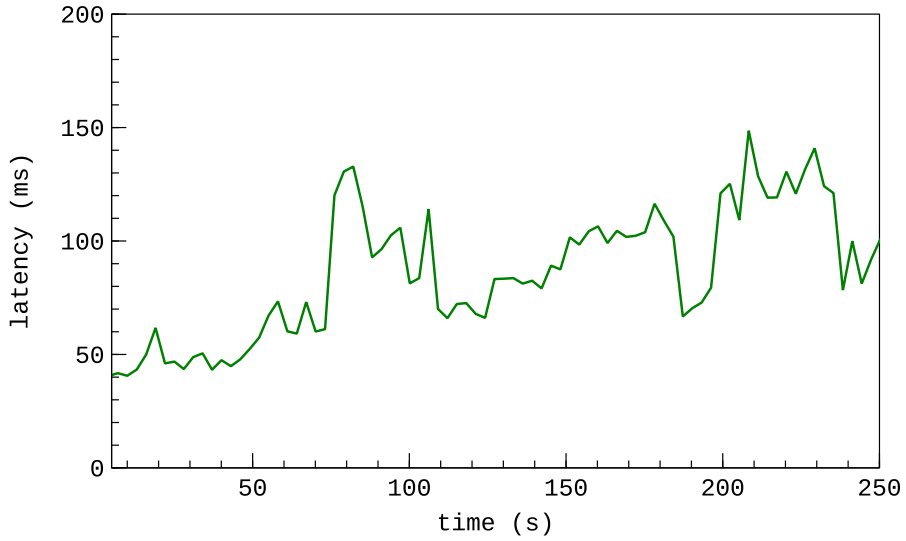

Figure 14: Indoor experiment: MPTCP latency slowly increases

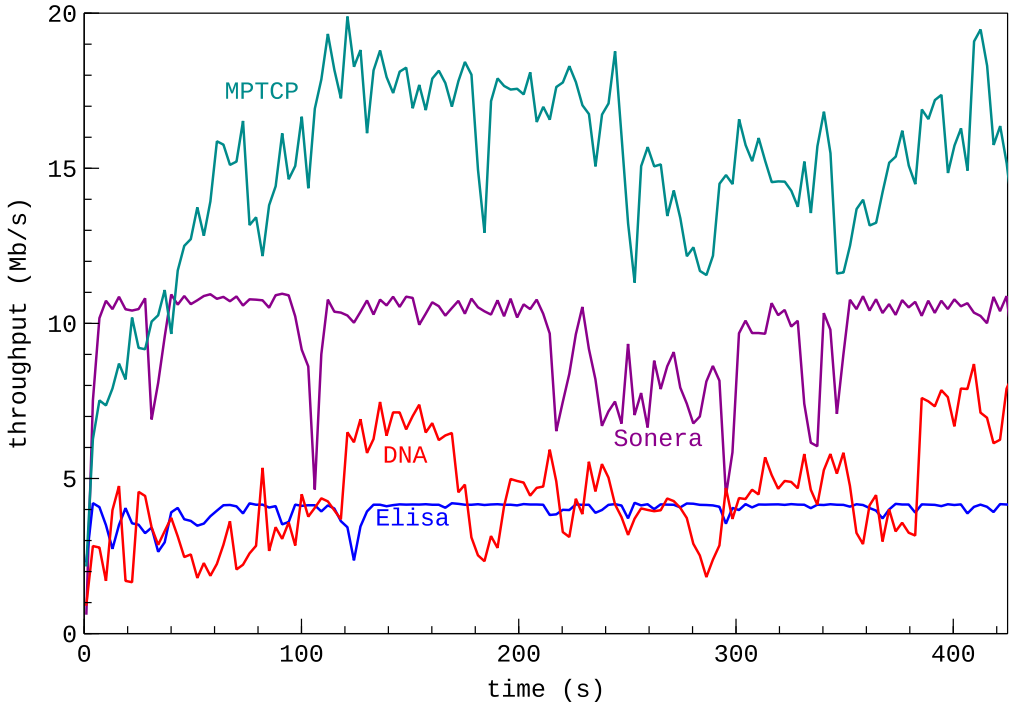

Figure 15: Indoor experiment: MPTCP throughput compared with normal TCP connections

The previous section showed that MPTCP does not reach full utilization. On an underutilized path, router queues are shorter and bufferbloat is less important. Hence, an MPTCP connection may have less problems with bufferbloat. This is visible on Figure 13, which shows the evolution of latency for normal TCP connections and an MPTCP connection during an indoor experiment (the client downloads as much data as possible from the MPTCP-enabled server). The latency of the normal TCP connections increases. Figure 14 shows that the latency has a tendency towards increase for MPTCP as well but it is much slower. The MPTCP connection that simultaneously uses the same three interfaces has a far better latency and a higher throughput (Figure 15 shows the evolution of the throughput for the same experiment).

6.4 Throughput probing

Figure 9 shows that the *throughput probing* algorithm provides better results than normal single-path TCP connections. This result was expected. The behavior of the algorithm is observable on Figure 16. It uses MPTCP on all interfaces during a probing time of 10 seconds then select the interface with the best throughput for the next 50 seconds. In that indoor experiment, the interface connected to the network of Sonera provides the higher throughput. Thus, after probing phases, the interface selection algorithm selects the Sonera network.

The connection gets extra throughput during the probing phases. It is normal since the traffic used for probing is normal application traffic sent over the three

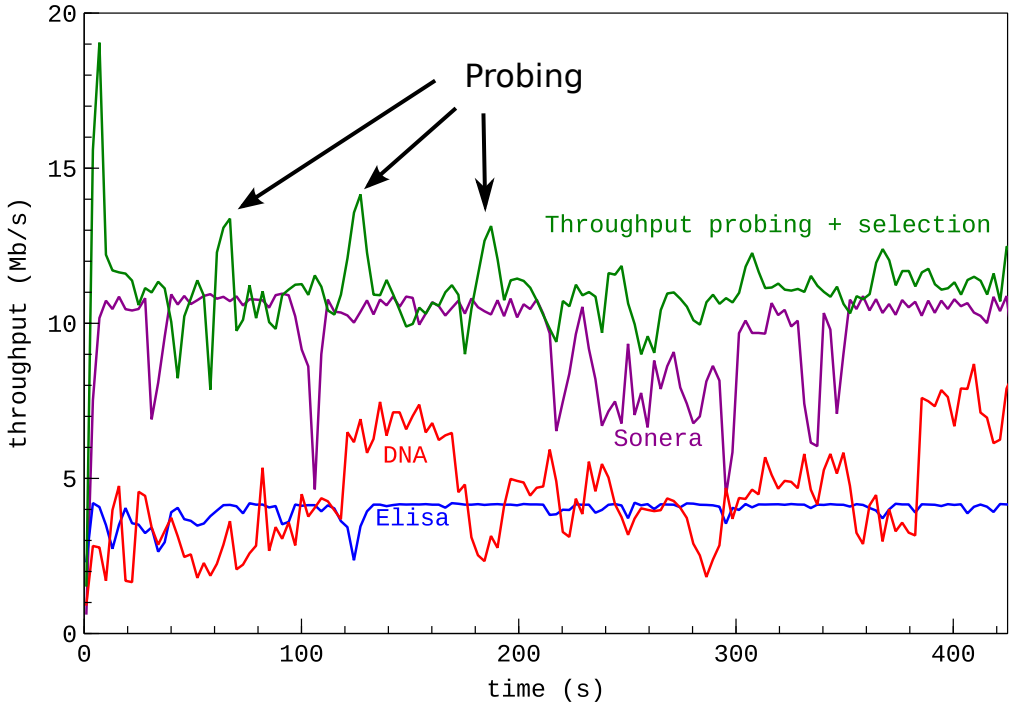

Figure 16: Indoor experiment: selection based on throughput probing compared with normal TCP flows

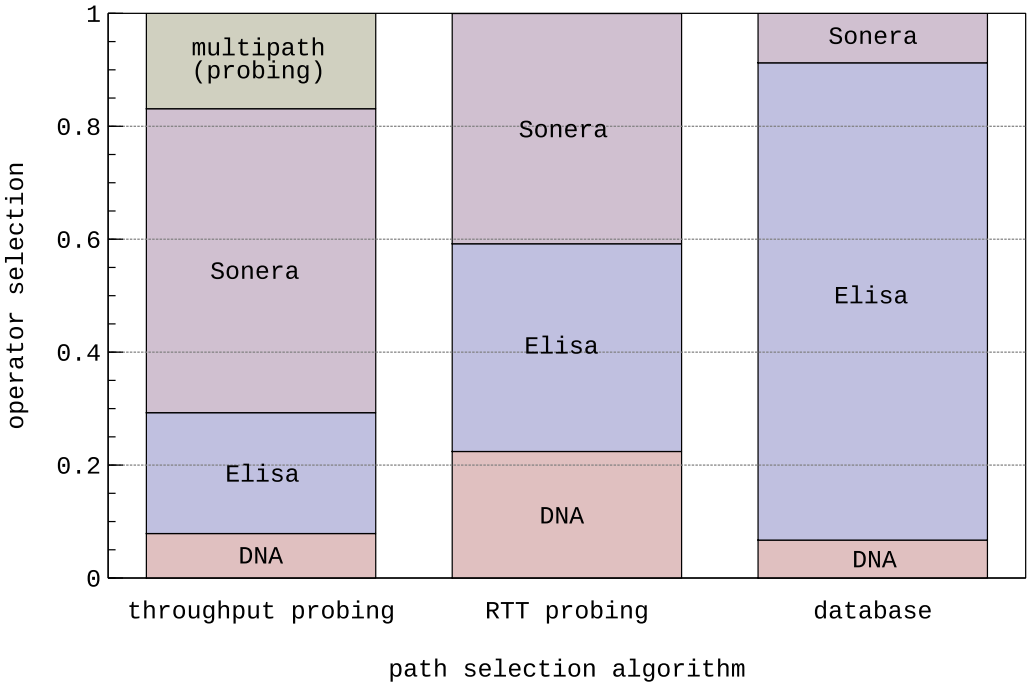

Figure 17: Outdoor experiments: operator selection ratio for different algorithms

interfaces using MPTCP. Hence, probing times increase the average throughput obtained at the end of the experiment. The probing time represents one sixth of the total time. As MPTCP on three interfaces provides 250% of the average available throughput on a single interface, the probing time alone could increase the throughput up to 125% of the normal single-path throughput.

During indoor experiments, that path scheduler provided 182% of the average single-path throughput to the connection. However, during outdoor experiments, it was only 119%. It is still better than the best throughput measured for single-path experiments but not as high as expected. This performance problem may be explained by a lack of responsiveness when network characteristics are quickly and frequently changing. Indeed, after the probing phase, the same operator is selected for 50 seconds. As a result, an interface may be selected during a long period of time during which it is no more the best interface.

6.5 Location and history-based selection

The use of a database to replace probing before selecting the operator that provides the best throughput allows to refresh the selection more frequently. In addition, it spares the energy required for probing on the other interfaces. Figure 9 shows that it provides better throughput and latency than the average single-path connection. However, it is not as efficient as its probing-based counterpart. Moreover, the improvement is slight and may be due to a difference in the conditions of the experiments.

Figure 17 shows that the decisions based on the database entries greatly differ from those based on probing. It means that the information stored in the database does not accurately describe the characteristics of the available networks at the moment of the decision process. In some areas, the database contains less than one measurement per 200 meters squared. In such cases entries are too sparse to get meaningful information about all three different operators. Moreover, network characteristics may change depending of the hour of the day and that path scheduler does not take the hour of the measurement into account. If more measurements were available in some areas, the location and history-based selection may show better results.

6.6 Best RTT and random selection

The three phones are connected to the laptop but only one is active at a time. The path selector chooses one random phone every 10 seconds. It may seem of limited interest but it allows to see if switching from one interface to another has unwanted of interesting effects on the connection. Moreover, as explained in Section 5.5, frequently changing the active interface may help to decrease bufferbloat and latency.

Outdoor experiments show that the *best RTT* algorithm is the selection process that provides the best latency. It is approximately five times lower than the average latency measured for single-path configurations. It seems to have little impact on throughput during outdoor experiments. MPTCP has a better latency during indoor experiments but it uses three interfaces concurrently. Overall, that algorithm met its objective.

Figure 18 is the evolution of latency over four minutes of measurements. The RTT based selection is triggered every 10 seconds and avoid the slow increase of the latency visible with normal TCP connections. The periodical peaks in the latency are due to the selection of an interface whose RTT is low when there is no traffic. As soon as the MPTCP priority system moves the traffic to that interface, the latency rapidly increases.

Figure 9 also shows that the random selection algorithm, which changes the active interface every 10 seconds, gets a low latency as well. Choosing a random interface is apparently enough to solve most of the problem as the difference between the selection based on the best RTT and the random selection is negligible.

Thus, bufferbloat may be the main reason behind high latency. Frequently changing the active interface can solve the problem of bufferbloat since it shares the load between the buffers on different paths and avoids the slow increase of latency observed on Figure 13.

One may want to give up the slightly lower latency of the RTT-based selection and get rid of the RTT probing over UDP. Indeed, probing consumes a lot more energy since it prevents the 3G interfaces from being completely turned off.

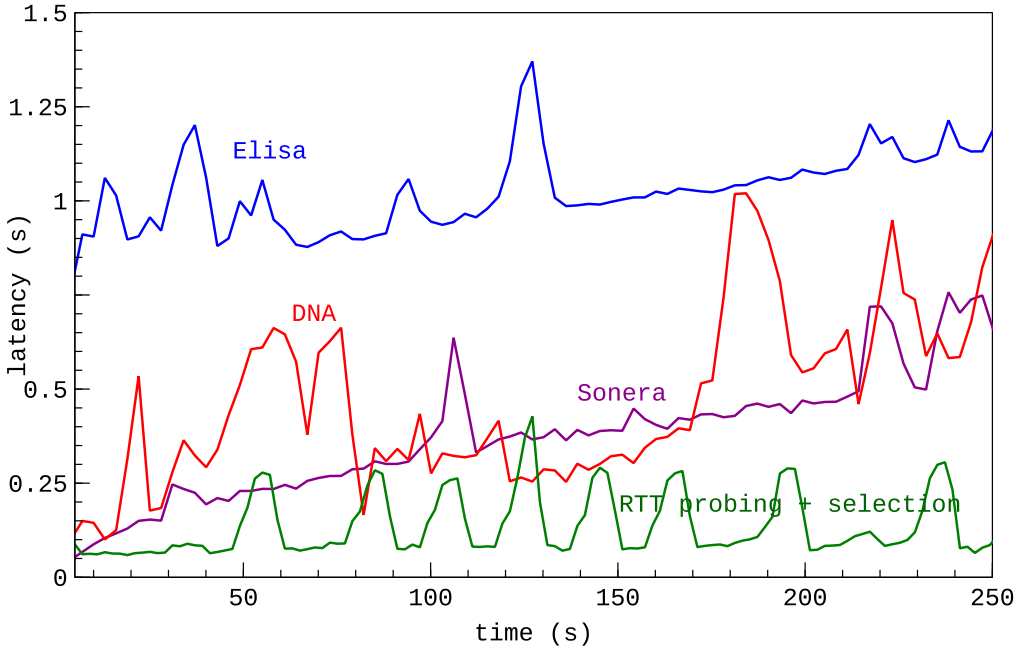

Figure 18: Indoor experiment: selection based on best RTT compared with normal TCP flows

6.7 Conclusion

In this section, the results of the experiments were presented and analyzed. An overview of the results for all configurations was given. For each configuration, notable or unexpected observations were underlined and explained.

The measurements confirm that MPTCP can substantially increase throughput if the available interfaces are used concurrently. It turned out that MPTCP can also decrease latency.

The interface selection algorithms met their objective as they improve performance without using all interfaces at once. Nevertheless, the results of the selection algorithm based on history and location are not as good as expected.

7 Conclusions

This thesis explored the use of Multipath TCP on multihomed mobile devices. As opposed to the previous related works that considered mobile devices equipped with one 3G access and one WLAN interface, this work focused on devices that support several 3G subscriptions. Phones with an active dual-SIM functionality are a possible application.

MPTCP allows a legacy TCP application to send and receive data on different paths concurrently thus increases the available bandwidth. Nevertheless, as using several interfaces simultaneously consumes more power, mobile endpoints could instead use the vertical handover capability of MPTCP to make a tradeoff between performance and power consumption. The active interface can be dynamically chosen according to a specific performance criterion such as available throughput or latency. In this way, MPTCP can be used to improve performance even though only one interface is used at a time. Three simple solutions for interface selection were considered in this work.

Previous works backed their results with simulations and short experiments. Nevertheless, the performance of MPTCP or any associated interface selection algorithm will eventually have to be evaluated in real-world conditions. That is why this work included the design and implementation of a test bed dedicated to real-world experiments. It comprises a framework that support easy implementation and evaluation of new interface selection processes for MPTCP. The three proposed path selection algorithms were implemented and tested.

To work appropriately, MPTCP requires a specific routing configuration. For now, as no operating system officially supports MPTCP, this configuration must be done manually. In order to make the process easier and save time during tests, this thesis proposed a tool that automatically configures Linux routing tables for MPTCP. It is a generic solution that works along the popular network configuration utility NetworkManager.

The results of the real-world experiments performed with the test bed confirmed that MPTCP increases the available throughput but also that it reduces the effects of bufferbloat. The interface selection algorithms met their objectives as they increased performance while limiting simultaneous transfers through 3G interfaces.

Future works could focus on optimizing interface selection by tuning the existing algorithms or implementing more elaborated ones. The test bed could be adapted to mobile platforms so it could use dual-SIM mobile devices instead of a laptop. This would be closer to potential applications and would allow to measure the real impact of the different configurations on energy consumption.

To conclude, MPTCP provides substantial network performance improvements on multihomed mobile endpoints. Dynamic interface selection is a reasonable way to reconcile energy-efficiency and resource pooling between available paths.

References

[1] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017," White Paper, Cisco, Feb. 2013. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf

[2] "Ericsson Mobility Report, Interim Update," White Paper, Ericsson, Feb. 2013. [Online]. Available: http://www.ericsson.com/res/docs/2013/ericsson-mobility-report-february-2013.pdf

[3] D. Wischik, M. Handley, and M. B. Braun, "The resource pooling principle," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 5, pp. 47–52, Sep. 2008.

[4] P. S. Schmidt, R. Merz, and A. Feldmann, "A first look at multi-access connectivity for mobile networking," in *Proceedings of the 2012 ACM workshop on Capacity sharing*, ser. CSWS '12. New York, NY, USA: ACM, 2012, pp. 9–14.

[5] S. Dimatteo, P. Hui, B. Han, and V. O. Li, "Cellular traffic offloading through WiFi networks," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*. IEEE, 2011, pp. 192–201.

[6] C. Sankaran, "Data offloading techniques in 3GPP Rel-10 networks: A tutorial," *Communications Magazine, IEEE*, vol. 50, no. 6, pp. 46–53, 2012.

[7] K. Ahmavaara, H. Haverinen, and R. Pichna, "Interworking architecture between 3GPP and WLAN systems," *Communications Magazine, IEEE*, vol. 41, no. 11, pp. 74–81, 2003.

[8] "Mobile networking through mobile IP, author = Perkins, Charles E." *Internet Computing, IEEE*, vol. 2, no. 1, pp. 58–69, 1998.

[9] C. Perkins, "IP Mobility Support for IPv4, Revised," RFC 5944, Internet Engineering Task Force, Nov. 2010. [Online]. Available: http://tools.ietf.org/rfc/rfc5944

[10] C. Perkins, D. Johnson, and J. Arkko, "Mobility Support in IPv6," RFC 6275, Internet Engineering Task Force, Jul. 2011. [Online]. Available: http://tools.ietf.org/rfc/rfc6275

[11] S. Rayanchu, "Expert testimony: Charles E. Perkins, senior principal engineer, Futurewei," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 16, no. 2, pp. 28–34, Nov. 2012.

[12] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream Control Transmission Protocol," RFC 2960, Internet Engineering Task Force, Oct. 2000. [Online]. Available: http://tools.ietf.org/rfc/rfc2960

[13] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration," RFC 5061, Internet Engineering Task Force, Sep. 2007. [Online]. Available: http://tools.ietf.org/rfc/rfc5061

[14] M. Riegel and M. Tuexen, "Mobile SCTP," Working Draft, IETF Secretariat, Internet-draft draft-riegel-tuexen-mobile-sctp-09, Nov. 2007. [Online]. Available: http://tools.ietf.org/id/draft-riegel-tuexen-mobile-sctp-09.txt

[15] A. Ezzouhairi, A. Quintero, and S. Pierre, "A New SCTP mobility scheme supporting vertical handover," in *Wireless and Mobile Computing, Networking and Communications, 2006.(WiMob'2006). IEEE International Conference on.* IEEE, 2006, pp. 205–211.

[16] P. Amer, M. Becke, T. Dreibholz, N. Ekiz, J. Iyengar, P. Natarajan, R. Stewart, and M. Tuexen, "Load Sharing for the Stream Control Transmission Protocol (SCTP)," Working Draft, IETF Secretariat, Internet-draft draft-tuexen-tsvwg-sctp-multipath-06, Mar. 2013. [Online]. Available: http://tools.ietf.org/id/draft-tuexen-tsvwg-sctp-multipath-06.txt

[17] J. Iyengar, K. Shah, P. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming," *SPECTS 2004*, 2004.

[18] J. R. Iyengar, P. D. Amer, and R. Stewart, "Retransmission policies for concurrent multipath transfer using SCTP multihoming," in *Networks, 2004.(ICON 2004). Proceedings. 12th IEEE International Conference on*, vol. 2. IEEE, 2004, pp. 713–719.

[19] D. A. Hayes, J. But, and G. Armitage, "Issues with network address translation for SCTP," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 23–33, Dec. 2008.

[20] M. Tuexen and R. R. Stewart, "UDP Encapsulation of SCTP Packets for End-Host to End-Host Communication," Working Draft, IETF Secretariat, Internet-draft draft-ietf-tsvwg-sctp-udp-encaps-14, Jul. 2011. [Online]. Available: http://tools.ietf.org/id/draft-ietf-tsvwg-sctp-udp-encaps-14.txt

[21] H.-Y. Hsieh and R. Sivakumar, "pTCP: An end-to-end transport layer protocol for striped connections," in *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on.* IEEE, 2002, pp. 24–33.

[22] I. van Beijnum, "One-ended multipath TCP," Working Draft, IETF Secretariat, Internet-draft draft-van-beijnum-1e-mp-tcp-00, May 2009. [Online]. Available: http://tools.ietf.org/id/draft-van-beijnum-1e-mp-tcp-00.txt

[23] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural Guidelines for Multipath TCP Development," RFC 6182, Internet Engineering Task Force, Mar. 2011. [Online]. Available: http://tools.ietf.org/rfc/rfc6182

[24] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, Internet Engineering Task Force, Jan. 2013. [Online]. Available: http://tools.ietf.org/rfc/rfc6824

[25] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *USENIX Symposium of Networked Systems Design and Implementation (NSDI'12), San Jose (CA)*, 2012.

[26] C. Paasch, S. Barre, J. Korkeaniemi, F. Duchene, G. Detal *et al.* MultiPath TCP - Linux Kernel implementation. [Online]. Available: http://mptcp.info.ucl.ac.be/

[27] M. Scharf, "Multi-Connection TCP (MCTCP) Transport," Working Draft, IETF Secretariat, Internet-draft draft-scharf-mptcp-mctcp-01, Jul. 2010. [Online]. Available: http://tools.ietf.org/id/draft-scharf-mptcp-mctcp-01.txt

[28] M. Scharf and T. Banniza, "MCTCP: A Multipath Transport Shim Layer," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, 2011, pp. 1–5.

[29] A. Singh and M. Scharf, "PayLoad Multi-connection Transport using Multiple Addresses," Working Draft, IETF Secretariat, Internet-draft draft-singh-mptcp-plmt-00, Aug. 2010. [Online]. Available: http://tools.ietf.org/id/draft-singh-mptcp-plmt-00.txt

[30] T. Dierks and C. Allen, "The TLS Protocol," RFC 2246, Internet Engineering Task Force, Jan. 1999. [Online]. Available: http://tools.ietf.org/rfc/rfc2246

[31] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[32] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550, Internet Engineering Task Force, Jul. 2003. [Online]. Available: http://tools.ietf.org/rfc/rfc3550

[33] V. Singh, S. Ahsan, and J. Ott, "MPRTP: Multipath Considerations for Real-time Media," 2013.

[34] V. Singh, T. Karkkainen, J. Ott, S. Ahsan, and L. Eggert, "Multipath RTP (MPRTP)," Working Draft, IETF Secretariat, Internet-draft draft-singh-avtcore-mprtp-06, Jan. 2013. [Online]. Available: http://www.ietf.org/id/draft-singh-avtcore-mprtp-06.txt

[35] S.-J. Koh and W. Hyun, "mSIP: Extension of SIP for Soft Handover with Bicasting," *Communications Letters, IEEE*, Jul.

[36] E. Boysen and J. Flathagen, "Using SIP for seamless handover in heterogeneous networks," in *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2011 3rd International Congress on*, 2011, pp. 1–8.

[37] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 281–287.

[38] C. Raiciu, M. Handly, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356, Internet Engineering Task Force, Oct. 2011. [Online]. Available: http://tools.ietf.org/rfc/rfc6356

[39] S. Barre, C. Paasch, and O. Bonaventure, "MultiPath TCP - Guidelines for implementers," Working Draft, IETF Secretariat, Internet-draft draft-barre-mptcp-impl-00, Mar. 2011. [Online]. Available: http://tools.ietf.org/id/draft-barre-mptcp-impl-00.txt

[40] J. Postel, "Transmission Control Protocol," RFC 793, Internet Engineering Task Force, Sep. 1981. [Online]. Available: http://www.ietf.org/rfc/rfc793

[41] Transmission Control Protocol (TCP) Parameters. IANA. [Online]. Available: http://www.iana.org/assignments/tcp-parameters/tcp-parameters.xml

[42] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681, Internet Engineering Task Force, Sep. 2009. [Online]. Available: http://tools.ietf.org/rfc/rfc5681

[43] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 8–8.

[44] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "MPTCP is not pareto-optimal: performance issues and a possible solution," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 1–12.

[45] S. Hassayoun, J. Iyengar, and D. Ros, "Dynamic Window Coupling for multipath congestion control," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, 2011, pp. 341–352.

[46] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley, "Opportunistic mobility with multipath TCP," in *Proceedings of the sixth international workshop on MobiArch*, ser. MobiArch '11. New York, NY, USA: ACM, 2011, pp. 7–12.

[47] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, "Exploring mobile/WiFi handover with multipath TCP," in *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*, ser. CellNet '12. New York, NY, USA: ACM, 2012, pp. 31–36.

[48] C. Diop, G. Dugué, C. Chassot, and E. Exposito, "QoS-aware multipath-TCP extensions for mobile and multimedia applications," in *Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia*, ser. MoMM '11. New York, NY, USA: ACM, 2011, pp. 139–146.

[49] C. Pluntke, L. Eggert, and N. Kiukkonen, "Saving mobile device energy with multipath TCP," in *Proceedings of the sixth international workshop on MobiArch*, ser. MobiArch '11. New York, NY, USA: ACM, 2011, pp. 1–6.

[50] L. Liang, H. Wang, and P. Zhang, "Net Utility-Based Network Selection Scheme in CDMA Cellular/WLAN Integrated Networks," in *Wireless Communications and Networking Conference, 2007.WCNC 2007. IEEE*, 2007, pp. 3313–3317.

[51] C.-J. Chang, T.-L. Tsai, and Y.-H. Chen, "Utility and Game-Theory Based Network Selection Scheme in Heterogeneous Wireless Networks," in *Wireless Communications and Networking Conference, 2009. WCNC 2009. IEEE*, 2009, pp. 1–5.

[52] P. N. Tran and N. Boukhatem, "Comparison of MADM decision algorithms for interface selection in heterogeneous wireless networks," in *Software, Telecommunications and Computer Networks, 2008. SoftCOM 2008. 16th International Conference on*, 2008, pp. 119–124.

[53] R. Tawil, O. Salazar, and G. Pujolle, "Vertical Handoff Decision Scheme Using MADM for Wireless Networks," in *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, 2008, pp. 2789–2792.

[54] L. Wang and D. Binet, "MADM-based network selection in heterogeneous wireless networks: A simulation study," in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*, 2009, pp. 559–564.

[55] L. Mohamed, C. Leghris, and A. Abdellah, "A survey and comparison study on weighting algorithms for access network selection," in *Wireless On-demand Network Systems and Services (WONS), 2012 9th Annual Conference on*. IEEE, 2012, pp. 35–38.

[56] P. E. Mogensen, T. Koivisto, K. I. Pedersen, I. Z. Kovacs, B. Raaf, K. Pajukoski, and M. J. Rinne, "LTE-advanced: the path towards gigabit/s in wireless mobile communications," in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*. IEEE, 2009, pp. 147–151.

[57] F. Patriarca, S. Salsano, and F. Fedi, "Efficient measurements of IP level performance to drive interface selection in heterogeneous wireless networks," in *Proceedings of the 9th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks.* ACM, 2012, pp. 25–30.

[58] O. Olvera-Irigoyen, A. Kortebi, and L. Toutain, "Available Bandwidth Probing for path selection in heterogeneous home Networks," in *Globecom Workshops (GC Wkshps), 2012 IEEE*, 2012, pp. 492–497.

[59] H. Kwon, M.-J. Yang, A.-S. Park, and S. Venkatesan, "Handover prediction strategy for 3G-WLAN overlay networks," in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, 2008, pp. 819–822.

[60] "Wi-Fi in heterogeneous networks - an integrated approach to delivering the best user experience," White Paper, Ericsson, Nov. 2012. [Online]. Available: http://www.ericsson.com/res/docs/whitepapers/wp-wi-fi-in-heterogeneous-networks.pdf

[61] X. Pérez-Costa, M. Torrent-Moreno, and H. Hartenstein, "A performance comparison of Mobile IPv6, Hierarchical Mobile IPv6, fast handovers for Mobile IPv6 and their combination," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, no. 4, pp. 5–19, Oct. 2003.

[62] Netradar. [Online]. Available: http://www.netradar.org/

[63] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, Jul. 1997.

[64] NetworkManager. [Online]. Available: http://projects.gnome.org/NetworkManager/

[65] M. Handley, C. Raiciu, and M. Bagnulo, "Outgoing Packet Routing with MP-TCP," Working Draft, IETF Secretariat, Internet-draft draft-handley-mptcp-routing-00, Oct. 2009. [Online]. Available: http://tools.ietf.org/id/draft-handley-mptcp-routing-00.txt

[66] M. Scharf and A. Ford, "Multipath TCP (MPTCP) Application Interface Considerations," RFC 6897, Internet Engineering Task Force, Mar. 2013. [Online]. Available: http://tools.ietf.org/rfc/rfc6897

A Energy-efficiency and throughput-based selection algorithms

The case of a mobile host that has three 3G interfaces 1, 2 and 3, identical in terms of energy consumption is considered. It begins to download a file of size S. The throughput available through each interface remains constant during the transfer and MPTCP is able to maximize throughput utilization. The mobile device runs an interface selection algorithm that selects the interface with the best throughput (in this case, interface 1). According to Pluntke et al. [49], energy the consumption of a 3G interface during a data transfer can be expressed as in (A1).

$$C_c + C_t \cdot T \tag{A1}$$

(A1) is the power consumption of the interface (in Watts), C_c is a constant component and $C_t \cdot T$ a component that linearly depends on the throughput where T is the throughput. (A2) is the energy needed to download the file of size S concurrently on all three interfaces, C_0 being the energy consumption due to timeouts after the end of a transfer on a 3G interface. (A3) is the energy needed to download the same file after the selection of the interface 1. In both cases, the device will consume at least $3 \cdot C_0$ as all interfaces have to be activated once, either for data transfer in the case of normal MPTCP or for probing or database querying in the case of interface selection.

$$\frac{S}{T_1 + T_2 + T_3} \left[3 \cdot C_c + C_t \left(T_1 + T_2 + T_3\right)\right] + 3 \cdot C_0 \tag{A2}$$

$$\frac{S}{T_1} \left(C_c + C_t \cdot T_1\right) + 3 \cdot C_0 \tag{A3}$$

In order for the selection to be useful in terms of energy efficiency, (A3) must be lower than (A2). It leads to inequalities (A4) and (A5).

$$\frac{S}{T_1} \left(C_c + C_t \cdot T_1\right) \leq \frac{S}{T_1 + T_2 + T_3} \left[3 \cdot C_c + C_t \left(T_1 + T_2 + T_3\right)\right] \tag{A4}$$

$$T_1 \geq \frac{T_1 + T_2 + T_3}{3} \tag{A5}$$

In order for the selection algorithm to decrease energy consumption compared with normal MPTCP, the selected interface must provide a better throughput than the average throughput of the three interfaces. Then, selecting the interface with the best throughput improves energy-efficiency.