# Practical Multi-Interface Network Access for Mobile Devices

by

Jakub K. Schmidtke

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

Despite the growing number of mobile devices equipped with multiple networking interfaces, they are not using multiple available networks in parallel. The simple network selection techniques only allow for single network to be used at a time and switching between different networks interrupts all existing connections. This work presents system that improves network connectivity in presence of multiple network adapters, not only through better network handovers, smarter network selection and failure detection, but also through increased bandwidth offered to the device over aggregated channels.

The biggest challenge such a system has to face is the heterogeneity of networks in mobile environment. Different wireless technologies, and even different networks of the same type offer inconsistent link parameters like available bandwidth, latency or packet loss. The wireless nature of these networks also means, that most of the parameters fluctuate in unpredictable way. Given the intended practicality of designed system, all that complexity has to be hidden from both client-side applications and from the remote servers. These factors combined make the task of designing and implementing an efficient solution difficult.

The system incorporates client-side software, as well as network proxy that assists in splitting data traffic, tunnelling it over a number of available network interfaces, and reassembling it on the remote side. These operations are transparent to both applications running on the client, as well as any network servers those applications communicate with. This property allows the system to meet one of the most important requirements, which is the practicality of the solution, and being able to deploy it in real life scenarios, using network protocols available today and on existing devices. This work also studies the most critical cost associated with increased data processing and parallel interface usage - the increase in energy usage, which needs to remain within reasonable values for this kind of solution being usable on mobile devices with limited battery life.

The properties of designed and deployed system are evaluated using multiple experiments in different scenarios. Collected results confirm that our approach can provide applications with increased bandwidth when multiple networks are available. We also discover that even though per-second energy usage increases when multiple interfaces are used in parallel, the use of multi-interface connectivity can actually reduce the total energy cost associated with performing specific tasks - effectively saving energy.

**Acknowledgements**

I wish to thank my supervisor, Dr. Sagar Naik, for his guidance and assistance with this thesis. I would also like to thank Dr. Sherman Shen and Dr. Pin-Han Ho, for acting as the readers of this thesis.

I also would like to express my gratitude to Pravala Inc. for offering me resources and support required for completion of this thesis. I am especially thankful to Richard Wagner for his personal help and support.

I am also grateful to my close friends Ewa Chachulska, Jan Łajkowski and Jakub Gawryjołek, for much needed positive distractions. I especially want to thank Marni Lee, for making my life in Waterloo easier and more enjoyable.

Last but not least, I want to thank my family, my mother Elżbieta, my father Krzysztof, and my brother Piotr for their support, encouragement and for believing in me.

**Dedication**

*To my mother*

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| 2G | 2nd Generation wireless telephone technology |
| 3G | 3rd Generation of mobile telecommunications technology |
| 4G | 4th Generation of mobile telecommunications technology |
| ACK | Acknowledgement |
| AISLE | Autonomic Interface SeLEction |
| API | Application Programming Interface |
| CPU | Central Processing Unit |
| DSL | Digital Subscriber Line |
| DSL | Digital Subscriber Line |
| FIFO | First In First Out |
| GPRS | General Packet Radio Service |
| HTTP | Hypertext Transfer Protocol |
| I-TCP | Indirect Transmission Control Protocol |
| I/O | Input/Output |
| IEEE | Institute of Electrical and Electronics Engineers |
| IP | Internet Protocol |
| IPv4 | Internet Protocol, version 4 |

| | |
|---|---|
| IPv6 | Internet Protocol, version 6 |
| ISP | Internet Service Provider |
| KB | Kilobyte |
| LTE | Long Term Evolution |
| MB | Megabyte |
| Mbps | Megabits per second |
| MLPPP | Multi-link Point-to-Point Protocol |
| MSR | Mobile Support Router |
| NAT | Network Address Translation |
| OSI | Open Systems Interconnection |
| POP3 | Post Office Protocol, version 3 |
| PPP | Point-to-Point Protocol |
| PPPoE | Point-to-Point Protocol over Ethernet |
| pTCP | Parallel Transmission Control Protocol |
| R-MTP | Reliable Multiplexing Transport Protocol |
| RTO | Retransmission TimeOut |
| RTT | Round-Trip Time |
| SACK | Selective Acknowledgement |
| SCTP | Stream Control Transmission Protocol |
| SEBAG | Session Layer Bandwidth Aggregation |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| WiMAX | Worldwide Interoperability for Microwave Access |

| | |
|---|---|
| WLAN | Wireless Local Area Network |
| XFTP | Multicast File Transfer Protocol |

# Chapter 1

# Introduction

In today's mobile world there are devices capable of accessing multiple types of networks - 3G/4G, LTE, Wi-Fi and others. Even though the number of mobile devices equipped with multiple networking interfaces keeps growing, users are not using multiple available networks in parallel. The simple network selection techniques only allow for single network to be used at a time and switching between different networks causes all sorts of connectivity issues. Networking connectivity in presence of multiple network adapters can be greatly improved, not only through improved network handovers, smarter network selection and failure detection, but also through increased bandwidth offered to the device over aggregated channels. The primary problem of multi-interface connectivity is the traffic splitting and scheduling. Different traffic types have to be handled in a way appropriate for them, in a number of scenarios and conditions, to offer improved user's experience. A separate issue is the increased energy consumption that may be the consequence of increased data processing and network interface usage, which is especially critical for mobile devices. This research focuses on different ways of traffic scheduling, and the energy costs associated with them.

## 1.1 Mobile devices

Mobile devices are gaining more and more popularity, and their capabilities keep growing. Among them are various laptops and other portable computers. But the category of the devices that changed the most over the last few years are feature-rich mobile phones, called "smartphones".

Smartphones not only allow users to have voice conversations or send and receive text messages, but also offer much more advanced features. Those phones have comparatively more computation resources and can run far more sophisticated operating systems and more advanced applications which users install. Smartphones also typically allow independent developers to easily create applications (subject to terms of service and the rules and restrictions of various application stores). Early smartphones offered only basic applications and did not need significant amounts of bandwidth (which was unavailable at that time anyway). Most of the features were targeted at business users, offering them simple services such as email and calendar. Over time smartphones gained more power and more data consuming features, such as integrated cameras and web browsers. Until then, except for the microphone that was required for voice calls, mobile devices allowed users mostly to receive content. Adding cameras allowed users to easily create content as well. Cameras present even in high end mobile cameras typically could not compete with full sized cameras manufactured around the same time. Most people, however, carry their mobile phones with them all the time, unlike their regular cameras. Having a camera in a device that users constantly carry with them is very convenient. This is the reason why nearly every smartphone released recently is equipped with at least one camera.

Another category in which mobile devices changed significantly is the network connectivity. A few years ago mobile phones were limited to cellular networks that, at first, offered poor data connectivity. Today, practically all smartphones and even some feature phones are equipped with Wi-Fi (or WLAN - Wireless Local Area Network) interfaces, that allow them to access the network at usually greater speeds than over the mobile networks.

## 1.2 Problems

As the mobile devices gain popularity, the number of devices competing for limited network resources grows, which becomes a problem especially in the wide area networks. These networks are designed to work at much larger distances than, for example, Wi-Fi networks. The cost is a lower bandwidth and a higher number of users competing for access which further decreases available resources. This is the case with 3G (3rd Generation of mobile telecommunications technology) and 4G (4th Generation of mobile telecommunications technology) cellular networks, but also with other wide area networks, such as WiMAX (Worldwide Interoperability for Microwave Access). Even though WiMAX is "similar" to Wi-Fi, it offers much smaller data rates [28]. On the other hand, 802.11-based networks offer higher speeds, but their range is very limited. All of the standards mentioned here are constantly evolving and being improved. When new revisions are announced they

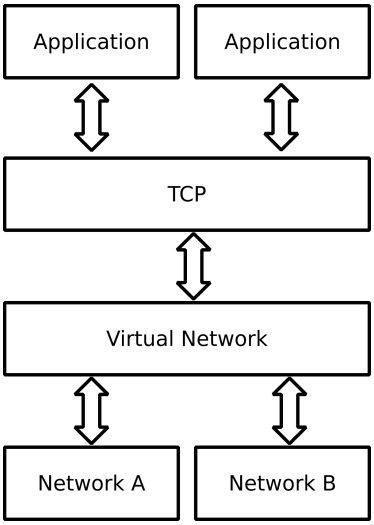



Figure 1.1: Virtual network interface

usually offer better parameters. However, a network with a larger area has to support more competing users. For this reason there will never be a single type of a network that offers long distance connectivity and data rates which are unchallenged by another technology designed to work at a shorter range. In turn, there will always be a number of different networks designed for various purposes. This means, that there will always be a plurality of different networking technologies that, if used properly, could offer improved network connectivity.

The problem is, the devices available today do not offer practical way of utilizing multiple networking technologies in parallel. Usually it is one network or the other, and user can choose which one it is. This is very far from efficient usage of resources available.

One of the approaches for better utilization of multiple networks would be to modify existing applications and the network services they interact it. However, given the number of different applications, this is hardly a practical approach. Another, is to "aggregate" multiple networks and resources they offer, and present it to applications as a single, virtual network with increased capacity - as presented in Figure 1.1.

We believe, that this is a viable and practical approach to taking advantage of plurality of networking technologies available. There are, however, multiple challenges associated with this approach.

The first of the problems is related to the fact that when two different interfaces are used, their characteristics may be completely different. Depending on the networking technology,

but also current usage and the number of users, packets sent over those different networks will experience different latencies, bandwidths, and drop rates. We want to be able to utilize the capacity offered by all of the interfaces concurrently. The naive method of splitting a packet stream between available interfaces might result in poor performance due to severe packet reordering. Other methods of the interface selection for data packets should be used. The goal is to use available networks' full potential without negatively effecting the data sent over the system. The scheduling method should avoid packet reordering and increasing latencies that would negatively affect the user's experience. It also needs to take into account not only latencies on connected networks, but also their available bandwidth. Those parameters can change over time and the method should adjust itself correspondingly.

Another problem is providing connectivity that is not interrupted when the device switches between different networks. This would be easy to achieve on the application level. However, we can neither modify internet servers nor the applications running on the device. Most applications use either TCP (Transmission Control Protocol) or UDP (User Datagram Protocol) sockets for communication. These run on top of the IP (Internet Protocol) network which does not have the concept of hand-over and cannot work with multiple source or destination addresses. A device connected to multiple networks has multiple IP addresses. When it changes any of the networks to which it is connected it receives a new IP address. This change causes all existing connections using the old IP address to break. These connections simply cannot be migrated between different interfaces. The practical system we are implementing will create a virtual network above existing, physical networks, which means that it will keep using the same network address. But we cannot avoid switching between these physical networks, and so we still need to be able to migrate data flows between the available physical interfaces in a way that is not disruptive to applications using these flows.

The last problem is the fact that the environment where the system discussed works is mobile. This means that the devices used have limited power supply. The short battery life is a problem most users of mobile devices have to deal with even without using advanced networking solutions, like the one discussed in this work. For any solution to be really practical, the impact on battery life should not be too significant.

## 1.3 Contributions

The main contribution of this work is the design and development of a system that provides improved behaviour in mobile environments with available multiple networking technolo-

gies.

The biggest challenge such a system has to face is the heterogeneity of networks in mobile environment. Different wireless technologies, and even different networks of the same type offer inconsistent link parameters like available bandwidth, latency or packet loss. The wireless nature of these networks also means, that most of the parameters fluctuate in unpredictable way. Given the intended practicality of designed system, all that complexity has to be hidden from both client-side applications and from the remote servers. These factors combined make the task of designing and implementing an efficient solution very difficult.

Our design of such a system is based on software components running on the mobile client device, as well as on a remote proxy server. The client-side software layer is operating between the applications running on that device and the multiple underlying physical networks. Traffic from the applications is intercepted and scheduled over the available networks to maximize their utilization, as well as provide transparent fail-over between different networks as their availability changes.

The server-side component of the system performs similar operations by scheduling traffic travelling in the opposite direction - to the client device. It also reassembles split traffic from the client and forwards it to remote internet servers as if it was sent by a regular device over a single network.

One of the most important aspects of the design of the system is its practicality, which means that it can be deployed in today's internet without requiring unrealistic modifications to existing networking infrastructure or internet servers (which means no modifications to those elements). This system is also deployable to mobile devices without involving network carriers or device manufacturers. There are several existing publications discussing different ideas for providing improved multi-interface networking experience. Most of the existing publications propose changes that cannot be easily deployed in the real world and only provide theoretical or simulation-based evaluation. In comparison, this system provides users with a seamless experience in real life scenarios and can be deployed with minimal (and realistic) modifications to real-life devices.

Novel elements of the system include a traffic scheduler that performs efficient flow detection and flow-based scheduling. It does not require prohibitory amounts of memory or processing capabilities, that would not be available on most mobile devices. Instead, it employs very simple, yet efficient, algorithm for flow detection. The algorithm is very fast and uses minimal amount of memory - which does not increase as the number of flows grows.

The scheduler is also capable of reacting to changing network and traffic conditions. The

network conditions can change in several ways. Existing working networks can disappear due to client's movement or even failures. The throughput offered be them can also degrade gradually, for instance due to changing number of other users and their activity. The network conditions can also improve, due to similar reasons. Whenever something changes in physical networks, the system can adapt to the new conditions, by switching to other networks, or simply by changing the scheduling decisions resulting in different usage of available resources.

The traffic parameters can change as well, due to user's actions and running applications. The system could react to changing demands of the client device, for example, scheduling more critical data streams on better networks, while scheduling background, less critical traffic over other networks - providing more bandwidth to the foreground applications.

In contrast to some of the existing work in the area, the evaluation of this system is not performed using simulations and theoretical network models, but in "real life" scenarios in regular networks on regular devices.

In addition, this work includes the measurements of the impact of the implemented system on the battery life of the real, mobile devices, to make sure that the system not only offers significant benefits to the user in terms of utilizing network resources, but also achieves this without significantly shortening the battery life of the mobile device on which it operates.

## 1.4 Outline of the document

The rest of this thesis is organized as follows:

Chapter 2 presents related work in the area of multi-interface connectivity and several other relevant publications that apply to this work.

The solution to the problem described in this chapter, and the system design is presented in Chapter 3.

In Chapter 4 the experiments performed and the methods of evaluating the system's behaviour are discussed.

Chapter 5 presents the results of the experiments performed.

Finally, Chapter 6 contains concluding statements.

# Chapter 2

# Literature review

There are several publications that discuss different ideas and aspects of multi-interface connectivity and bandwidth aggregation. All of them have various shortcomings, preventing them from being a base for multi-interface oriented solution for mobile hosts. There are a number of different solutions for most network layers. Even though most OSI (Open Systems Interconnection) layers are discussed in this chapter, the focus is on layers 3 to 5, as presented in Figure 2.1.

The system presented and implemented as a part of this work operates at the third OSI layer - the network layer. This chapter, in addition to presenting existing research that focuses on different layers, includes discussion and reasoning behind this design decision.

## 2.1 Application layer solutions

"Application layer solution" (Figure 2.2) means that the application itself has to be aware of the fact that multiple, physical interfaces are available. This is very different from simple applications that open several "connections". Even though multiple flows are required at the application level (as presented in Figure 2.3), the application must be aware of (and manage) the actual, system-level interfaces available.

### 2.1.1 Applications using multiple physical interfaces

A multi-interface system in the application layer would have to deal with a very low level concept of the network interface and its parameters. Those operations are very

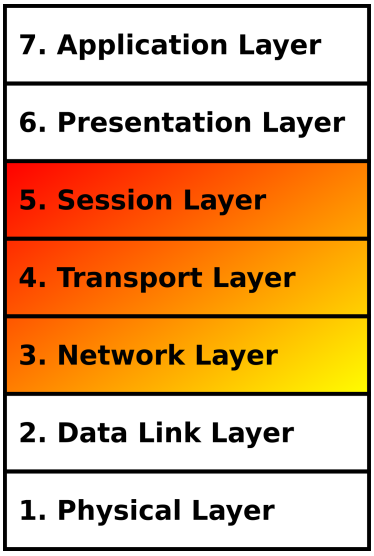



Figure 2.1: OSI layers

tightly bound to the operating system and each system type has its own API (Application Programming Interface). Different types of interfaces within the same system can have different parameters and capabilities depending on the network type or on the type of device driver supporting the given interface. Every application wanting to use multiple interfaces would have to be able to understand those different devices and their parameters. In the case that an application is written for more than one platform, a separate, different code would have to be used for each of them. This additional effort would have to be repeated for every application that would benefit from the multiple interfaces' availability. Adding support for multiple interfaces at the application level would be a laborious and difficult task, making multi-interface connectivity impractical at this level.

Another problem is that applications would need low level system access. This kind of access is usually required to be able to access and modify low level parameters of network interfaces and control interface selection. Most user space applications do not and should not have that kind of access.

### 2.1.2 Applications using multiple connections

For completeness' sake, this section contains descriptions of several application level solutions that, even without using multiple physical interfaces, try to solve problems relevant

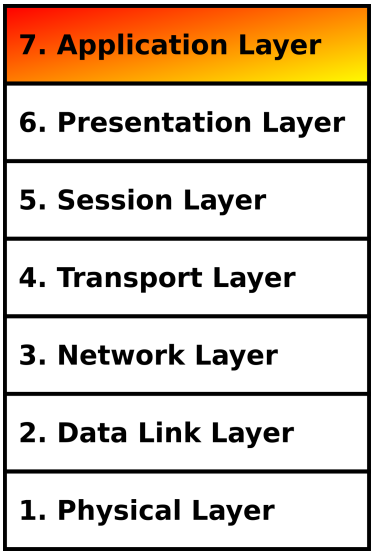



Figure 2.2: Application layer

to multi-interface connectivity. Although some of the problems are similar, this is very different from dealing with actual, separate physical interfaces.

A number of applications that use multiple, separate connections for various reasons exist. For example, XFTP (Multicast File Transfer Protocol) [4] opens several TCP connections in parallel to improve its performance over satellite links which offer large bandwidths, but also significant latencies. When a single TCP connection is used its limited window size, combined with a slow start algorithm, causes the connection to take a long time over these links with lengthy round trip times. Subsequent recoveries will also take a long time to restore a high data rate. By using several TCP connections and splitting parts of transmitted files between them, XFTP is able to perform significantly better. Since the maximum TCP's throughput is limited by $throughput_{max} = receive\ buffer\ size/round\ trip\ time$, a single TCP connection over a link with high bandwidth and very high delay might not even be able to utilize the entire bandwidth available. By using several parallel TCP connections, XFTP uses a much bigger, "virtual" buffer size that offers better behaviour over links with large latencies.

In this case, the aggregation is a simple task. XFTP is an application that knows exactly what needs to be sent, its size, and can split the data into several parts to be sent in parallel. It avoids problems related to per packet scheduling, reordering packets, and many other solutions, but it can do so only because it "works" at the application layer. Solving the same problem at the lower layers is much more difficult.

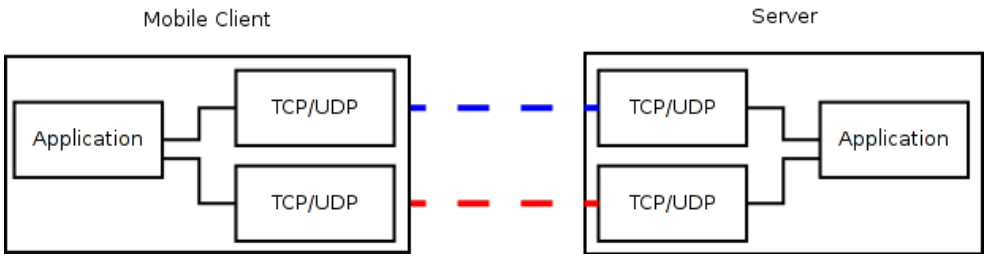



Figure 2.3: Application with multiple flows

Another example of applications that use multiple, parallel TCP connections are web browsers. Most of them open several connections to fetch different objects in the web page faster. For some applications, such as all BitTorrent clients, multiple connections are the natural way of operating. However, even in these cases applications assume only one interface. They would have to be modified to handle a number of separate interfaces connected to different networks efficiently. There also exists a myriad of other applications that do not use multiple data streams.

### 2.1.3 Limitations of application layer solutions

Application layer solutions are impractical for several reasons. They require applications to access and modify low level system resources, posing problems with application portability and security. However, the most important issue is the fact that such a solution would require all applications implementing it to be modified. Also, the respective servers those applications connect to would have to be modified as well. This task would be close to impossible to accomplish.

There exists another issue with this type of solution. Because they would operate on the application level, each application would have to run a separate instance of data scheduler. Each application in the system would be running its own, possibly completely different, scheduler. There would be no cooperation between different applications regarding how available interfaces are used. For example, scheduling connections of some of the applications over the first interface, and connections of remaining applications over the second, would be very difficult. Instead, each application would have to try to use all available interfaces in parallel, possibly yielding worse overall connection parameters. To achieve any form of cooperation, all those applications would have to either communicate with each other or with a central, unified resource manager with which all of them would have to be able to communicate. This would increase the implementation and deployment complexity

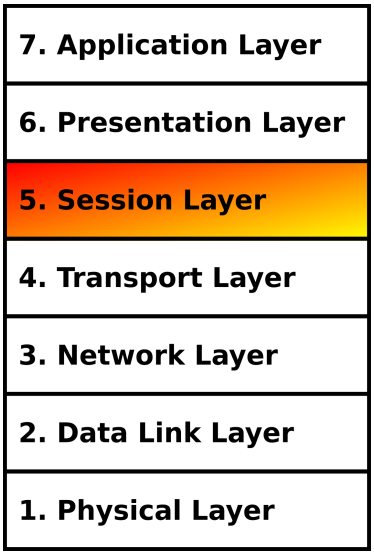



Figure 2.4: Session layer

and difficulty even further.

## 2.2 Session layer solutions

Session layer solutions (Figure 2.4) are often very similar to transport layer solutions from an implementation perspective. They too require an application to be recompiled to use different transport layer calls or to mask those calls using the LD_PRELOAD method (or similar). The difference is that they do not enforce transport protocol modifications and use existing unmodified transport protocols underneath. They only participate in data exchange between the local application and the remote host, using underlying connections established by unmodified transport protocols for actual data transfer.

Manoj, Mishra, and Rao propose using SEBAG - "Session Layer Bandwidth Aggregation" [46]. The authors identify some of the problems experienced by users of multi-interface capable devices as well as problems of transport layer solutions. They propose a scheduler that splits data flow from the application between a number of underlying TCP connections to the destination. Data to be split is divided in segments that are larger than single TCP packets. Segments are assigned to connections using the "Expected Earliest Delivery Path First" method, which considers estimated latency and available bandwidth of individual links, sending more data over links with lower latency and higher throughput.

On the remote side of the connection the appropriate SEBAG process restores the order in which the application sent the original packets. This solution does not introduce its own congestion control, but it relies on TCP's algorithms and only performs link selection. Their paper presents results of some real-life experiments, but it lacks details, especially in the way it integrates with the applications. SEBAG also shares the problem of all transport layer solutions. It needs to be deployed on both the local device and each remote server to which the mobile client might connect.

Another session layer solution that was tested using experiments with real implementation is presented by Thompson *et al.* [64]. It requires no modifications to the applications, only to the LD_PRELOAD mechanism to mask some of the network calls. It uses standard TCP and does not require any cooperation from the server. It was, however, designed for a slightly different environment, and even though it offers bandwidth aggregation for multiple flows (but not for a single flow, unlike the other solutions mentioned), it assumes no mobility. It requires only a daemon on the client device which measures the available links' parameters. It also tries to predict the behaviour of connections the application opens to bind those connections to specific interfaces. It does not allow for the migration of connections, and if the prediction was incorrect it might cause undesirable scheduling. A network failure or the user moving would cause some of the interfaces to become unavailable and all the connections scheduled over that interface would be interrupted.

### 2.2.1 Limitations of session layer solutions

Session layer and application layer solutions are in some ways very similar. However, in the session layer implementation-related issues are not as important. Modifying every single application that uses a specific solution is unnecessary. A single implementation per operating system type would be sufficient. However, in this case that implementation would have to be more complex and "trick" the application into using it rather than the transport protocols directly. This approach is complicated and prone to error.

Concurrently, the problem of deploying such a solution is as difficult as in the application layer case. For the mobile client to be able to use internet services over multiple interfaces using a session layer solution each of the servers providing those services would have to be modified. Even though modifications would not require changing the source files of those programs, the server's configuration would have to be modified. The number of such servers is enormous. Also, different servers are controlled by different organizations and most of them are very protective about their infrastructure's integrity. For these reasons, session layer solutions are also impractical.

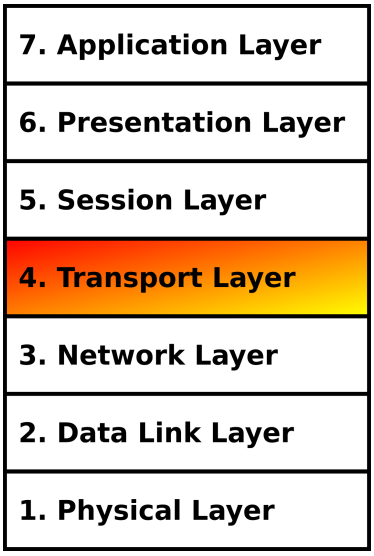



Figure 2.5: Transport layer

The only way such a solution could be adopted would be through creating a widely recognized standard supported by several big organizations and content providers. Even so, adoption would take several years.

## 2.3 Transport layer solutions

There are a number of publications that propose solutions located in the network stack's transport layer (Figure 2.5). This is probably the most common layer at which solutions to the multi-interface connectivity problem are proposed. This is the point where existing protocols perform congestion control and buffer management and are the elements that likely need modification so that the changes in that layer are the most natural.

### 2.3.1 SCTP-based protocols

The Stream Control Transmission Protocol (SCTP) [62] was originally introduced in 2000. It is designed to serve a role similar to the popular transport protocols TCP and UDP. It offers some of these two protocols' features, and also has some new, unique properties. Just like TCP, SCTP is connection oriented and offers ordered, reliable data delivery. At the

same time, just like UDP, it is message-oriented and can transfer data without ordering in an unreliable mode. Data can also be transmitted as multiple, separate streams within the same connection. The important feature for multi-interface connectivity is the support for multi-homing, in which both endpoints of a SCTP connection can consist of multiple IP addresses. This feature allows the protocol to offer transparent fail-over between different networks. However, SCTP cannot use more than one interface at a time as a primary data transfer channel. Other paths can only be used for retransmitting lost data chunks, or in case when the primary path fails. Another limitation of multi-interface support in SCTP is that it is not dynamic. SCTP has the ability to use multiple interfaces that are available when the connection is established. However, it does not react to the interface changes once the connection has been opened, so its use in mobile scenarios is very limited. Stewart *et al.* [63] propose an extension to SCTP that allows for dynamic address reconfiguration - ADDIP. The problem is that it needs to be supported by both endpoints of each SCTP connection and still does not enable bandwidth aggregation. Normally SCTP would also require support from the operating system, but it is possible to tunnel SCTP packets over UDP protocol [26]. It is also possible to run TCP-only applications with SCTP, but it requires modifications in the operating system's kernel [13]. Also, just like the other transport-layer solutions, SCTP still needs to be supported by both ends of every connection. It also does not work with NAT (Network Address Translation), unless the NAT server is modified as well [31].

Various other publications explore different ideas for improving SCTP's performance, one of which is AISLE (Autonomic Interface SeLEction) [16]. AISLE does not propose bandwidth aggregation, but measures bandwidth available on all existing links (using packet pairs) and allows SCTP to dynamically choose the single interface with the highest available bandwidth. It also estimates bandwidth used, based on ACKs (Acknowledgements) over the current primary link, in a way similar to TCP Westwood [47] and TCP Westwood+ [30]. This proposal is evaluated purely using simulations.

Iyengar *et al.* [34], using simulations, studied different retransmission policies for SCTP and conclude that the default policy could be greatly improved when all the available links are used for transmitting data. Huang and Zeng propose another modification to SCTP's retransmission policy [33]. Both solutions are only evaluated using simulations.

Fracchia *et al.* [29] propose an algorithm that tries to distinguish between packet loss due to radio problems (in wireless scenarios) and congestion. They simulate a modified SCTP protocol's behaviour using different congestion algorithms depending on the actual and estimated data rate. Estimations are performed differently on primary and secondary links - either using packet pair method or the algorithm used by Westwood+ TCP. This modification of SCTP can also change which link is used as the primary one.

Some of the ideas for improving the original SCTP's behaviour include methods that allow for bandwidth aggregation. The authors of "Concurrent Multipath Transfer Using SCTP Multihoming" [36] propose using separate congestion windows for each link, each with separate congestion control parameters, and sending data over any link that has space in its congestion window. In the follow-up publication [35] they examine some of the issues in greater depth. They compare the results with a special application that performs data transfers using several parallel SCTP "associations".

Argyriou and Madisetti also argue for bandwidth aggregation for SCTP [6]. Their modification of SCTP protocol tries to identify shared bottlenecks using a method presented by Rubenstein *et al.* [55] and performs congestion control for flows that share those bottlenecks. However, it does not present necessary modifications to the protocol. It also uses a simplistic retransmission algorithm based on somewhat unrealistic assumptions. It also ignores the impact of a limited size of the receiver's buffer.

Casetti and Gaiotto present Westwood SCTP [17] that tries to modify the SCTP in a minimal way. It uses most of the standard SCTP features, but also performs bandwidth estimation based on Westwood+ TCP. The standard SCTP SACKs (Selective Acknowledgements) and retransmission algorithm is used, but packet scheduling depends on link characteristics to minimize the expected delivery time. It assumes, however, that all available paths have very similar delays and only focuses on bandwidth differences. It is only evaluated using simulations.

cmpSCTP, presented by Liao *et al.* [43], is based on SCTP with dynamic interface extension (ADDIP) [63]. This modification adds multiple congestion windows and separate buffers for each path. This is a different approach, since most other solutions that use separate congestion windows still use a single buffer for easier packet rescheduling. It also introduces some additional sequence numbers for data packets increasing the

Another modification of SCTP, LS-SCTP, is presented by Saadawi and Lee [56]. They only evaluated LS-SCTP using opnet simulator, but they plan to develop a practical implementation in the future. This solution requires both sides of the connection to be modified, and also introduces path IDs and additional sequence numbers to the protocol. It also benefits from application modifications, in which case specific path selection is provided. This protocol offers bandwidth aggregation, but unfortunately the path selection algorithm is not described, except for the statement that it is based on bandwidth availability. Authors do not talk about the several algorithms required for this solution to work, such as initializing and shutting down the connection and setup of all used sequence numbers. There are also some potentially problematic or simplified assumptions made, such as in-order delivery of packets following the same path and the fact that the receiver's buffer does not

constrain the sender. The method of reusing sequence numbers for packets reassigned between different paths is not discussed, and it could cause problems in the real life scenario. There are also some incorrect assumptions about the way the original SCTP retransmits packets when the primary link is causing problems.

Shi *et al.* [59] present an evaluation using not only simulations, but also testing their solution with Linux-based implementation. Their proposal requires some modifications to SCTP, but only on the sender's side (however, the receiver still needs to support the basic SCTP). This algorithm performs a shared bottleneck detection introduced by Rubenstein *et al.* [55], and separate congestion windows for each path. It follows the original SCTP design and sends data using the primary link first, but it also can utilize extra links when the primary link is congested.

### 2.3.2 TCP-based protocols

Hsieh and Sivakumar [32] propose a protocol called pTCP (Parallel Transmission Control Protocol) which achieves bandwidth aggregation by opening several connections - one over each interface - using modified TCP protocol. Each of these TCP-like connections is responsible for congestion control and reliability, just like the regular TCP, but the buffer management is done by a separate module called Striped Connection Manager. It operates above several data links and assigns (and in certain cases re-assigns) individual packets to underlying connections. It is also responsible for opening and closing individual connections to the destination. The biggest drawback of this solution is that it is not compatible with regular TCP; it uses a modified TCP header.

Baldini *et al.* [11] present a somewhat similar idea. They propose opening several standard TCP connections and transmitting data from the application striped over multiple channels. In some aspects it is better than Hsieh and Sivakumar's proposal, which uses modified TCP protocol that could encounter routing problems. The evaluation is based not only on simulations, but also on experiments using real-life implementation of the protocol. However, it shares some of the issues of Hsieh and Sivakumar's solution. Applications to benefit from Baldini's group's protocol have to be modified (although in very small way) and recompiled. Another disadvantage is the fact that parts of both systems have to be running on both sides of every connection; on the client device and on any server it might wish to connect to.

### 2.3.3 Other protocols

Magalhaes and Kravets [45] present their own R-MTP (Reliable Multiplexing Transport) protocol. It is different from TCP-based protocols because it does not use separate TCP streams (modified or not) for underlying operations. It is instead designed to work as one algorithm operating on several channels. It uses selective ACKs, and its congestion control mechanism is rate-based, actively probing the available bandwidth using packet pair method (see Keshav [40]). The presented results, achieved with user-space implementation of the protocol, show that this protocol can offer some benefits, especially in wireless scenarios when loss-based congestion control (like regular TCP) does not perform very well. The results also show that aggregating two channels with significantly different characteristics can actually lead to worse results than when only the link with higher bandwidth is used. This protocol, however, shares the problems of the other transport-layer solutions. It needs to be supported by both the client and the server of every connection. Since this is a completely new transport protocol, all the routers on the path between the client and the server should be configured in a way that will not cause them to drop those "unknown" packets. For the same reason solutions such as NAT would also not work.

### 2.3.4 Limitations of transport layer solutions

Even though it is easiest and most natural to introduce the multi-interface connectivity to the transport layer, solutions for this layer pose several problems. Introducing new transport protocol usually requires modifications to both the operating system's network stack and the applications. This is necessary because there really is no generic "application layer protocol", or "session layer protocol" common to all applications. All programs that perform network operations, even if they run specific data protocol such as HTTP (Hypertext Transfer Protocol) or POP3 (Post Office Protocol, version 3), still use the transport protocol directly, which in most cases is TCP or UDP. Introducing changes at the transport layer means that the way applications communicate with it has to be modified, either by changing those programs or by "tricking" them to use the new protocol. This is possible in some cases by using techniques such as masking symbols in application binaries with LD_PRELOAD, but they might limit the number of the protocol's available features.

An example of an application that was modified to use a custom, multi-interface aware transport protocol (Protocol "Z") is presented in Figure 2.6.

One problem that cannot be alleviated, is that whenever new transport protocol is used it requires both sides of every connection to be modified. This can be seen in Figure 2.6.

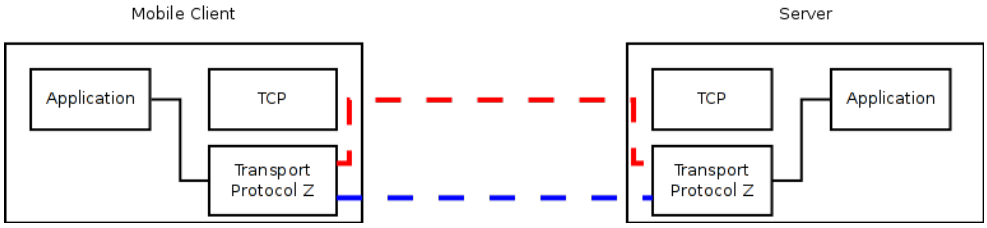



Figure 2.6: An application modified to use protocol "Z" instead of TCP

Both sides of the connection and the respective "applications" that are connected have to be modified and run the same protocol. In real life scenario one of the "applications" would be, for example, a web browser running on user's device. This side of the connection would be relatively easy to modify. However, the other application would be some random web server that the user decided to connect to. Modifying that side of the connection is, for large number of server instances, practically impossible. This issue is essentially the same as with application and session layer solutions.

In a lab environment, extending all participating entities does not pose a problem. In a real life scenario applications connect with hundreds and thousands of different remote machines all over the world. Modifying all, or even a substantial percentage of them, is simply impossible. Even the protocol that offers significant improvements to the user's experience would only work with a very limited number of services. The SCTP protocol, which does not really offer true mobile multi-interface connectivity, is the most popular of the protocols mentioned and has the largest number of working implementations. A relatively large number of people are involved in its development, more than in any other multi-interface transport protocol mentioned in this section. That said, even after 10 years of existence SCTP is not widely used. In turn, any modification to the original SCTP, is even less popular. None of the other transport protocols not based on SCTP are even close to SCTP's popularity. As the history of IPv6 (Internet Protocol, version 6) shows, introducing new network protocols at the global level, even as much needed as IPv4's (Internet Protocol, version 4) successor, is an extremely slow process.

Another limitation of new transport protocols is the fact that they need to work properly not only with remote servers, but also with all the network elements along the way. This means that all the routers and other network equipment having anything to do with transport protocol need to be aware of that new protocol, or at least not block or drop it. For example, some routers allow only selected traffic types through. It is likely that packets belonging to the protocol unknown to them will be dropped. For similar reasons, network address translation mechanism (NAT), which is used more and more often due

to a very limited number of available IPv4 address, would prevent those protocols from working - unless all the machines providing this service were modified as well.

Expecting modifications of not only remote servers, but also a number of intermediate devices in the network, including NAT routers controlled by an enormous number of different entities, is not realistic. Even if there existed significant pressure for adoption of a single, well established protocol, it would take several years for it to be broadly used. For these reasons transport-layer solutions are not practical in a short or even medium time perspective. Thus, any multi-interface architecture to be deployed on today's devices should not be based on the transport layer solution.

There is also another important aspect of transport layer solutions - they work on a per-application basis, making it difficult to manage available network resources between different programs. Each of the applications has its own list of connections and associated buffers, state, and parameters of algorithms responsible for retransmissions and congestion control. Each of them measures the network parameters, detects problems and makes decisions separately from others. In comparison, if all the traffic from all applications was controlled and scheduled by the same entity, the network resources could be assigned in a way that could possibly provide a better user experience. If multi-interface management is controlled on a per-application basis, then achieving the same behaviour is more difficult, involves a separate system that coordinates different transport layer connections and complicates the entire system and all the algorithms involved.

## 2.4 Network layer solutions

The next layer down the stack is the network layer, presented in Figure 2.7). This is the layer at which the Internet Protocol, the base of the global network, operates.

### 2.4.1 IP-in-IP tunnelling

In the network layer there already exists a well established standard that supports device mobility. It is the mobile IP [51] standard, which exists in two versions: For IPv4 [50] and IPv6 [38]. It allows hosts to move between different networks while preserving the IP address they can be reached at and maintaining existing connections. When the host leaves its home network, packets from other hosts are still routed to the home network where the special host, "Home agent", redirects the packets to the "Foreign agent" operating in the network the mobile host is connected to using an IP-in-IP tunnel. They are then delivered

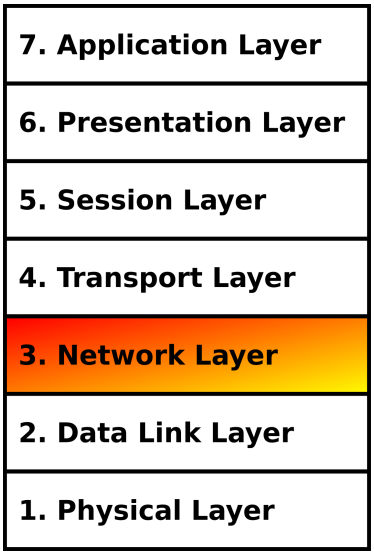



Figure 2.7: Network layer

to the mobile host. Packets sent back from the mobile host can either be tunnelled back to the home agent first, or sent directly to the corresponding host. This standard enables host mobility, but does not allow hosts to utilize multiple interfaces at the same time.

Ahlund and Zaslavksy [3] propose an extension to the mobile IP. It allows mobile hosts to register multiple interfaces at the home agent. This is meant to provide faster handover between different networks and facilitates selecting the best of available paths. The home agent still uses a single path to tunnel the data to the mobile host, but it can choose the path that offers the best parameters. In certain situations, packets from different corresponding hosts can be transmitted to the mobile host directly over different interfaces, possibly offering increased total bandwidth as well. Each corresponding host that is capable of exchanging data directly without going through the home agent selects the path that offers the best parameters between that host and the mobile device. As there is no coordination between them, it is possible that all such hosts decide to use the same path. This solution does not offer real bandwidth aggregation. Additionally, to provide multi-homing support not only do the mobile hosts have to be modified, the home agents and corresponding hosts must be modified as well.

A number of publications propose the real bandwidth aggregation in the network layer. Phatak and Goff [52] and Phatak, Goff, and Plusquellic [53] present one of the first attempts to provide bandwidth aggregation over multiple IP links. They propose a solution that does not require modifying applications. Their implementation intercepts IP packets before

they are sent out of the network interface and optionally encapsulates them inside new IP packets using IP-in-IP tunnelling. The newly created packets can then be sent using a different interface than the one the original packets were supposed to be sent over. At the remote host the reverse operation happens. Tunnelled packets are extracted and delivered. To achieve that, generated IP packets that carry tunnelled data use the "IP-in-IP" protocol identifier (the same that Mobile IP uses for tunnelling). The tunnelling operation is performed only when the IP packet is sent over alternative interfaces - if the interface is not changed, the packets are sent without tunnelling. The multi interface management is accomplished by using unused fields in the IP header.

This proposal has several problems. First, it requires both endpoints to be modified, which is the same limitation as the transport layer solutions. Also, the authors make a questionable assumption, which is that the end-to-end delays over different links are either bandwidth or latency dominated at the same time. They also assume that all available links have very similar latencies, which in many real-life cases is not true. Additionally, this proposal does not address issues related to dynamically changing the properties of available links, which is very likely to happen in real life. Another problem is the proposed solution for TCP-related issues which consists of modifications to the TCP stack itself, which is no longer a network layer solution. Even though the authors created a working implementation of the algorithm, it is more proving the concept and a source of results for analysis in simple, fixed network scenarios than a candidate for a working, real life solution.

Chebrolou and Rao [20] and Chebrolou, Raman, and Rao [19] present another network layer solution. Chebrolou and Rao's earlier publication introduces the architecture and the later Chebrolou, Raman, and Rao piece evaluates it in more depth and also examines the effect on TCP's behaviour. This solution, just as Phatak and Goff's [52], is based on IP-in-IP tunnelling. Instead of requiring both ends of each connection to be modified, however, a network proxy is used. This design is based on the assumption that the most limited links (in terms of bandwidth and latency) are close to the mobile device. Providing higher bandwidth and lower latencies inside the wired internet is much easier than over wireless links, close to mobile devices. This design provides bandwidth aggregation between mobile client and the proxy. Data sent between these two points is tunnelled over IP packets with the protocol type set to "IP encapsulation". The original packets are extracted at the proxy. After required reordering they are sent to the final destination which does not require any extra processing. This means that modifications can be limited to the mobile device and a single proxy. The servers providing services for mobile clients do not need modification. For them, the traffic looks like it was generated by a regular device with a single interface. On the mobile device the modifications are minimal. IP packets have to

intercepted, but this can be done transparently to transport protocols and applications, requiring changes only in the operating system. Because of these properties, even though this proposal is based entirely on simulations, it would be possible to implement and deploy a similar solution in today's internet.

The packets' actual scheduling is based on an expected delay incurred on individual links. The algorithm measures available bandwidth on all the links using the packet pair spacing method, employing the available payload. Each packet is scheduled to available links according to its earliest possible delivery; where link bandwidths, latencies, and queue sizes are taken into account. On the remote side of the tunnel, packets are buffered to restore their original order and then released to the final destination. That buffering would cause TCP running on top of that tunnel to experience undesired bursty traffic. For this reason, Chebrolou, Raman, and Rao propose adding delays between TCP data or TCP ACK packets as they are released from the buffer to the network.

This proposal seems like a good candidate for a mobile, multi-interface architecture. However, the whole evaluation was performed entirely using simulations. In the scenarios, links with different bandwidth ratios were considered, but the problem of different, varying latencies is not. Instead, the algorithm measures the path's latency at the beginning and assumes that it does not change. The reasoning behind this is that wireless base stations should be considered an extension of the internet and the latencies of paths in the internet change slowly. Delays incurred by network problems and lower layer retransmissions in the wireless environments due to congestion were ignored. Our preliminary tests show that this simplified approach does not yield satisfying results. On the contrary, latencies observed on wireless links change frequently, causing reordering and much longer packet delays. As a result, transmission rates of TCP flows often drop below the bandwidth that would be observed when only a single link was used. These delay problems cannot be ignored and are very significant at higher data rates. Improving wireless technologies offers increasingly higher data rates. TCP operating at those rates would be more sensitive to larger, varying delays caused by changing latencies on different links used in parallel.

Another, similar proposal is described by Lin *et al.* in "Dynamic Bandwidth Aggregation for a Mobile Device with Multiple Interfaces" [44]. It also requires network proxy and tunnels data between the mobile client and the proxy before the original packets are extracted and sent to their destination. The packet scheduler operates based on the link bandwidth, size of the packets, and lengths of packet queues. It does not consider link latencies. This solution is evaluated solely using simulations and compared against very simple algorithms, predefined link ratio scheduling, and packet striping. The authors also assume that TCP data packets have 1024 bytes. This does not reflect any real traffic distribution, in which most of the packets have one of few "typical" sizes, none of which is

1024 bytes [21].

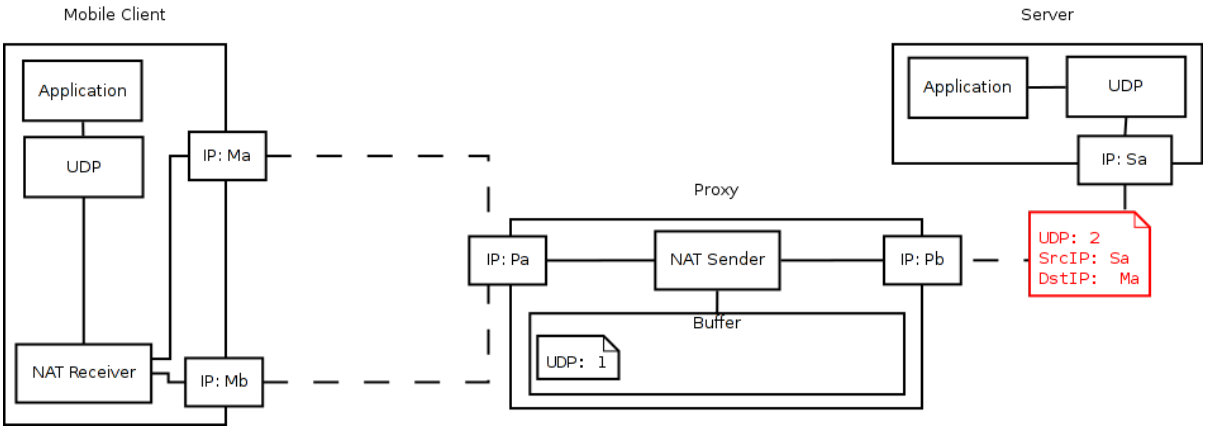
### 2.4.2 Network address translation (NAT)

Unlike some of the previously described proposals, a solution presented by Evensen *et al.* [24] was implemented and evaluated using a network testbed. The design, however, is similar. It requires a network proxy that participates in data transfers between the regular server and the mobile client. Thanks to the proxy, the server does not have to be modified and does not know about the multi-interface connectivity that takes place between the mobile client and the proxy. The difference between this and earlier solutions is that it does not perform tunnelling. The data that arrives at the proxy, is split between available interfaces. IP packets following the primary path are not modified at all, and those that are sent over the alternative path have their headers modified using a network address translation (NAT) mechanism. In the mobile client, the headers of those packets are modified again and delivered as if they arrived over the primary interface. This requires the proxy to use both the special kernel module and the userspace application that makes scheduling decisions. On the mobile client's side, simple IP table redirection is sufficient.

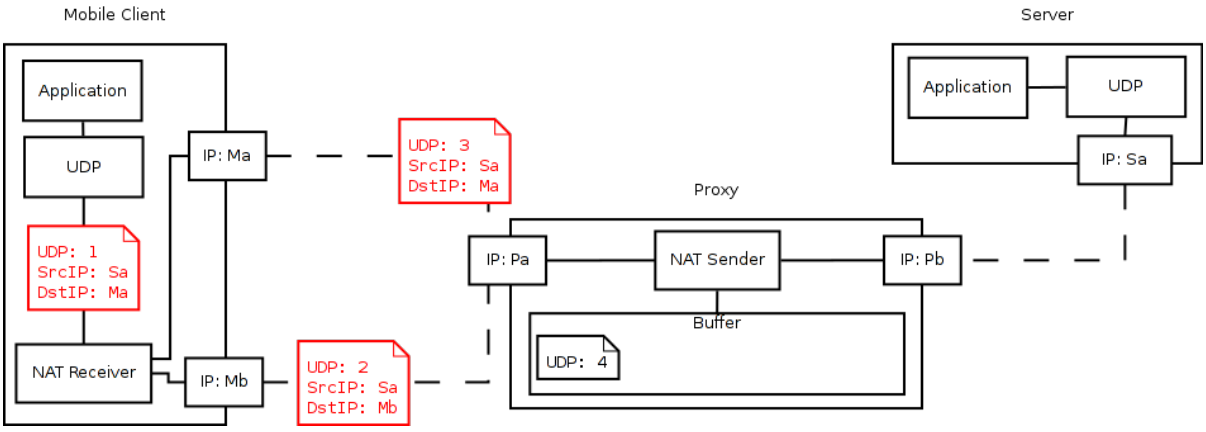
This mechanism is presented in Figure 2.8. The sender sends all the UDP packets to the same address on the mobile device, denoted with "Ma". The proxy modifies some of them to be delivered to the second interface of the client - the one with address "Mb". After the packets are delivered to the client device, their original destination address is restored to the original, "Ma" value. This way the application is not aware of the fact that traffic splitting took place.

The proxy scheduler application constructs a "send vector" based on link bandwidth estimation. In this vector the links with higher available bandwidth occur more often than those with lower bandwidth. The vector is then used by the kernel module for scheduling packets for different links. No latency values are considered while creating this vector. Instead, the difference in delays between available links is equalized by adding delay to packets to be sent over links with lower latency. This minimizes packet reordering (and the amount of buffer space the client needs to restore the right order).

The latency measurements are performed by sending probes over available links and estimating the client's incoming data rate. The authors of this proposal are considering modifications to the design to use the packet pair spacing method. The current design allows only for bandwidth aggregation of the downlink data - packets from the client are only sent using the primary interface.




(a) Server sends all the packets to Ma




(b) Some of the packets are modified to be delivered to Mb

Figure 2.8: Solution based on Network Address Translation

Even though the "delay equalizer" presented in this design does limit the amount of packet reordering the client experiences, it also introduces additional delay as presented in Figure 2.9. When multiple links are used they are likely to have different latencies. Under normal conditions, without equalizing those latencies, sending packets over different interfaces results in receiving them on the other end of the link in a different order. When both of these packets belong to the same flow the order should be restored by buffering packets received out of order. When the equalizer is used, all packets, even those sent over links with lower latency, will experience the total delay of the slowest link available.

The authors of this proposal do not examine the TCP traffic, but increased delay would most likely negatively affect TCP's behaviour. Several problems with higher delay are described by Allman *et al.* [4]. The main issue mentioned in this study is the TCP connection's limited maximum throughput, but also that a longer time is required to achieve that maximum throughput in the slow-start mode. Furthermore, a longer time is required for packet loss recovery. This behaviour is undesired. In the mobile environment the maximum TCP throughput might be somewhat less problematic due to lower bandwidths. The length of the slow-start phase, however, is very important, especially for short-lived TCP connections that, when experiencing increased RTT (Round-Trip Time), could spend the whole time in the slow-start mode.

On the other hand, when the equalizer is not used, packets arrive at the proxy in a different order and have to be buffered to restore the original order, and released to the receiver in bursts. The ACK packets generated by the receiver in response to those packet bursts will be grouped as well. This behaviour is similar to ACK compression observed and presented in Zhang, Shenker, and Clark's article [71], which negatively affects TCP's self-clocking mechanism and its RTO (Retransmission TimeOut) and RTT measurements. ACK compression was also observed in a study lead by Chakravorty [18], as a result of using TCP over GPRS (General Packet Radio Service) links. To deal with these issues additional techniques such as packet pacing or ACK pacing (for TCP traffic) could be used. Any of those techniques would increase the experienced latency even further, aggravating the problems mentioned above.

In the Evensen led study [24], TCP traffic is not examined at all, so no observations of this equalizer's effect on it are presented. Even with the UDP packets, clearly the lower latency of some links is wasted to compensate the higher latency of others.

The Evensen team's evaluation of the solution is rather limited. All their experiments are performed using only UDP traffic, and they focus on packet loss and reordering, but mostly as a factor of the amount of buffer the client requires to be able to restore the proper packet order. TCP traffic (and all TCP-specific issues related to introducing delays

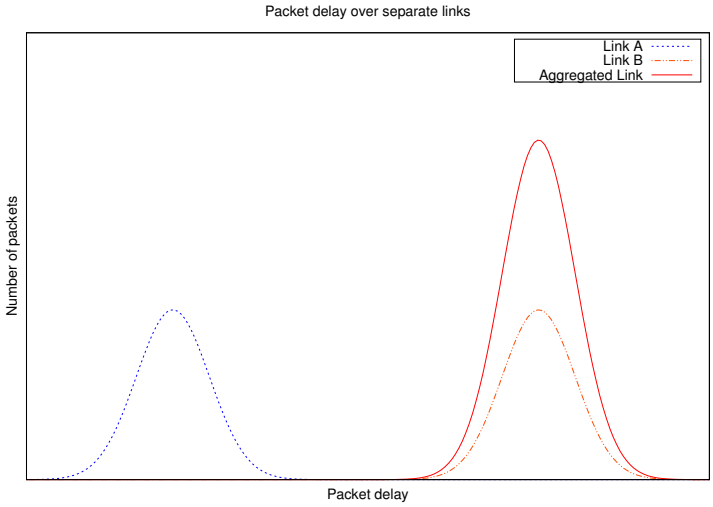



Figure 2.9: Packet delays over individual and aggregated links

or reordering and their effects on TCP's algorithms) is not considered at all. Also the example presented to justify the equalizer does not seem very realistic. Their proposal describes two links, where the one with higher bandwidth also has higher latency. In some scenarios, for example when satellite links that offer high bandwidth and high latency at the same time are used, this might be correct. In general, in real life scenarios the links that offer higher bandwidth usually have lower latency.

This solution contains some interesting ideas, but the evaluation is limited and the implementation is very simplified. It also lacks some of the features required for mobile multi-interface connectivity architecture to be deployable on today's devices - it does not handle network hand-over, or any issues related to link failures or switching the primary network, which is likely to happen in a mobile environment.

### 2.4.3 Packet manipulation

Tsao and Shivakumar present a completely different approach [66]. Authors call their solution a "layer 3.5" solution, as it uses Linux's netfilter mechanism. Instead of tunnelling IP packets, it captures, processes, and, if necessary, generates TCP packets as they traverse the network stack. The main focus of this proposal is targeted at TCP, but UDP-based experiments are also performed. They call their solution a "super aggregation", compared to a "simple" bandwidth aggregation, and describe techniques that, according to them,

offer better connection parameters than simply combining bandwidths offered by individual links. This solution is implemented entirely on the client's side, with no proxy in the network, and no modifications to remote endpoints. The argument against simply combining available bandwidth is based on the fact that, for example, adding a relatively small bandwidth offered by 3G networks to much larger bandwidth offered by Wi-Fi networks does not improve the overall experience. This assumption might be true in very specific cases. However, in contemporary real life scenarios it is usually not valid. The bandwidth Wi-Fi technology offers is usually much larger than the maximum throughput of the internet connection that the given wireless network uses. This limits the effectively available bandwidth to much smaller values. At the same time, bandwidth offered by modern 3G networks is significantly larger than it is described in the paper.

Tsao and Shivakumar base their ideas on several observations. One of which is that one source of TCP's performance problems in Wi-Fi networks is related to ACKs travelling back to the sender. To deal with this problem, the proposed architecture offloads those ACKs to the 3G network, improving the situation in the Wi-Fi network for the actual data packets. This offloading is done only when the TCP connection has a large congestion window. This way the increased delay of ACKs sent over the 3G network would not affect the TCP's behaviour too much.

Another idea is to minimize the impact of "TCP blackouts", as they call the temporary stalls in Wi-Fi network. Normally when the Wi-Fi network stops sending packets for a while due to lower layer congestion and retransmissions, TCP times out and decreases its congestion window. The solution is to send a packet (over the 3G network) that advertises zero receiver's window size to the sender to temporarily "freeze" the transmission, without affecting the congestion window.

Other ideas include retransmissions of TCP packets lost in the Wi-Fi network over the 3G network and application layer "replays" for connections using known protocol types.

Although the idea of trying to use available interfaces in a better way than simple packet striping is essentially good, this publication fails to propose a feasible solution. It is based on invalid assumptions of typical link parameters and the locations of links' bottlenecks. This architecture tries to require only client modifications, but it is based on the assumption that it is able to send IP packets with mismatched source addresses (IP packets with Wi-Fi interface's source address are sent with no modifications over the 3G network). In reality, in most network and router configurations it is simply not possible. Also, performing "application layer connection replays" at the network level is questionable.

### 2.4.4 Limitations of network layer solutions

The biggest limitation of network layer solutions is that they require an additional network device - the proxy. That introduces a single point of failure. If the proxy crashes, or if the network it is connected to experiences problems which make the proxy inaccessible, all the connections between mobile devices and other hosts are interrupted.

Another problem is that data packets, instead of being sent directly between the network server and the mobile client, now have to go through the proxy. This path is usually longer than the direct one, and is likely to have a higher latency. This might negatively affect data connections, especially TCP's performance.

Another problem is related to the fact that the solutions that require any form of packet encapsulation introduce additional protocol overhead.

Those limitations can not be completely avoided, but can be limited to some extent. Careful protocol design that adds minimal overhead, strategic proxy placement to minimize the additional latency, and replicating and load balancing the proxy system to minimize the chance that proxy becomes inaccessible should limit the "cost" of the solution. It might also be possible to detect problems with the proxy from the mobile device and disable the entire service completely. This would interrupt all existing connections and prevent the device from utilizing multiple interfaces, but at least network connectivity would not be completely broken because of problems in some remote location.

### 2.4.5 Network layer solutions - summary

Network layer feels like a feasible level at which multi-interface bandwidth aggregation could be located. The IP protocol is the base of today's internet. It offers a single, common addressing domain (except for private networks) so that different network technologies, usually deployed by separate providers, can be combined (contrary to the link layer solutions presented below). At the same time, the way the IP protocol works is very simple. Protocols at the higher layers are designed to operate by sending and receiving simple packets that do not offer any sophisticated features. This makes it easy to modify the way these packets are exchanged in the network in a way that is completely transparent to higher layers and yet has some unexpected features. The solutions that use network proxy introduce a single point of failure, but in exchange they offer the ability to communicate with any server without modifying it. This is a huge benefit and makes it possible to deploy an architecture offering multi-interface connectivity without modifying large parts

of existing internet's infrastructure. Network layer solutions are transparent to the applications running on the mobile client as well. Even though the operating system running on the device still needs to be changed and/or equipped with additional applications, the myriads of user applications do not have to be touched at all.

Another advantage of the network layer solution is that at this level packets belonging not only to different flows and different transport protocols are exchanged, but also packets sent by different processes. In the transport layer solution, each instance of the system was responsible for packets belonging to one flow within a single application. Here, multiple flows are exchanged at the same level, which makes it possible for the scheduler to employ more sophisticated algorithms and better allocate available network resources.

At the network level (as opposed to the individual devices), the IP protocol is the point at which all network connectivity seems to converge. There are a number of different transport and application layer protocols, but most of them run on top of the IP protocol. At the same time, there are a number of lower layer protocols and networking technologies used today. Most of them allow IP packets to be exchanged. With the upcoming LTE (Long Term Evolution) standard, the telecommunications world is also switching to the IP protocol, not only to support it, but to use it as the main way for exchanging data in various networks. This is a very strong argument, possibly even stronger than those mentioned above, for introducing architecture offering multi-interface, mobile connectivity at this layer of the protocol stack.

To the best of our knowledge, a network layer solution that offers all the required features, is deployable, and could be used in today's internet has not been proposed. However, network layer offers a great potential and we believe that a successful solution can be built at this level and that it would certainly contain some of the ideas presented in this section. Just as any other solution, this approach would have its own limitations, but we believe that the offered benefits could significantly surpass the costs.

## 2.5 Lower layer solutions

Bandwidth aggregation can be performed below the network layer as well - in the lowest two layers of the OSI stack (Figure 2.10). Adiseshu *et al.* [2] present "A Reliable and Scalable Striping Protocol". They describe an algorithm for scheduling packets to be sent using available links. They also mention the limitations of link layer striping methods. One limitation is that the algorithm might not be able to modify the packets that it sends. This is because the packets cannot be modified and there might not be a space to add

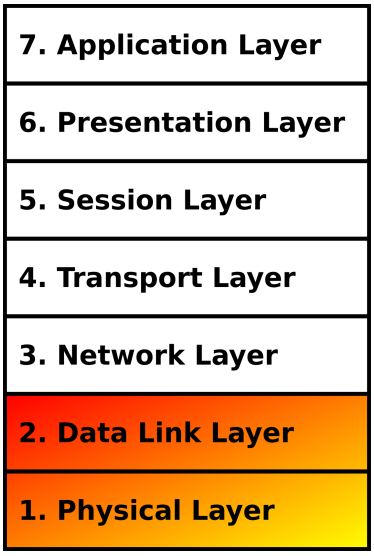



Figure 2.10: Lower layers

any information to them. Some of the link layers use fixed packet sizes, but even those that support variable lengths, such as Ethernet, have limits. If the packets being sent are too close to this limit no information can be added. Without adding extra data to these packets, all the links have to be synchronized to achieve in-order delivery. They describe "quasi-FIFO" delivery (FIFO - First In First Out), which means that generally packets are delivered in order, except for periods of time when synchronization is lost and needs to be restored. This synchronization requires lower level protocols to provide distinct "codepoints" to mark synchronization packets that are injected. When running over an Ethernet network, these packets have a different value set in the packet type field in the header. They also discuss implementing the same set of algorithms working in the transport layer, where modifying data packets is possible, but this is not the link-layer solution anymore.

Snoeren proposes another solution [61]. It is designed to operate in a wide area wireless network environment and offer a higher bandwidth by splitting the traffic between several physical links. However, one of his assumptions is that the wired part of the path that packets sent over different wireless links take is the same, or, at least, is isolated from cross traffic. While this assumption is certainly satisfied when several wireless links of the same carrier are used, it is not true for a general multi-interface scenario. Requiring the mobile device to use links belonging to a single provider would be very limiting. Also, mobile devices equipped with multiple interfaces usually offer interfaces of different types. Even

if both network types are offered by the same provider, their physical paths are likely very different. The design also specifically ignores the effect of device mobility.

There exists even an IEEE (Institute of Electrical and Electronics Engineers) standard for aggregation in the link layer, called Ethernet bonding [27]. It requires all physical links to be part of the same logical switch. This could mean several physical switches which are configured as a single logical device. Also, all the links being aggregated are required to work with the same transfer rates. The standard was designed to offer improved bandwidth and fault tolerance by multiplying a number of physical connections working in parallel, and the aggregation happens in the Ethernet layer. In the mobile environment links with completely different characteristics are used.

Another standard solution in the link layer is Multi-link PPP (MLPPP) [60]. It is a feature of PPP (Point-to-Point Protocol) which is used over a variety of physical networks, such as phone lines, serial cable, and some wireless networks. This approach has the same limitation as other link-layer solutions; both endpoints of each link have to in the same physical location. In the mobile internet scenario, available interfaces will likely operate in separate link-layer spaces, preventing them from using solutions such as MLPPP. It is, however, possible to run PPP protocol over another protocol. For example PPPoE (Point-to-Point Protocol over Ethernet) is a method of sending PPP packets over the Ethernet network. It would be possible to build a solution that would work using MLPPP protocol sent over protocol from a higher network layer. This would allow all the remote endpoints of all PPP links which are part of the connection to physically be in the same location. However, that would incur significant additional overhead.

### 2.5.1 Limitations of lower layer solutions

In general, link layer solutions might be feasible when multiple physical links of very similar characteristics are used between the same two entities. The same type of networks are used, with preferably similar properties, and both endpoints are within the same domain of control at the link level. In the mobile multi-interface network scenario, links available for the device to use are usually very different. A wireless network has different characteristics and properties than a cellular data network. Also, those network's properties change over time without any guarantees of bandwidth or latencies. Synchronization between those two network types would be very difficult. In most cases, the remote endpoint of each of those links at the link level is not only serviced by different networking technology, it lies within a different administrative domain and in a different physical location. A cellular data connection's remote endpoint is within a carrier's network infrastructure. The wireless

link, on the other hand, is connected to networking infrastructure of the ISP (Internet Service Provider) it uses, which in most cases is not even the same company. There is no relation or connectivity between those two entities at the link layer, so it is impossible to use link layer striping in this scenario. Also, even if several separate networks were connectable in the link layer at the remote location, to establish a link-layer bandwidth aggregation solution would require the provider's involvement. This could sometimes be possible, but it is a limiting requirement.

Lastly, the link layer solutions usually just perform the task of striping the data between available physical paths. Any other features, such as traffic type based scheduling, flow detection and flow based scheduling, are easier to implement at the higher layers.

## 2.6 "Abstract" solutions

The previous sections described existing research divided by the network stack layers it applies to. Kumar and Golash's proposal [41] ignores implementation details, and focuses on an analysis of theoretical link load balancing. It builds on Bennett, Partridge, and Shectman's [12] and Blanton and Allman's [14] observations, which analyze the consequences of packet and ACK reordering. As expected, both packet and ACK reordering should be absolutely avoided. Kumar and Golash conclude that packet-based aggregation is ineffective for high bandwidth links with different characteristics. In that scenario packet order needs to be restored, requiring potentially large buffer sizes. It also needs the use of sequence numbers, which would increase data overhead. Instead, flow-based scheduling could be performed. In this approach, each packet is assigned to a link based on the flow it belongs to. The idea is to send all the packets being a part of a single flow over the same link. Obviously, in case there is only a single flow in the system, this solution does not offer increased bandwidth. In an environment where several connections and applications are working at the same time, combined bandwidth of the flows could be larger than any of the rates offered by individual links, without introducing reordering issues.

Kumar and Golash consider different methods for flow identifications. The simplest one, in which the list of flows is stored, is impractical due to high memory and computational requirements for maintaining this list when there are many concurrent flows. Instead, they propose invariant-based flow identification and hashing for link assignment. Instead of static hashing, they propose using dynamic hashing - where flows are assigned (using the hashing function) to "bins" instead of the links. If the link assignment is unbalanced, the bin - link mapping can be changed so the hashing function does not have to. They also describe the use of a Bloom Filter (see Bloom [15]) combined with two hashing functions

for better flow to their link assignment to avoid assigning new flows to already congested links.

This proposal is purely theoretical and does not present results of any simulations or experiments. Obviously it cannot be a candidate for a real, deployable architecture, but it contains several observations and interesting ideas that could be taken into account while designing the multi-interface architecture for mobile connectivity.

## 2.7 Other relevant research

Publications that are the most relevant to this work are those that talk about multi-interface connectivity and bandwidth aggregation. However, there are a number of other papers that focus on issues not directly related to multi-interface connectivity, but are still important and should be considered in this work.

### 2.7.1 TCP modifications

Several papers mention, or even use ideas from two versions of the TCP protocol: TCP Westwood [47] and TCP Westwood+ [30]. TCP Westwood introduces modifications to the TCP's sender side algorithms. No modifications to the receiver's side are needed. In the traditional TCP, packet loss is interpreted as a sign of congestion, causing the TCP sender to slow down. In wireless networks packets are often lost (or significantly delayed causing TCP to time out) due to the wireless medium's nature. In this scenario TCP slowing down is undesired. Unfortunately, standard TCP implementations interpret these symptoms as a congestion, and slow down. This is the source of problems with TCP over wireless links. TCP Westwood's goal is to be able to distinguish between packet loss due to wireless channel loss and loss due to actual congestion. To do that, TCP monitors the rate at which data is acknowledged with ACK packets. Based on those measurements, the congestion window is consistent with the effective bandwidth, instead of blindly halving the congestion window every time a packet is lost. This mechanism is called rate-based congestion control and is used by several, mostly SCTP-based, algorithms. Evaluation of this TCP implementation shows significant gains in most wireless scenarios. The newer version, TCP Westwood+, tries to further improve this behaviour. It attempts to deal with ACK compression that negatively affects bandwidth estimation.

### 2.7.2 Commodity hardware

As previously mentioned, one of the assumptions behind the multi-interface architecture for mobile connectivity is that it should be easy to deploy, preferably using only commodity hardware. The limitations of such architecture for data aggregation can be examined once it is built. However, some research in the area of using commodity hardware in a network routing context already exists. It is completely unrelated to multi-interface or mobility scenarios, but it can help assess if building such an architecture on commodity hardware is feasible. It also suggests solutions that would likely provide satisfying results (in terms of network data processing, not the actual data aggregation).

The Route Bricks software router architecture is presented by Argyraki *et al.* [5] and by Dobrescu *et al.* [22]. The former proposes an architecture for building software routers, the latter presents detailed design and implementation procedures, as well as evaluation of this solution. Even though it can not yet compete directly with hardware routing solutions, it is surprisingly close. In this publication, individual hardware components and their limits are analyzed. It presents recent advances in commodity hardware that make the concept of software routing closer than ever. The difference between the traditional architecture and the new one is the way individuals CPUs (Central Processing Units) access memory. Instead of going through the same, single Front Side Bus, groups of processing cores have separate, designated memory controllers. This allows for much faster memory access. At the same time, it uses network controllers with separate, hardware queues which are allowed to increase parallelism and reduce locking.

Even though this software solution is not as fast as hardware routers, it demonstrates a huge potential in modern commodity hardware for very fast, parallel I/O (Input/Output) processing. It is even more promising considering that software routing is designed for working inside very fast, wired networks. In the bandwidth aggregation scenarios data rates are usually smaller, even considering the large number of clients using the same infrastructure simultaneously. Also, since in Route Bricks architecture the bottlenecks are mostly I/O operations, the fact that data aggregation architecture is likely to perform more computing operations than a basic router might not be an issue at all.

### 2.7.3 Indirect TCP

Another research area, seemingly unrelated to multi interface connectivity, involves a technique called “Indirect TCP” [8]. It is used over a wireless connection between the mobile device and a “Mobile Support Router” that is connected to the wired network. When

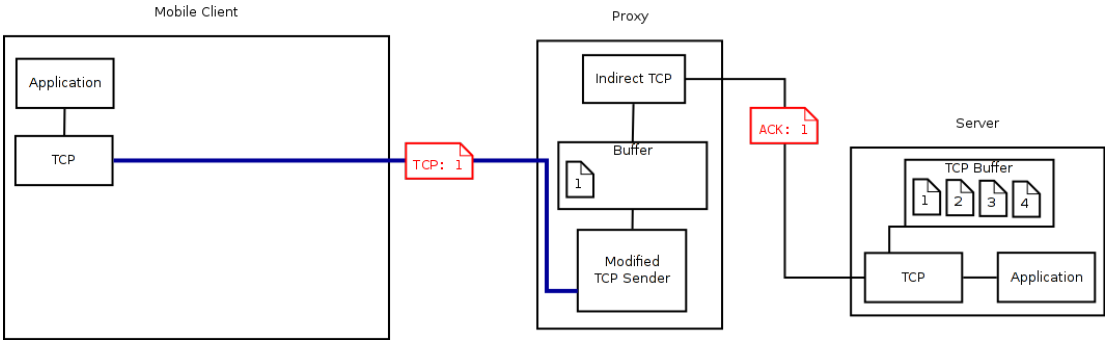
the mobile device switches between different wireless cells, the IP packets of existing connections should be routed using a different path. Mobile IP facilitates this re-routing. However, during the handover, the TCP connections experience packet losses and degrade their rates significantly. The same issue happens for "regular" wireless packet losses, but is even more critical during hand-overs. This proposal uses existing IP mobility solutions and focuses on improving the TCP's behaviour. When I-TCP protocol is used, the mobile device, instead of opening TCP connections directly to the end host, opens an I-TCP connection to the device connected to the wired network called Mobile Support Router (MSR). This router, in turn, establishes the TCP connection with the remote host on behalf of the mobile device. From the remote host's perspective, it behaves like a regular TCP connection with the application running on the mobile device. However, data packets sent to the mobile device are acknowledged by the MSR long before they reach their destination. After that, the MSR is responsible for transmitting (and retransmitting when needed) that data to the mobile device using I-TCP protocol.

Standard TCP, which is used between the MSR and the remote host, operates over a wired network which, in most cases, offers parameters that are significantly better than those of wireless networks. When the regular TCP is used over wireless networks it experiences performance degradation due to undesired behaviour of the TCP congestion algorithm when packet losses not induced by actual congestion are experienced. In this scenario, however, the protocol that operates over the wireless link is different. It can be aware of the wireless nature of the network and it can also use event notifications for events such as connection problems or cell hand-overs. The protocol can then react better to conditions which are typical to wireless environments.

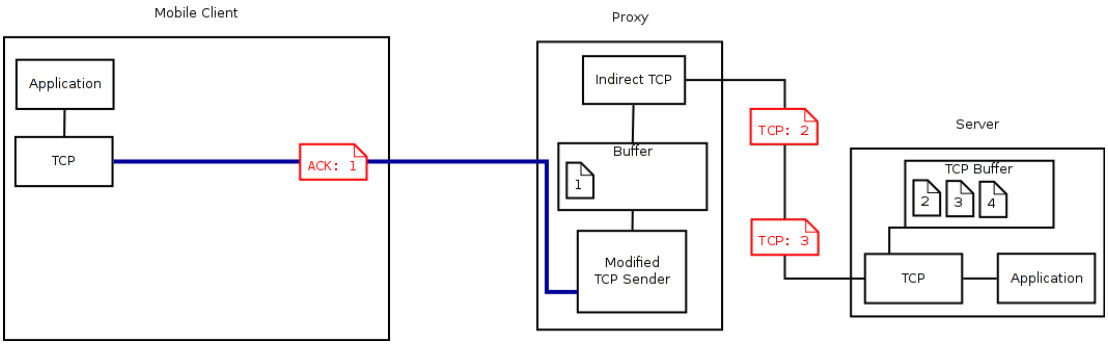
The role of the MSR is to "pretend" to the remote, fixed host that the mobile device actually received the data packets and it delivers that data later using protocol better suited for that particular part of the path without experiencing all the problems related to running TCP over wireless, flaky links.

This behaviour is presented in Figure 2.11(a) and Figure 2.11(b). In Figure 2.11(a) the first TCP packet has not yet been delivered to the mobile device (it is still in transit). However, the proxy server has already buffered it and sent an ACK packet towards the original TCP sender.

In Figure 2.11(b) the first ACK packet has been received by the original sender, which removed the first TCP packet from its buffer. It still remains in proxy's buffer, since it has not yet received the ACK packet from the mobile device. Even though the ACK from the final receiver is still in transit, the TCP sender already sent TCP packets 2 and 3. If the system was not modified, it would be still waiting to receive the first ACK packet from the




(a) Sender sends the first packet




(b) Sender sends subsequent packets

Figure 2.11: Indirect TCP

client.

The study's authors admit that in this scenario the semantics of TCP acknowledgements are violated. The sender receives acknowledgements for data packets that have not yet reached their destination. The study's authors argue, however, that this is not critical because most TCP applications have application-level acknowledgements. According to them, using I-TCP does not yield weaker end-to-end semantics of TCP, provided there are no MSR failures or long disconnections.

Another proposal that is designed specifically to improve performance of TCP operating over GPRS links is presented by Chakravorty *et al.* [18]. They examine TCP connections' behaviour over GPRS links, which usually has high latency, but also tends to stack packets together. They observe several problems. First, TCP's slow start algorithm prevented some short lived connections from even reaching the available bandwidth. This would not affect long lived file transfers, but during typical web browsing most HTTP connections are

very short. Another characteristic of GPRS links is grouping small packets together. This caused TCP's ACK packets to arrive in groups which has a negative impact on TCP's self-clocking mechanism and its retransmission timeout measurements. Another problem they observed was that the second TCP connection opened some time after the first, long-lived connection, was experiencing very high round trip time and behaved poorly.

In this design, TCP connections are "split" by the proxy device, located close to the wired-wireless boundary (on the "wired side"). When the proxy receives TCP data packets for the mobile device on the wireless side of the proxy it acknowledges them immediately. From this moment, the responsibility for actually delivering those packets lies on the proxy itself. This is very similar to the design presented in the I-TCP study [8]. In this case, however, no special protocol between the proxy and the mobile device is used. One of the assumptions is that neither the mobile device, nor the remote server, needs to be modified in any way. Instead, the proxy tries to deliver packets it received from the remote server (and already acknowledged) to the mobile device using standard TCP protocol. However, the proxy does not behave like the standard TCP sender. Slow start algorithm is not used at all. Instead, they monitor the bandwidth-delay product of the link to the mobile device and use it for calculating the congestion window. This information is shared between all TCP flows which is similar to the idea of "Congestion Manager" Balakrishnan, Rahul, and Seshan presented [9].

These scholars identify two causes of packet loss. One is the radio loss, when the GPRS link layer gave up trying to deliver packets, the other is caused by long link stalls during handovers. Selective ACKs and special lost packet counters are used to detect and aggressively retransmit packets lost due to radio losses. Also, the congestion window is not decreased using standard TCP algorithms, but kept at the same level until conservative timeouts are triggered which indicate extensive link blackout or a handover. When this happens, the congestion window is set to 1 to avoid overloading a link that is likely unusable. Once the next ACK packet is received the congestion window is restored to its original size.

On the "wired side", the proxy controls the amount of data it receives from the sender by window advertisement using a method similar to the one Kalampoukas, Varma, and Ramakrishnan describe [39]. At the beginning of a new TCP connection data is accepted as fast as possible. Later, if required, advertised window size changes in a smooth way to avoid advertising a zero window size.

This design is evaluated using both simulations and Linux-based implementation, showing improved bandwidth utilization, fairness, and error recovery time.

Both "split TCP" designs described above improve TCP behaviour over flaky wireless

links. However, they both break TCP semantics. Acknowledgements are no longer end-to-end and when the server receives an ACK packet it does not mean that the receiver actually received the corresponding data packet. Instead, some other entity along the data path received it and is now responsible for its delivery. However, if that entity or the actual receiver crashes, or the network breaks, the receiver might never see some packets the sender received delivery acknowledgements for.

Xie *et al.* [70] present a "Semi-Split TCP" that is supposed to improve TCP's behaviour over a wireless network in a way similar to I-TCP and Split TCP, but without violating end-to-end ACK semantics. This design is very similar to previously described indirect TCP solutions and uses a proxy. The difference is that the proxy never acknowledges the data that the ultimate receiver has not acknowledged. At the same time, it tries to "shield" the remote sender from any wireless-related problems that might be happening between the proxy and mobile device. It does this by sending ACK packets to the sender at a slower rate than it acquires them from the receiver. Over time, there are a number of packets that have been received by the mobile device (and acknowledged at the proxy), but for which ACK packets were not yet sent to the sender. If there is a problem with the link on the wireless side of the proxy, even though the receiver temporarily stops responding with ACKs, the proxy keeps sending those "unused ACKs" it has accumulated thus far.

The evaluation shows that even though this design improves TCP behaviour, it does not perform as well as I-TCP or Split TCP techniques. the authors claim that this is a justified price for preserving TCP's end-to-end semantics. However, they fail to provide a convincing example of an application or a scenario where violating those semantics would be a problem. The argument they make, that some mission critical or financial applications might use TCP ACKs for verifying data's delivery is questionable. Most applications (especially financial or mission critical ones) use some sort of application-level acknowledgements which are sometimes even strengthened with checksums. Even when regular TCP is used, the fact that the sender receives acknowledgement of a data packet it sent does not mean that the receiving application actually acquired the data. Instead, it only means the TCP stack of the remote system received the packet. If the application has crashed, it might never see the data. In that situation, obviously, the TCP connection is broken, but the same thing happens when the proxy that "pre-acknowledged" the data crashes; the TCP connection is interrupted. The difference between those two scenarios is that the entity that acknowledged the data's reception is closer or further away from the actual application that consumes the data. In both cases, the fact that the sender received the TCP ACK packet does not mean that the relevant application actually received the data. Given that the semi-split algorithm is much more complicated than other proposals and that the improvements to TCP's behaviour are smaller, it is reasonable to use one of the other

solutions even though they seemingly "break" TCP's semantics.

## 2.8 Energy usage implications

The other area of this research is exploring the energy consumption implications of using multi-interface connectivity system on mobile devices. The limited battery life of mobile devices is a very serious issue recognized by a number of researchers that published various findings in this area, as well as a number of ways to improve energy consumption. The general approach leading to improvements of the efficiency of network operations on mobile devices is to increase the intensity of those operations (the data rate and the length of individual transactions), while decreasing the frequency of these operations. There are studies that analyze energy consumption of different networking technologies, like Wi-Fi, 2G (2nd Generation wireless telephone technology) and 3G, and the effect of "energy tail" associated with some of them [68, 10, 58], as well as energy consumption at different data rates [68]. Some of the work studies possible improvements that can be achieved by simply delaying data transmissions until different connectivity conditions are available (and trying to predict when that is going to happen) [42, 54, 57], or by grouping data together to perform bigger operations, but less frequently [10]. Some of the authors even suggest using aggressive data pre-fetching, to minimize required frequency of network operations [10]. There is also some work that tries to improve energy efficiency by modifying the other part of the wireless link, the one that operates on the edge of the wireless network, by introducing a proxy that delays data transfers [23]. Some of existing work simply analyzes how users interact with their mobile devices with respect to battery usage [48, 25, 67]. These studies suggest that users differ a lot, but agree on the fact that battery life remains a significant problem in the mobile environment. Finally, there are studies that focus on the methodology of energy measurements, and the size of the parameter space [7, 49, 1].

The existing work studies users behaviour and energy consumption and efficiency of different network types in various scenarios and use cases. However, there is no prior work on energy consumption and efficiency in scenarios when multiple interfaces are used simultaneously, for active data transfers. Whether this scenario is even feasible in real life remains an open question, and this research aims to explore this area and answer a question, whether practical multi-interface connectivity systems make sense on mobile devices.

## 2.9 Summary

In this chapter several proposals at different network stack layers have been described. Several solutions propose new protocols for achieving multi-interface connectivity in the mobile environment. Even if the results they promise are compelling, modifying all the servers users need to support specific new protocols is simply unrealistic even over a relatively longer period of time. Any system to be deployed in the close future cannot require changes to internet servers or other network devices such as routers.

In addition, new wireless technologies' development and the market's fragmentation make the lower layer solutions difficult to deploy. For those reasons the most promising solutions are located at the network layer, especially given the convergence of different networks to a single IP-based approach.

There are a number of proposals that would offer feasible solutions if they were validated using practical implementations. Most of them, however, are based on simulations only. Preliminary investigation shows that the results achievable in real life significantly differ from the simulated ones. Even the proposals that were actually implemented, in most cases, are simple proof of concepts and are very far from full, deployable architectures.

Finally, there are a number of research publications that do not relate to this area at all, but which contain ideas and solutions that to some extent could be applied while designing and building practical, multi-interface architecture for mobile environments. Most of TCP-related research, such as TCP modifications and the "split TCP" approach, was inspired and designed to work in single-interface networks, solving problems with different types of wireless links. These publications do not explore the area of multi-interface connectivity, but some of the ideas are a good starting points for searching for solutions to problems in this environment.

# Chapter 3

# The design and implementation of the multi-interface system

The network layer approach offers the most potential of the available alternatives as mentioned in the "Related Work" section. The network layer solution has several advantages. It does not require applications to be modified or "tricked" to use different transport layer system calls. That is, it is relatively independent from both the operating system and the application platform and programming language used. At the same time, it can capture all IP traffic to the internet of any kind of application whether it uses TCP, UDP, or any other type of transport protocol, can benefit from it. Another advantage of this approach is that a single instance of the solution is responsible for all the traffic from all the applications running on the device. This means that it could potentially make better decisions knowing "the whole picture".

Network layer solutions are high enough in the OSI model not to be bound to any particular type of underlying physical network. Furthermore, in the process of network convergence, the IP protocol should be used for even more types of network traffic, including regular voice calls. Operating in the network layer could possibly allow this solution to handle even voice calls, allowing them to be handed over from a cellular network to a Wi-Fi network (and possibly other network types as well). This feature is already offered by UMA [69], but that is a hardware solution operating at a very low level.

Another positive aspect of network layer solutions is the little amount of changes required in the operating system. Several of the proposals discussed in the previous chapter required different levels of modifications of the operating system they were designed for. Obviously, the more changes that are required the more OS-specific the implementation

becomes and the harder it gets to deploy. Network layer solution can be deployed in the system with a relatively small number of modifications. On devices using different flavours of Linux, including Android system, Chrome OS and various embedded distributions, required modifications are limited to installing a virtual network interface driver (TUN) and running a special daemon that includes packet scheduler and all other required modules. This driver is a part of the official Linux kernel source and in some cases it might be installed by default. There is also a similar driver available for Windows. In both cases no sources or binaries of the operating system's components have to be modified.

Some of the solutions previously discussed required different "tricks" to support device mobility. The problem is that when the device changes the physical networks, the IP address of its interfaces changes. For any external server, after changing those addresses the same mobile device is seen as a completely different host. Meanwhile all the applications running on this device would also see that change and they might behave in unexpected ways. The virtual network driver creates a special, virtual interface in the system. It has its own IP address which has one very important property, it does not change when the device moves between different networks. If this device is the default one for the outgoing traffic, then applications will use this interface and will never notice network changes. Also, for remote servers, the device does not change its address when it changes physical networks. Without the virtual interface, the device has no identity at the IP network level. It is identified by the addresses of its interfaces and they change from network to network. Once the virtual interface is used, the mobile device gets its own identity which does not depend on the physical networks it is connected to or any network hand-overs.

Another major design decision is to use a network proxy. The proxy's role is to hide any mobility and multi-interface related issues from other hosts in the internet. Several of the proposals discussed assumed modifications of remote servers in the internet to support the host's mobility and bandwidth aggregation. One of our goals is to create a practical solution that could be deployed "tomorrow" without having to substantially modify the internet. Given this assumption, modifications to other hosts are impossible. In that case, for all hosts in the internet the mobile device might connect to, it needs to look like a regular host with a single interface. Having a proxy device that lies between the mobile device and other hosts allows the proxy to hide all the issues related to mobility and multiple interfaces, and to present the mobile device to the world as a regular host in the proxy's internal sub-network.

The "network layer with a proxy" approach has several key advantages. However, just like all the other types of designs it also has disadvantages. The biggest disadvantage is that all the traffic to and from the mobile device is transmitted through the proxy. One of this setup's consequences is that if the proxy experiences some problems, such as if it

crashes or the network it is connected to becomes unavailable, the mobile device can not connect to the network. This issue's severity can be minimized by locating a proxy in a well connected, reliable network. Mobile devices can connect from various places, often over unreliable, slow networks. On the other hand, we can carefully select the location of the proxy. Since it is not tied to the infrastructure mobile devices are connecting through, or even geographical location, we can use a location that guarantees high availability. The other reason for disconnection - proxy's crashes - can be dealt with by using more than one server as a proxy and by building a system that provides load balancing and fault tolerance. Finally, the mobile device side of the system can detect problems with the proxy and disable the virtual device's use and the rest of this architecture altogether. The benefits of multiple interfaces will no longer be offered and all existing connections will be interrupted, but at least the device will keep the basic level of connectivity, just as if it did not have this system at all.

Another aspect of sending all the traffic through the proxy is that every path between the mobile device and other hosts is longer than it would be if no proxy was used. Longer paths are likely to have higher latency which, in turn, causes all connections from the mobile device to experience this higher latency. This is the cost of having the proxy and all its benefits. It cannot be avoided, but it can be minimized by careful proxy placement. The proxy should have very good connectivity to the internet and, ideally, it should be close to the physical paths that connections from the mobile device would take if no proxy was in place. For example, if the mobile device uses a 3G network that belongs to one of the providers, that provider has a wide network of public access points and connections from both its 3G and Wi-Fi networks all go through one central data centre. Locating a proxy close to that data centre (or even inside it) would increase the length of physical paths (and their latency) only minimally.

## 3.1 Architecture overview

The very high level overview of a system's architecture is presented in Figure 3.1. The general scenario is providing a service to mobile devices over their multiple network interfaces, using several of them simultaneously. Mobile devices such as smartphones, but also laptops and netbooks equipped with multiple network interfaces, should be able to connect to all networks in their range and be able to use all those interfaces concurrently. This requires parts of the system to be deployed on each of those mobile devices. The rest of the network does not require any modifications. Over the 3G network, the mobile device connects to a Base Transceiver Station (BTS) and uses "data access" to the internet as offered by the

cell carrier. Another interface could connect to a Wi-Fi network. It might also be a wired Ethernet connection if such a network is available and the device provides an appropriate port. Those networks might have regular clients equipped with only a single interface as well. The device simply behaves just as if it was one of those clients and uses the regular network access which depends on the provider of that Wi-Fi or wired network. It could be a DSL (Digital Subscriber Line) modem, a cable modem, or any other method that can provide access to the internet.

The second part of the system, the Proxy, is deployed somewhere in the internet. It does not have to be inside internal infrastructure of any of the network providers the device uses. It does not have to be in any other specific location. The only requirement is that the proxy needs to have access to the internet and it has to be accessible over the internet using multiple public IP addresses.

In this design the Proxy resembles a "Home Agent" specified by Mobile IP protocol, but in this case mobile devices are never connected directly to their "home network". Another difference is that with mobile IP it is possible to use triangular routing, where IP packets travelling from the mobile device to the corresponding host are sent directly instead of going through the home agent. In this design, however, we always assume that all of the packets to and from the mobile host are sent over the proxy. This is mostly because of issues related to multiple interfaces. The traffic can be split between different interfaces and has to be reassembled and, possibly, the order of the individual packets might need to be restored. Since one of the assumptions is that no changes to other hosts are permitted, this has to be done at the proxy.

In Figure 3.2 the detailed architecture of the system on both the mobile device and the proxy side is presented. It also displays the content of the data packets sent between the mobile device and the proxy, and between the proxy and some remote server. In the figure, there are three applications running on the mobile device. One of them uses TCP traffic, another - UDP traffic, and the last one sends and receives IP packets directly. The process of sending data from those applications to the servers in the internet is following:

1. Applications A and B generate TCP and UDP data packets by sending some data to TCP/UDP sockets. Application B might use UDP protocol directly, or, for example, try to resolve a DNS name which results in sending DNS requests using UDP protocol.

2. These packets are passed to the IP layer, and are encapsulated inside IP packets. The mobile device's configuration specifies the virtual interface as the default one, so the source address of generated IP packets is set to the virtual interface's address.

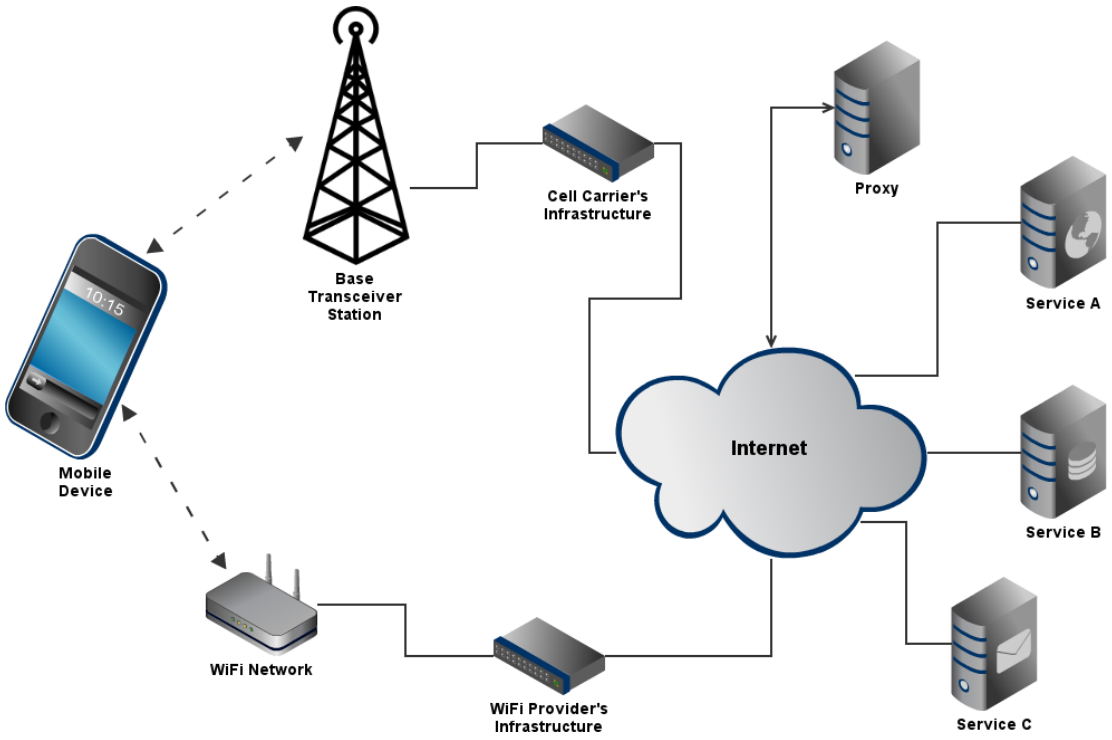



Figure 3.1: System architecture overview

3. Application C generates some IP packets directly by using, for example, raw sockets. These data packets are encapsulated in IP packets sent over the default, virtual interface - the same way as packets from applications A and B were processed.

4. The IP layer passes generated packets to the virtual interface. Since this is a virtual device, instead of leaving the device over some physical network all the packets are received by a special application running on the device, called "Scheduler".

5. The scheduler receives not only the payload of those packets, but complete IP packets, including their headers. It can inspect those packets, including all the fields present in IP headers such as the destination address and transport protocol type. It can also access the content of used protocol's header such as TCP or UDP port numbers and other properties. The scheduler could also inspect the actual payload of those packets as well, if required. In most cases, this level of inspection should be avoided due to computational overhead, but it is possible if required.

   Based on the content of processed packets and the knowledge about the properties of available networks, the scheduler decides which physical interface should be used for each individual IP packet.

6. The scheduler sends received IP packets to the proxy using TCP or UDP protocol. The exact way the communication between the mobile device and the proxy occurs, as well as reasons behind this choice, are discussed in the "Protocol" section. These new TCP or UDP packets are sent over the chosen interface. In Linux this is achieved by using per-interface routing rules. The proxy offers several different public IP addresses, which can even be assigned to the same physical interface (of the proxy). The number of those public addresses is the maximum number of different interfaces that can be supported on the mobile device. When the scheduler running on the mobile device sets up its physical interfaces, it configures a routing rule to the proxy's different IP addresses. It will create a routing rule for packets to the first proxy's address to use the first physical interface, those to the second proxy's address to use the second physical interface, *etc.* Based on this configuration, the scheduler can easily select the interface it wants to send packets over by using the appropriate address of the proxy. This configuration allows the scheduler to control which interfaces are used without modifying any parts of the operating system running on the device.

7. The application running on the proxy receives the packets which reach the proxy on one of its public IP addresses. This application contains almost the same scheduler as the one that is running on the mobile device. The biggest difference is that

the proxy's scheduler has to support several different mobile devices and the mobile device's scheduler communicates with only a single proxy.

Once the scheduler receives the packets and they are examined, they could be either immediately sent further or buffered. Buffering packets is needed to restore their order, as the proxy may have received them in a different order than they were originally sent. Different networks have different parameters, and depending on the scheduling method, the receiving scheduler may need to restore the correct order. Once the order is restored (or if no reordering was needed), the payload of those packets (so the original IP packets generated by applications running on the mobile host) are injected to the proxy's IP stack using its virtual interface.

8. Packets reaching the proxy's IP layer over the virtual interface have the destination IP address set to the remote server they are supposed to reach's address. Their source IP address is set to the mobile device's virtual interface's address. They can simply be public IP addresses, but since the number of public IPv4 addresses is very limited, in most cases they will be private addresses from the proxy's internal "private network". In case private addresses are used, standard Network Address Translation (NAT) is needed. It can be applied on the proxy itself, or could be a separate router located between the proxy device and the rest of the internet. The whole system works just as if all mobile devices were connected with their virtual interfaces to a local, private network with the proxy (connected to that local network with its virtual interface) device working as a gateway with NAT mechanism. Source address of packets leaving the private network is rewritten to the proxy's public address, and packets arriving at the proxy, which are destined to that private network, have their destination address rewritten to the address of the corresponding device inside the private network.

9. Eventually, all packets sent to the internet will have public source IP. When they reach their destination, they look like a regular packets generated by a device with only one interface.

When the remote server responds, it generates packets for that public IP. Whether this is the IP of the NAT box running on or close to the proxy, or the public IP of the virtual interface on the client (in case each mobile client has its own public address) they need to be routed back to the proxy. In case the public address space is used, the proxy has to be the router which is responsible for delivering packets to that sub-network. If the NAT approach is used, after converting addresses to the private space, they too need to be routed to the proxy. The proxy, on the other hand, has to be configured to forward packets for

that address range to the virtual interface. When they reach that interface the scheduler running on the proxy receives them. From here, the proxy scheduler behaves just like the one running on the mobile client. Packets are assigned to one of the mobile device's physical interfaces, encapsulated, and sent over the internet to the mobile device. There they are received by the mobile device's scheduler, reordered (if needed), and the encapsulated IP packets are re-injected to the IP layer, from which they are delivered to the appropriate application. The IP packets the application receives are the same packets that the remote host generated (except for NAT address rewriting that might have taken place). So, from an application's perspective, there is no difference between a multi-interface scenario, and a situation in which the device is connected with its virtual interface to the private network with the proxy acting as its default gateway and NAT router.

In this design, both remote servers and applications running on the mobile host (and even transport protocols on the mobile device) see regular IP traffic between the mobile device and the remote server. However, that traffic is sent and received over the virtual interface that is always there, and it does not disappear or change its address when the underlying physical networks change.

In the following section individual components of the system are discussed in greater detail.

## 3.2 Tunnelling method

In Section 2.4 we discussed several different network layer methods of data exchange between the mobile device and either the proxy or the remote server (depending on the design). The most commonly used method is IP-in-IP tunnelling, where IP packets are encapsulated inside new IP packets with different source and destination addresses. This approach adds 20 bytes of data overhead (for IPv4) because each original IP packet now has one extra IP header. Another solution is packet rewriting. In this case, the original destination address is changed to the address of the mobile device's selected interface. This solution does not introduce any data overhead, but it cannot be used for traffic in both directions. The proposed design using a NAT-like solution was operating on data sent to the mobile device. In that scenario the destination address of all IP packets is known because it is the device's main interface's address. Traffic in the other direction was not considered. Our design assumes a bidirectional tunnel between the proxy and the mobile device. If we modified the destination address of IP packets leaving the mobile client to go to the proxy there would be no way of restoring the proper destination address at the proxy. Each packet received could be directed to any of the hosts in the internet. Another

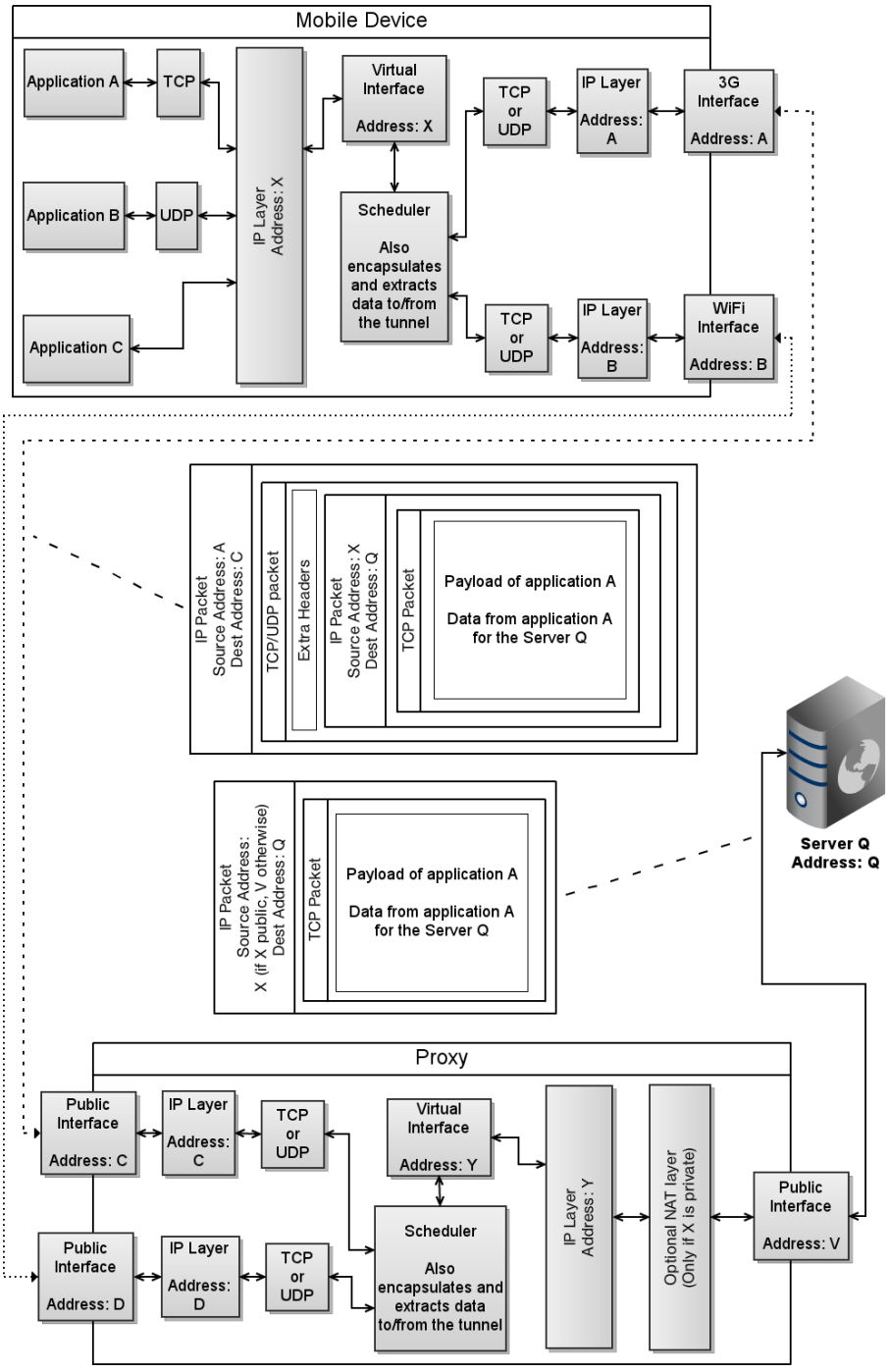



Figure 3.2: Modules of the mobile device and the proxy

problem is the fact that we cannot include any additional information in packets sent using NAT technique unless we modify the original packet. Additional parameters are needed, for example, for any sequence information required to restore the proper packet order and also for other control data. Any modifications to the original packets would have to be different depending on the payload protocol type, making this solution impractical.

The IP-in-IP approach also makes including any extra information difficult. In this case, however, it is possible to utilize unused IP headers to carry additional control data. It is not very straightforward, however, and the amount of the extra data is very limited.

For the reasons mentioned, but also for easy implementation's sake, we decided to tunnel data using UDP or TCP protocols. With this approach any amount and format of control data can be included with data packets. This obviously increases the protocol's data overhead - each packet is increased by the size of one IP header, one UDP or TCP header, and any additional data fields. The address rewriting technique cannot really be used, so the difference in overhead between our approach and the IP-in-IP one is (at least) 8 bytes when UDP protocol is used, and 20 bytes when using TCP. However, in this case the internal packet structure can be freely modified. There is no need to include the complete IP header in the packet's payload, or even the transport protocol's header when it is known. Techniques for compressing TCP/IP headers exist [37] and can be easily applied here, limiting the overhead added.

Thus far, whenever tunnelling is described both UDP and TCP protocols are mentioned. We want our system to be able to use both protocols, although the UDP transport protocol is preferred. UDP introduces less data overhead and lacks its own congestion and reliability algorithms, but practical experience shows that in some cases UDP traffic will be blocked. Some public Wi-Fi access points seem to limit the types of traffic they allow, allowing users to perform only a selected set of tasks over the network. Even the University of Waterloo's public Wi-Fi network restricts the devices to only HTTP and HTTPS traffic over TCP protocol unless they run a special utility or are recognized as running one of the operating systems which have full access. This recognition is performed using special Java Scripts on the login web page which, in case some automatic authentication system or set of scripts is used, might be difficult to run. This is done to increase the network's safety. This kind of filtering is common and shows that some wireless networks might be quite restrictive. One of our protocol's requirements is that it should be able to run in those "semi-hostile" environments. Of course it would be better to use UDP instead of TCP, but in case UDP access is unavailable, having TCP is much better than not being able to use a particular network at all.

Using TCP instead of UDP introduces several problems. Increased data overhead is

only one of them. Another issue is related to the fact that TCP is reliable and performs congestion control. In application space, this is a good thing - developers do not have to implement those elements themselves, they can just rely on guarantees TCP provides. However, in this case we are using TCP to send packets sent by other protocols. With no tunnelling those packets would travel over the IP network that does not have any guarantees; it can lose or reorder packets and even corrupt them. If an application uses UDP protocol, it assumes that UDP packets can be lost or reordered. The application has ways of dealing with that. Usually UDP protocol is used precisely because the application does not need TCP's delivery guarantees. For example, VOIP applications prefer to lose packets than to introduce additional delays related to the way TCP operates. That is why they use UDP protocol. By tunnelling UDP packets over TCP protocol, we "add" delivery and ordering guarantees to them (at least between the mobile device and the proxy). However, instead of improving the network conditions for applications using those UDP packets, we might be making things much worse by, for example, adding additional delays and increased data usage due to retransmissions. Considering that the application using UDP protocol does not need those guarantees, this is something we should avoid.

This situation is even worse when TCP over TCP is used. TCP protocol assumes that the underlying medium can lose or reorder packets. It runs its own reliability mechanism and retransmits the data when needed. When TCP is sent over TCP, the same algorithms are run twice. In case of problems, the underlying TCP layer can be retransmitting data that the higher TCP protocol decided to retransmit as well. The data usage and delays increase, decreasing the throughput. That, combined with two congestion control algorithms running at the same time, using the same time scale, leads to the effect known as "meltdown" [65].

Given these issues, TCP protocol should be avoided when possible. We want to provide the option to use TCP protocol only as a fallback and focus our efforts on the UDP tunnelling. The TCP case should be also explored, but it has significantly lower priority. In the following sections only the UDP case will be considered unless explicitly stated otherwise.

## 3.3 Communication between the mobile device and the proxy

Each mobile device and the proxy server communicate with each other using a custom protocol. It is used mostly for tunnelling IP packets exchanged between the client and the

rest of the internet. This protocol needs to not only carry those IP packets, but also some additional data the remote side of the tunnel requires to properly reassemble data streams that were possibly split between different interfaces. This is particularly critical in TCP's case, which might decrease the throughput if packet reordering is introduced. However, in case other transport protocols such as UDP are used, the reordering should also be avoided if possible.

Another role of the protocol is to provide a control channel between the mobile device and the proxy. This control channel can be used for exchanging information needed for better scheduling. Both the proxy and the mobile device have their own schedulers, and they each control the traffic sent from one end of the tunnel to the other. One side of the tunnel does not need to make the same decisions as the other (for example, TCP ACK packets do not necessarily have to be sent over the same interface the TCP data packets of the same stream use), but both sides need some cooperation. For example, one side of the tunnel might have more information than the other. If the mobile device notices that one of the network interfaces has just been disabled or that the signal level has dropped below a certain threshold, it could instruct the scheduler running on the proxy to stop using that particular interface. This data is only available on the client, which is usually much closer to the part of the path that is likely to cause problems. The proxy, on the other hand, does not have direct access to the information about the interfaces' or the networks' physical parameters. When the proxy receives control updates from the client it can react much faster to changing conditions. Instead of waiting for the connection over the disabled or flaky interface to die, it can stop using that interface as soon as it receives information about problems with said interface from the client - sent over another interface that is still working properly.

Another usage type is exchanging scheduling policies. There are different approaches to traffic scheduling, and the user could possibly control some of them. For example, the user might want to configure which protocols they care about or how much data can be sent over the 3G link without reaching data plan limits and incurring increased monetary costs. This kind of data is only available at the mobile device level, where some kind of user application would be running that can be used to configure the system's behaviour.

On the other hand, some of the information could only be available at the proxy which, possibly, could be a part of bigger infrastructure a network provider owns. There could be some global decisions based on current network usage in different areas and different policies applied to different users (for example to offload all the traffic from the temporarily overloaded 3G network). These kinds of settings would be transmitted to the mobile client's scheduler to modify its behaviour over the control channel.

The exact format of the protocol is an implementation detail. One of the global decisions, however, is how the data of this protocol is sent over the network. As previously discussed, the IP packets should be sent over UDP protocol if possible. Concurrently, any control information sent over the control channel should be reliably delivered. If we depended on the UDP protocol, we would have to provide some methods of ensuring the channel's reliability, practically implementing a TCP subset. This is why the "data (or tunnelling) channel" is a separate UDP channel, and the "control channel" uses separate TCP connection. In practise, each mobile device will maintain two separate channels with the proxy over each interface - one for tunnelling the data and the second for the control data.

The first of the channels - the data channel - uses UDP protocol whenever it is possible. In some situations UDP communication may not be possible. Usually it is caused by firewalls that allow for only some of the types of traffic to go through. Whenever UDP channel cannot be established, the system will fallback to TCP protocol. The data channel is used for:

- IP tunnelling
- Latency measurements
- Exchanging non critical control messages that can be lost (for example any usage statistics)

Basically, any data that does not need to be reliably delivered, including low priority control messages, should be exchanged over the unreliable UDP channel. This channel, if UDP protocol is used, also provides a better way of measuring channel latency which would be much less reliable if a protocol with retransmissions was used instead.

The second channel - the control channel - always uses TCP protocol. It is used for:

- User authentication and authorization
- System configuration
- Scheduler parameters
- Link/network parameters and updates
  - Interface becomes unusable

  - One of the networks is no longer available at all

- Any other control messages that require reliability

Any type of data that is critical for the system's correct operation needs to be exchanged using a reliable control channel. It is important to note that even when the data (or tunnelling) channel uses TCP protocol there are still two separate channels. In that case, they would use two separate TCP connections. This is because the data channel has high traffic. On the other hand, there should be little traffic over the control channel. If the control messages were sent over the same TCP channel as the tunnelled packets, they would be buffered behind all the previously sent, but not yet delivered, data. This would increase the control channel's delay. By using two separate TCP connections we minimize this delay, allowing both devices to exchange control data with lower delay. This does not deal with any buffering of IP packets in the network between the mobile device and the proxy, but only with local TCP buffering.

## 3.4 Packet scheduling

The system's core is the "scheduler". The scheduler needs to be running on the mobile device and on the proxy. Those two instances of the system can be almost symmetrical. There are small differences between the scheduler running on the client device and the one running on the proxy. The differences include the fact that the proxy's scheduler needs to support multiple clients and the mobile device's does not. Other differences include different data on each side of the tunnel. The scheduler on the mobile device will have more information about local network conditions, the device's interfaces, and their parameters. Conversely, the scheduler running on the proxy might have some additional information about the global network state (in case it is integrated with specific provider's networking infrastructure). The proxy's scheduler also needs to provide special control operations used for user authentication and authorization (potentially using an external database).

The scheduler's core, however, can be identical. On both sides of the tunnel the scheduler's role is the same - it is supposed to examine incoming packets and based on their content, current network conditions, and the system's configuration schedule them for delivery to the other side of the tunnel. At this location the architecture's important algorithms will be running, making all of its features, such as client mobility or bandwidth aggregation, possible. In the following sections several aspects of the scheduler and the features it needs to provide are discussed.

The high level design of the system only assumes the existence of "some" scheduler. There can be a number of completely different scheduler implementations, built to achieve different goals. In this research we focus on two primary goals the system should achieve to offer its users improved network experience - mobility and bandwidth aggregation.

### 3.4.1 Mobility and fault tolerance

In the single interface scenario, mobility is a relatively simple problem. When one network becomes unavailable and the device connects to another, the system has to either reroute packets or establish a new network path that existing connections could use. No data access is possible when the device changes networks. Also, when the network becomes unusable (due to interference or distance to the access point or another device used for wireless data access) there is no alternative. The device tries to use the only available network, hoping to eventually get data through.

In multi-interface scenarios, each of the available interfaces might change networks individually. When one interface becomes unavailable there could be other interfaces that still can be used for sending and receiving data. The networks might become rapidly unavailable due to hardware problems, such as crashes or disconnections. In this case the scheduler needs to provide a fault tolerance for existing traffic streams. Practically, the fault tolerance is just a special case of the mobility problem. Actually the issues related to mobility are more difficult to deal with than those related to sudden crashes and disconnections. When the network goes away it is easy to simply stop using that particular interface. However, usually networks do not become unavailable rapidly. In the typical scenario, network capacity gradually degrades to the point when no traffic goes through. However, if there are other interfaces available, it often makes sense to stop using that degrading network much earlier. It is not clear how that decision should be made. Different networks could have completely different characteristics and determining which of them should be used, and for which fraction of the traffic, is a difficult task. This research focuses on the traffic scheduling and not network monitoring techniques, so the knowledge about the networks used by schedulers is somewhat simplified. However, the schedulers could easily take advantage of improved network quality and capacity information, once the appropriate monitoring techniques were added to the system.

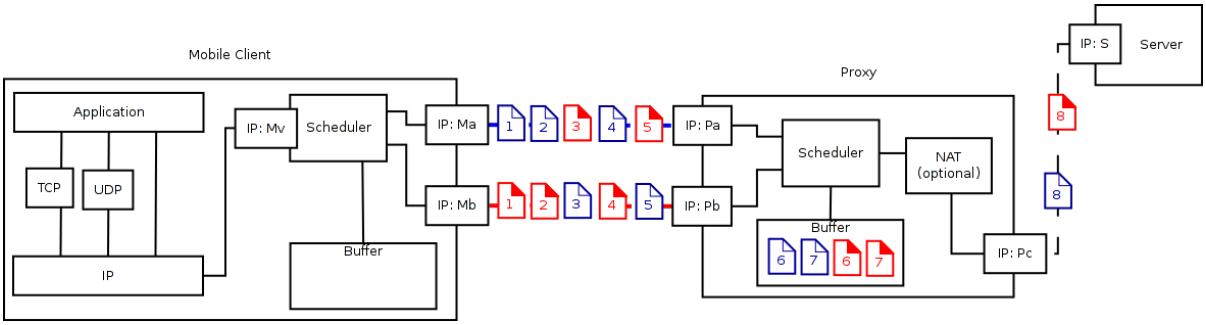
### 3.4.2 Bandwidth aggregation

Bandwidth aggregation is the biggest problem the scheduler needs to face in a multi-interface, mobile environment. The basic idea behind bandwidth aggregation seems quite simple - to maximize total throughput, achieving more than the maximal bandwidth offered by any single interface of the several available ones. Several proposals discussed in Section 2 focused on a single data stream at the transport layer or above it. There are also some lower layer solutions, but most of them either focus only on a specific traffic or transport protocol type or ignore that aspect completely, providing a generic solution. Also, in most cases evaluating the systems is very limited and often based only on simplified simulations. Finally, some of the results ignore the complexity of TCP related issues that could potentially make the entire system unusable in real life scenarios.

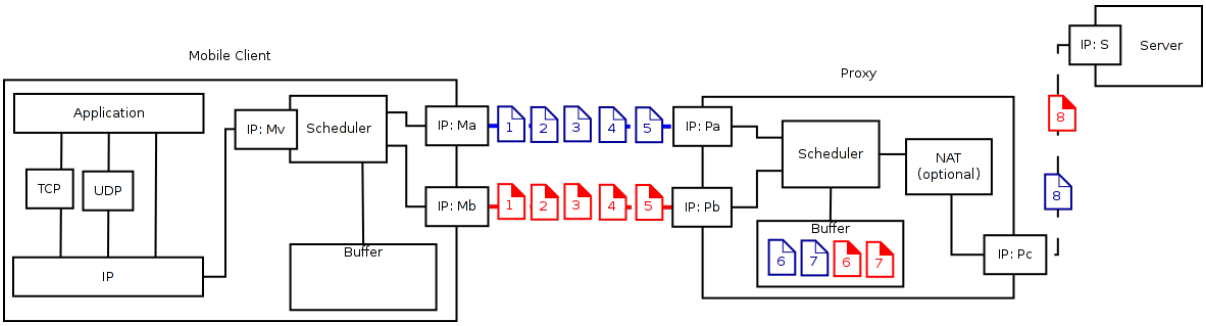
On the other side of the solution range is a design described by Tsao and Shivakumar as offering "super aggregation" [66]. It does not try to perform bandwidth aggregation because its authors argue that usually only one network offers high bandwidth and that extending it with much smaller bandwidth other links offer does not make much sense. Instead some "tricks" are used that supposedly improve overall performance on per transport protocol or per application type basis.

The most common approach among existing proposals is per-packet scheduling. Most designs describe schedulers that make decisions based on individual packets. Kumar and Golash present a completely different approach [41]. They suggest that much better results can be achieved by using flow based scheduling instead of packet based. This publication, however, is purely theoretical. Our preliminary results with simple schedulers show that their conclusions are also likely true for real-life scenarios. When real TCP traffic is used over several links with different and always changing parameters it is very difficult to achieve significant bandwidth increase. On the other hand, when flow based scheduling is used no improvements can be achieved when there is only one active flow.

The per-packet scheduling is presented in Figure 3.3(a). Packets that are part of different flows are present on both physical links, which can cause reordering and seriously affect the performance. In comparison, the flow-based scheduling presented in Figure 3.3(b) keeps all the packets belonging to the same flow on a single link. This approach minimizes packet reordering (to the levels naturally occurring in IP-based networks) and offers much better performance.




(a) Packet-based scheduling




(b) Flow-based scheduling

Figure 3.3: Packet-based vs flow-based scheduling

## 3.5 Priority scheduler

The priority scheduler is meant to offer improvements in the first category: mobility and fault tolerance. It does not attempt to offer increased bandwidth, but network continuity in presence of networks appearing and disappearing. Its design is simple, yet very effective, when compared to the ordinary network behaviour while switching between the networks. This scheduler uses the configuration specifying the desired priorities of different networks. These priorities could be as simple as "use Wi-Fi, unless it is not available", but could also include more parameters of the networks - their SSIDs, carrier names (in case of 3G networks), or any other options that may be relevant. It could even use the quality parameters of the networks and not use the networks that do not meet quality requirements. The type and the number of configuration parameters used does not change the basic design of the scheduler, so in this document only a simple configuration will be considered.

It is important to mention, that even the most basic configuration ("prefer Wi-Fi over 3G") which seems like the mode of operation of a regular device offers significantly improved behaviour. This scheduler offers network continuity - it is capable of migrating existing connections between different physical networks. This is not something that can be achieved with a simple "connection manager" as found on most of the mobile devices.

This scheduler's implementation is relatively simple. It manages the list of available links, and schedules every outgoing packet over the link with the highest priority - subject to other constraints, like minimal link latency, quality, etc. Since the packets are, generally, sent only over a single link, there is no need for any additional processing or buffering of incoming packets, which are simply forwarded right away.

## 3.6 Round-Robin scheduler

The round robin scheduler is built with the other goal in mind - the bandwidth aggregation over multiple links. This scheduler, however, is not meant to achieve improved performance in real life conditions. It represents the naive approach to the problem of bandwidth aggregation, used mostly as a reference to other schedulers. In the discussion about packet vs flow-based scheduling in the previous chapter, shortcomings of packet-based methods were mentioned. This scheduler is based on simple packed-based bandwidth aggregation, and as such is not meant to be treated as a real, usable solution. It is just a point of a reference to other schedulers, that should perform better.

The implementation is also quite simple. Outgoing packets are scheduled over available links in round-robin fashion. Incoming traffic can optionally be buffer to restore the original

order of packets. Since this scheduler can use multiple links in parallel (which means multiple network paths, with possibly significantly different characteristics), packets will get reordered. Restoring the order of the packets is, however, more complex task that it may seem. The simple approach would be to buffer out-of-order packets until the order can be restored. The main problem with this approach is caused by packets losses. Any packet lost will make the algorithm buffer and try to reorder all subsequent packets for some time, until the packet is considered lost. This will increase the overall link latency, as perceived by TCP running over aggregated links, trigger retransmissions, increase the back-off time, and, in general, negatively affect the performance. Modifying the algorithm to work well with TCP protocol is at least a complex task, and an area of research on its own. Since this scheduler is meant to be used as a reference only, and not a real-life solution, we assumed a simplified approach which involves simply forwarding packets, whether they are in order or not. The TCP protocol running over the aggregated link will deal with packet reordering.

Even though this scheduler is presented as "bandwidth aggregating" scheduler, it will also have "mobility" and "fault tolerance" characteristics. This is due to the overall system design - links that fail or disappear are simply removed from the system, and any scheduler has to switch to other available networks, if there are any. This is, effectively, hand-over in the presence of failures or networks disappearing due to mobility.

## 3.7 Hashing scheduler

The scheduler presented in the previous section operates on per-packet basis. It is not meant to be a real-life solution, but just a reference point. There are other packet-based algorithms discussed in the previous chapter, none of which seems to be a viable candidate for a real-life implementation. Also, none of them aggregates traffic based on flow information. Even though in work by Chebrolou *et al.* [20, 19] the concept of different flows is mentioned, it is only used in context of fairness while merging packets from different flows - the data is still aggregated on per-packet basis.

Traffic aggregation done at the network level gives the opportunity to recognize the fact that different packets belong to different flows and potentially improve the way the bandwidth is aggregated.

To the best of our knowledge there is no practical, network-layer solution proposed, that would aggregate bandwidth on per-flow basis. There exists some analysis of theoretical link load balancing with flow identification in Kumar and Golash's work [41], as well as some observations of the effects of packet reordering in Bennett *et al.* [12] and Blanton *et al.* [14]

publications. None of them, however, introduces a solution that could be implemented and deployed in practice.

The hash-based scheduler is designed to be a practical implementation of flow-based aggregation.

### 3.7.1 Flow identification

The hashing scheduler uses port-based flow identification. The IP packets themselves do not contain port number information, as it is specific to transport protocols. Both TCP and UDP protocols, however, include port number information, and those two protocols represent the absolute majority of all internet traffic. All other packets, that do not contain port numbers, are a very small part of traffic and can simply be treated all as one, special "flow", or scheduled in a completely different way altogether without affecting the overall results in any significant way.

Each TCP and UDP packet contains two numbers - the source and the destination port. Typically, one of those numbers is a well known port number - for example 80 for HTTP. This port number is not very useful, since every application could use several separate flows with the same well known port number. The other port number, however, called "ephemeral port number" is a short-lived, automatically allocated number which, for every connection originating from the same system (and active at a time), is different. For packets sent from the client to the server the ephemeral port number is the source port number. For packets coming back it is the destination port number. Since we do not know which side of the connection is the client and which is the server, we take advantage of port number ranges. Even though those port number ranges can vary between different operating systems, they follow the same logic - the well known port numbers have small values, and ephemeral port numbers have large values. Because of that, for each packet handled by the system, we take the higher of the two port numbers and use it as flow identification.

In the mobile environment the mobile device is, almost in all cases, exclusively a client for multiple different remote servers. This means that all ephemeral port numbers are generated by the same system. Also, the observed behaviour is that those numbers are not assigned randomly, but sequentially.

Based on those assumptions we designed an algorithm for assigning individual packets to specific links with respect to the flows those packets are a part of.

### 3.7.2 The algorithm

The algorithm takes the higher port number from each packet (which should be the ephemeral port), calculates hash value for it, and based on this hash value it assigns the link the packet should be sent over. Each packet within the same flow results in the same hash value, and will be scheduled over the same link, unless link configuration changes.

The key of the algorithm is the hashing function. Because the port numbers we use are assigned sequentially, we designed a function that assigns linearly increasing values evenly in the result space. To achieve that we take the 16 bit port value and reverse the order of bits in it. The result is a 16 bit number that "points" to a specific spot in 0-65535 range. A different size of the result could also be used if needed, and less bits of the port number could be used. This function has very interesting characteristics for sequential values. The port numbers, reverse-bit values and resulting hash values are presented in Table 3.1 and Table 3.2. These tables contain only 3 bit values, but the characteristics apply to 16 bit values as well.

The hash function assigns different flows with sequential port numbers evenly in the 16 bit space. The second part of the algorithm is to assign values from this space to specific links. To achieve that, each link is assigned a weight, that reflects how "good" that link is, and how often it should be used. If approximate link characteristics are known in advance those weights could be static. For dynamically changing (and unknown in advance) link characteristics any other method could be used. In our experiments we used fixed link weights as well as packet loss measurements, but any other, more sophisticated method could be used. Measurements of available bandwidth, physical qualities, or possibly a cost associated with a link could be used to generate those weights. The method for achieving that is out of scope of this paper and is left for future work. Once there is a list of links with weights assigned, we can divide the whole 16 bit space used by the hashing function among them - the higher the weight of the link, the bigger range in that space a link is assigned - and the bigger number of flows is scheduled to be sent over that link.

As the conditions change - so measured link properties and their weights change as well, those ranges will be adjusted. The most significant change is adding or removing a link. In that case flow assignment could change, and the effect of packet reordering could occur. But those changes are significantly less frequent than, for instance, sending every other packet within a single flow over a different link.

The biggest advantage of this algorithm is that it does not actually have to keep track of individual flows. It does not need to store any specific data about individual flows in the memory, and the amount of space it needs (even though it is extremely small) depends

| Port number | Port in bits | Reversed bits | Reversed value |
|---|---|---|---|
| 0 | 0000 | 0000 | 0 |
| 1 | 0001 | 1000 | 8 |
| 2 | 0010 | 0100 | 4 |
| 3 | 0011 | 1100 | 12 |
| 4 | 0100 | 0010 | 2 |
| 5 | 0101 | 1010 | 10 |
| 6 | 0110 | 0110 | 6 |
| 7 | 0111 | 1110 | 14 |
| 8 | 1000 | 0001 | 1 |
| 9 | 1001 | 1001 | 9 |
| 10 | 1010 | 0101 | 5 |
| 11 | 1011 | 1101 | 13 |
| 12 | 1100 | 0011 | 3 |
| 13 | 1101 | 1011 | 11 |
| 14 | 1110 | 0111 | 7 |
| 15 | 1111 | 1111 | 15 |

Table 3.1: Port numbers and "reversed" values

on the number of links, and not on the number of flows or packets. At the same time it offers good bandwidth utilization as long as there are multiple flows available - which, in today's use of the internet, is a typical situation.

The disadvantage of this algorithm is that it does not try to achieve fairness between different flows. Depending on the port number assigned and characteristics of individual flows, it is possible that the ratios of amounts of data assigned to each link are different from ratios of their weights. However, even in this scenario, as long as there are some flows assigned to every link, the algorithm allows for higher total throughput than would be possible using a single link - and without incurring penalties related to extensive packet reordering.

### 3.7.3 Hash weight adjustments

The hashing algorithm is effective when hash values calculated for different flows are distributed across the entire hashing space. Our results show that in most cases this is true, especially as the number of parallel flows increase. However, there are no guarantees that

| Hash | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Port Number | | | | | | | | | | | | | | | | |
| 0 | X | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | X | | | | | | | |
| 2 | | | | | X | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | X | | | |
| 4 | | | X | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | X | | | | | |
| 6 | | | | | | | X | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | X | |
| 8 | | X | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | X | | | | | | |
| 10 | | | | | | X | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | X | | |
| 12 | | | | X | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | X | | | | |
| 14 | | | | | | | | X | | | | | | | | |
| 15 | | | | | | | | | | | | | | | | X |

Table 3.2: Hashing function

there are no special conditions, where all active flows use port numbers that assign them to the same link. This is a rare case, and difficult to observe in real life conditions. However, we performed a set of experiments with carefully chosen source port number for test flows. We were able to observe all the flows being scheduled over the same link and sharing only one's link bandwidth, leaving the other link unused.

To solve this problem, we propose a separate algorithm that can be used in combination with our hashing approach. One of the strengths of the hashing algorithm is that it does not need to keep track of individual flows. Doing that would be highly impractical and costly. At the same time it means that per-flow link assignment is not possible. Figure 3.4 presents the diagram of this algorithm.

Instead of trying to identify and monitor individual flows, we keep track of amount of data sent by the flows in different ranges of the hash space. Each link is assigned a region of this space, the size of the region depends on this link's weight. We divide each link's region into a number of sub-regions. The experiments we performed were done using 4 sub-regions per each link, but any number larger than one could be used to achieve different resolutions of the algorithm. Every flow, that is assigned to a specific link contributes to the amount of data sent over specific sub-region of this link, depending of this flow's hash

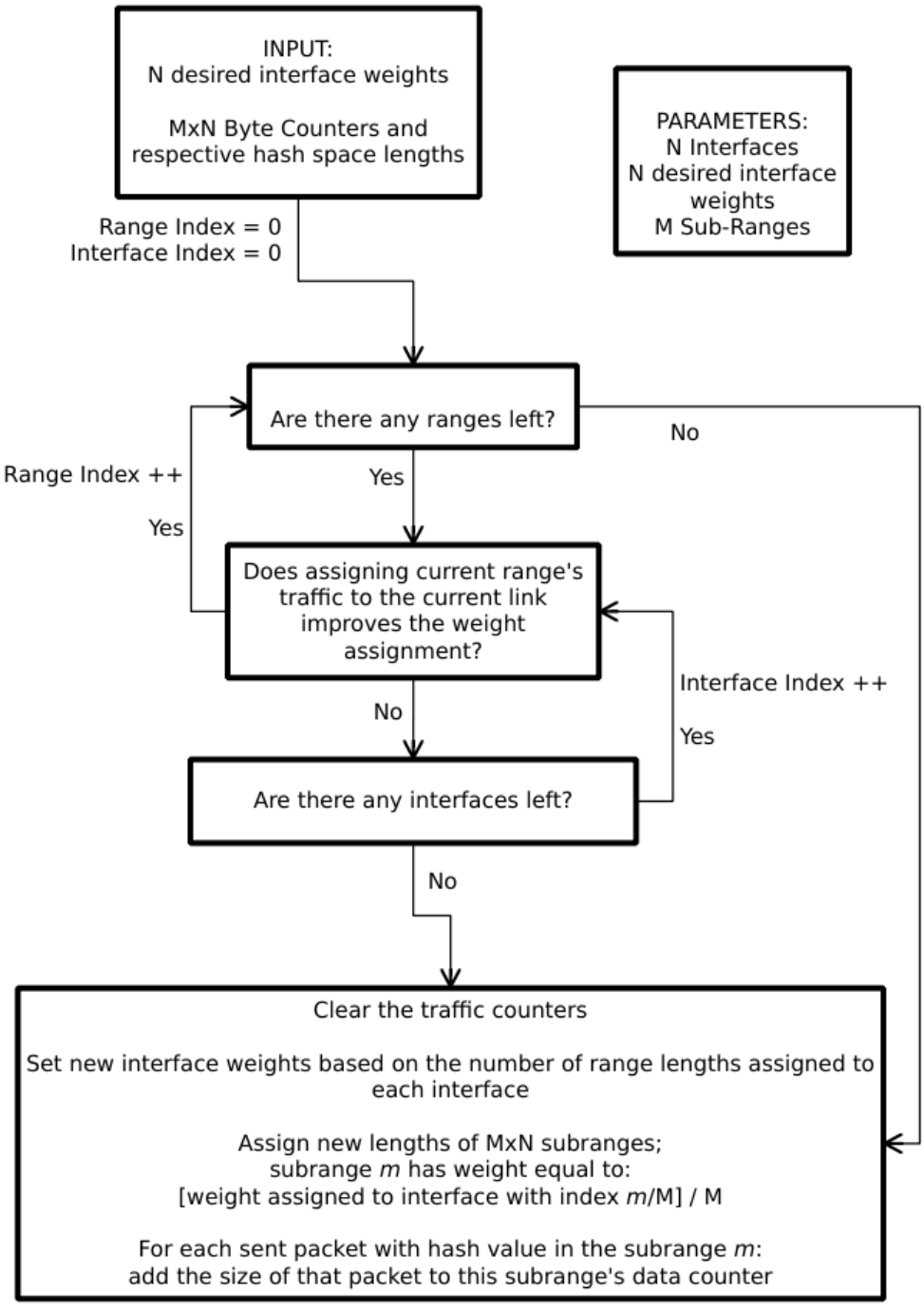



Figure 3.4: Hash weight adjustment algorithm

value. The algorithm is run at specific intervals and it examines sub-ranges of every link and adjust the range of the hash space that is assigned to each link. It examines the total amount of data sent by the system, and iterates over all link's sub-ranges, assigning them to individual links depending on the desired data share of each link - which is, essentially, this link's weight. The algorithm does not change the order the links cover the hashing space, just the size of this space that is assigned to specific links.

For example, if two links have the same weights, but one of the link gets all of the data (for simplicity we assume that flows that contribute to that data are evenly distributed over this link's hash share), the adjusted weight of that active link will be reduced to the half of the original size (so one fourth of the hashing space), and the adjusted weight of the inactive link will be increased to three thirds of the entire hash space. This way half of the data that was sent over the first link will be reassigned to the second link.

This might seem like pushing the problem further away - instead of skewed flow allocation over available links we now operate at four times higher "resolution" (if 4 sub-regions are used per each link). If the data flows are very skewed (like in our carefully crafted experiments), the problem would still persist. However, this is not the case. As we adjust weight of each link, the sub-regions that we use for monitoring data flow are resized too - each link, even after weight adjustment, still is divided into the same number of sub-regions (in this case 4). The difference is that in absolute values those ranges are smaller now, and cover smaller part of the hash space. During the next iteration of the algorithm, those regions will represent the data distribution at a higher resolution, allowing the algorithm to tune those weights further and better. Effectively, those links that originally were not getting enough (with respect to their weights) data will decrease monitoring resolution, while the active, "hot" links will increase the monitoring resolution over time, allowing the algorithm to adjust weights to achieve better flow assignment.

Depending on the number of sub-regions for each link and the interval at which the algorithm is run, the process of adjusting the system to the data flow can take different amounts of time. Also, when there is only one, "heavy" flow in the system, we will not be able to achieve bandwidth aggregation at all. However, in the process of adjusting weights, very narrow range of the hash space, the one that this flow is assigned to, will be assigned to a single link, which means that any other flow will be almost guaranteed to be assigned to one of the remaining links.

## 3.8 Power consumption

The previous sections discuss several different algorithms and solutions for improving the connectivity in mobile environment. The advantages and disadvantages with respect to traffic are presented. Most of the solutions may also have additional cost - increased energy usage. This cost is very important especially in the mobile environment. Whenever additional data processing is involved, the energy usage grows. In the case of solutions that use multiple interfaces in parallel, these costs will be even higher, since multiple network adapters will be active at the same time. To the best of our knowledge there are no studies on effects of multi-interface connectivity on the energy usage of mobile devices.

The monitoring of energy consumption is achieved by performing multi-interface connectivity tests and experiments on devices that draw power from the external power supply, instead of using their own batteries. This power supply is connected to the computer system that monitors and logs the energy usage during all the experiments performed. The scenarios and findings are presented in the next chapters.

# Chapter 4

# Evaluation

Evaluation of the system and algorithms presented in the previous chapter is be performed using several different usage scenarios in a testbed environment. The client side of the system described is implemented and deployed on the Android smartphone devices - Galaxy Nexus. The proxy side of the system is running on a regular Linux server. Public HTTP servers were used for data transfers.

The high level overview of the testbed is presented in Figure 4.1. The energy usage monitoring part of the testbed is the same as used in earlier studies of mobile devices using single interface network connectivity [7, 49, 1].

The mobile device has access to both a standard Wi-Fi network, as well as mobile carrier's 3G data network. The data transfer tests are performed using a custom software that performs single or multi-flow TCP data transfers from a TCP server. Since the Wi-Fi network's access point, system's proxy, and the test server are all located at the same location, the network access over the Wi-Fi network is throttled to offer at most about 5 Mbps throughput, which is a value often offered by DSL links. This is to run the experiments in conditions that are more likely to be experienced outside of the testbed environment, in the real world. After all, most of the services likely to be accessed from a mobile device are located outside of the local network that device is connected to.

During the experiments, the mobile device is using Keithley 2304A Power Supply, and the energy usage is monitored and logged using a computer workstation. Two sets of data are gathered for each scenario - one related to the network operations (data rate), the other - to the amount of energy consumed.

The metrics for the evaluation also fall in two categories: traffic parameters, and energy

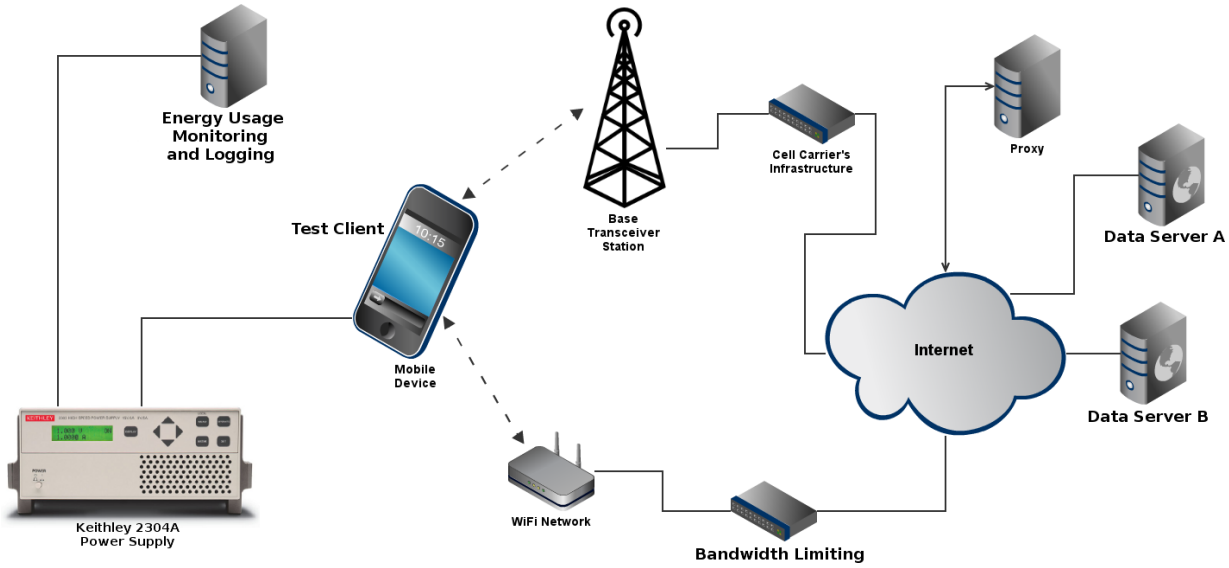



Figure 4.1: Testbed overview

parameters. The first one can be seen as the benefits the system has to offer, the other - what energy costs are associated with specific solutions.

The experiments are performed in different usage scenarios, both static, when network conditions remain similar, and dynamic - when the device switches between the networks or when it gets connected or disconnected from them.

During the experiments there will be three nodes involved:

- The client device, running the multi-interface system as well as the test application. The multi-interface system will be forwarding and scheduling packets using one of the algorithms. The test application will be downloading data using one or several TCP flows at maximum possible speeds.
- The proxy device, running the multi-interface system and only forwarding packets using the specific scheduler.
- The remote web server. It will not be running the multi-interface system, it is simply a regular, public, internet server. It is serving data files of specific size over one or several TCP connections (depending on the client). Basically the test application will open some number of TCP connections to this server, and the server will start pushing data over them.

The applications considered will be test applications, but will use standard TCP connections to download data. Which is consistent with any application downloading data over TCP - like file download, but also a Youtube player application, etc.

## 4.1 Client device

The client device used is the Galaxy Nexus device running Android 4.1. It was rooted to deploy the client-side parts of the system. However, after some modifications to our software, it would be possible to deploy it on Android 4.0 or greater, without requiring it to be rooted. This means that deploying our solution in practice would be even easier. We also required root access for configuring system routes for "simulated multi-interface" scenario.

One of the very important aspects of energy usage of the device is the brightness of the screen. The screen during the experiments remained on, as would be the case during regular use of the device. To make test results more consistent, the "auto brightness" feature was disabled, so the phone kept using the same brightness level during all of the experiments. Brightness was kept at the default level, which is half of the brightness range.

## 4.2 Control parameters

There are several control parameters that affect the behaviour of the system.

One of the most important parameters is the Wi-Fi network throttling. Since the experiments were performed in a very well connected environment, the bandwidth available over the Wi-Fi network was much greater than typically available in real life. It was also much greater than the bandwidth offered by 3G mobile network.

To make the results applicable to typical environments, the Wi-Fi access point was configured to throttle data throughput to about 5 Mbps, which is a value often available over DSL links. It was also similar to (although slightly lower than) the rates available over the mobile network.

Another parameter is the data set downloaded from the network. In many experiments we define a "task" as an operation that involves downloading a specific amount of data from the network. We chose the data size to be 100 MB. Since most of the scenarios require multiple TCP flows, the "task" we use for comparing different scenarios is downloading

4 data files of 25 MB each. This also means that in most cases we used 4 separate TCP flows operating in parallel.

One of the most important parts of the system is the hashing scheduler. It allows us to specify the "weights" for each of the available network interfaces, that represents the portion of data to be assigned to each of the interfaces. It is not possible to statically predict the parameters offered by the 3G network, so those weights could not be configured statically. It is possible to build a network quality monitoring system, that would configure and adjust these weights during the operation of the system. However, this feature is out of scope of this work. For these reasons we decided to assign equal weights to each of the network interfaces. This is sufficient to present features of the system and offers good results. Proper link monitoring methods and dynamic weight control could improve these results further.

The last of the control parameters we can adjust is the intervals at which the hash weight balancer runs. Typically it should be running every few seconds. However, to better present its impact on the results, in some of the scenarios we configure it to run less often.

## 4.3 Captured parameters

During all of the experiments there are three parameters that we capture and record.

The first captured parameter is the current time. Because some of the data is gathered on the client device itself, and some of it is captured on the desktop computer monitoring the state of the power supply, these results have to be merged. To make that possible, the time on both the desktop machine and the client device (the Android smartphone) are synchronized.

The testing software that actually performs the data transfers, monitors the data rate of each of the individual flows, and records them in the data file - separately for each test performed.

The desktop machine that is monitoring the power supply records the real time current drained by the client device every second. These data is later merged with the data rate information, to create a data log file that includes data rates and corresponding current drained values.

Based on these parameters all other values are obtained: total aggregated data rate, total energy used to perform a task, and the total time it took to complete that task.

## 4.4 Traffic parameters

The most important metric for evaluating the performance of the system is the bandwidth offered to flows sent over the system. It is not simply the amount of data sent over each of the interfaces, but the bandwidth that is experienced by TCP flows running on top of the system examined. We are not only looking at bandwidth of the individual flows (in the multi-flow scenarios), but also at the total bandwidth of the sum of all the flows in the system.

The way the test client interacts with the web server using multiple TCP flows transmitted over our system is presented in Figure 4.2.

Another important factor we are looking at is how much changes in the underlying networks affect userspace TCP flows. Any changes in the underlying networks (failures, device switching networks) will affect userspace flows. The shorter those disruptions last, the better.

The bandwidth experienced and the length of disruptions can be examined using infinite data transfers. However, we are also looking at the time it takes to perform specific "task". We always measure the data rates experienced by the TCP flows used by the test utility, but for comparisons with other scenarios we simply use the total time it took to complete all of these data transfers. The less time the whole task takes - the better.

The final effect on the traffic this system has is the additional latency introduced. When the tunnelling solution is used, there is additional processing of every data packet, on both the client device and the proxy server. This additional processing increase the overall latency. Another source of latency increase is the fact, that packets have to be sent between the client and the remote server over the proxy server, instead of taking the direct, most likely shorter, path. Even though the second part of latency increase is most likely significantly greater, it heavily depends on the placement of all the devices involved - the client device, the remote and proxy servers. Since there is no "typical", or even "worst case" scenarios, this part of latency increase cannot be meaningfully evaluated. Instead, only the latency increase due to additional processing with the proxy server on the "usual" data path is studied.

## 4.5 Energy consumption

The energy consumption, especially in mobile environment, is an important factor in the design of system like this. Even if the system offers amazing advantages in terms of

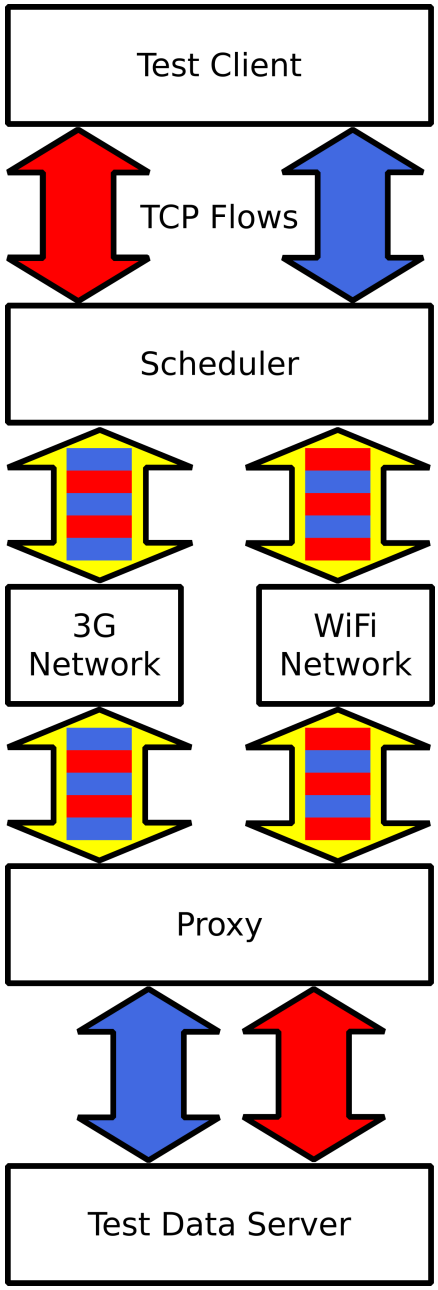



Figure 4.2: Test client using multiple TCP flows

increased bandwidth, fault tolerance and good mobility handling it cannot be deployed in the real life if it causes too much of energy usage increase. More energy used means shorter life on a single battery charge. Any solution, that decreases that time by too much, cannot be considered a practical one.

The metric for evaluating the system is the energy consumption. The solutions discussed require some additional data processing. Also, since in some of the scenarios multiple network interfaces are active at the same time, increased energy usage is unavoidable. However, by measuring that usage in the absence of the system presented here, and comparing it with the usage when specific algorithms are used, we can tell whether the difference is significant or not.

Similarly to the monitoring of traffic parameters, we can look at energy usage in two ways. One is to measure the energy consumption while performing specific tasks. This energy usage will certainly increase when using the multi-interface connectivity and algorithms. However, we can also measure the amount energy required to perform a specific task. We do this by transmitting a fixed amount of data, and measuring the total energy used during that task. In this case it is not clear anymore whether the proposed solution increases the energy consumption, or quite opposite - it actually saves the energy.

## 4.6 Experimental scenarios

There are several test scenarios examined on the same device in the same testbed environment. Different scenarios are designed to expose various characteristics of both unmodified device, as well as the device running the system develop and using different possible solutions.

### 4.6.1 Measured properties

During the tests, the same set of data files will be transmitted using several concurrent TCP connections. For each of experiments, the following properties are monitored and gathered:

#### The data rate of each of the individual TCP flows

This property shows how much of a bandwidth each of the TCP connections is experiencing, as well as any disruptions in the transmission

### Total, "aggregated" data rate

This property is simply the sum of all individual data rates of all the flows. It shows how much bandwidth, overall, is offered to userspace applications.

### Total time of downloading the entire data set

This property shows how much time it takes to complete a set of tasks. Shorter values mean better performance - less time the user has to wait for completion of various network operations performed by userspace applications.

### Energy consumption during the test

This parameter shows how much energy is drained from the battery each second while using specific features in various scenarios. It is a base for simple comparison of different scenarios and the cost of using different algorithms or different network interfaces.

It shows the base "price" the user is paying in the battery life for using certain features. The more data is processed and more network resources are used, the higher the energy consumption rate is going to be.

### Total energy used for completing the test

This parameter describes how much energy was used during the entire test. The simple "energy per second" parameter is likely only to go up while using more advanced algorithms and solutions. However, it is possible that even though energy usage rate goes up, especially while using multiple interfaces concurrently, the total energy used to achieve specific task (here - to download a set of data files) may as well go up (or not be affected by much).

By examining how much energy, in total, is used for completing the test, we can tell whether it is possible for the user of such a system to actually save some energy or not.

### Latency increase

This parameter describes by how much the total path latency (between the client device and the remote server) is increased due to additional data processing, tunnelling and forwarding when the proxy device is on (or very close) to the desired network path.

### 4.6.2 Single interface tests

These tests involve running the device with only one network type used at a time. Each of the scenarios is run for both Wi-Fi and 3G networks used individually.

To establish basic characteristics of the system and the experimental environment both networks are used for data transfer without any parts of the system running on the client device, except for the testing software itself. These tests are meant to obtain reference points and characteristics of unmodified device running in the testbed environment.

Both networks are also tested using the device with our system running. In single interface scenario there is no bandwidth aggregation possible. Because of that only the simplest solution is used - the priority based scheduler. However, since there is only one interface, the scheduler used does not make any difference. The purpose of this experiment is to establish the minimal possible cost of using the system, in both the increased energy consumption, as well as slight decrease in data rates available to the applications. That bandwidth costs is the consequence of using traffic tunnelling and data overhead associated with using additional protocol headers.

This test is also used to obtain the increase in path latency when our system is used. To do that, the regular "ping" application is used. The remote server should remain the same, and several tests with and without our system running are performed to obtain the increase in the latency reported by that utility.

### 4.6.3 Simple mobility scenario

This test is performed using only the simple priority scheduler. Since only one interface is used at a time, no bandwidth aggregation is possible. It also does not make sense to perform this tests without our system running, since in that case switching the networks would cause interruption of any existing TCP connection.

This test involves the device switching from Wi-Fi network to 3G network, as well as switching back from 3G to Wi-Fi. In this case the overall time to complete the test "task" is not relevant. The focus of this experiment is to examine the behaviour of TCP connection during hand-overs (and whether that TCP connection "survives" such an operation or not).

### 4.6.4 Multi-interface scenarios

The experiments in this category are performed with the device connected to both types of networks at the same time. They are performed using both "bare" client that is not running our system, as well as with both types of schedulers that offer bandwidth aggregation.

#### The "simulated" multi-interface scenario

This experiment simulates the multi-interface connectivity, without running the actual system developed. It requires the device to be forced to keep both (Wi-Fi and 3G) interfaces active, and force different flows over different interfaces using custom routing rules.

This test shows the costs (in terms of increased energy usage) of using both interfaces at the same time, without "paying" the price of running any additional services on the client device.

Since the interfaces are used independently, the total bandwidth is also the maximum bandwidth available to any bandwidth aggregation mechanism.

#### Packet based bandwidth aggregation

The first, round-robin based scheduler is tested during this experiment to demonstrate problems with the naive bandwidth aggregation method, and establish a reference point for comparison with the hash-based scheduler. This solution is not expected to perform well in real world conditions, and only performed as a point of comparison.

#### Hash based scheduler

This test is meant to demonstrate the performance of the actual, practical scheduler that offers bandwidth aggregation. The performance of the system (as experienced by the userspace TCP flows) is analyzed and compared to both simulated, bare system results, as well as to the simplistic, packet-based algorithm. The costs in increased energy usage are also analyzed and compared to other scenarios.

#### Dynamic hashing adjustments

The hashing scheduler is expected to perform well in typical conditions. However, as discussed in Section 2.4, it is possible that the basic hashing algorithm will not be able to

offer much improvement in terms of aggregated bandwidth due to specific parameters of the traffic.

This experiment is performed using carefully configured TCP connections that expose this problem. This situation is possible also in real life, but it is hard to predict. Instead of performing large number of tests and selecting those that experienced skewed traffic parameters, we force the system to create TCP connections that expose this problem.

For this test all TCP connections are configured to use specific source port numbers - the configuration that causes the hashing algorithm to assign all the flows to the same network interface. Without the hash adjustment algorithm only one interface would be used for data transfers, and the total bandwidth would be limited by that single network.

This version of the scheduler will be able to reallocate TCP flows to different interfaces despite very specific, "mean" port number configurations - thanks to the hash weight adjustment algorithm.

# Chapter 5

# Validation

This chapter includes the results of experiments performed presented in the form of charts, as well as discussion of the results observed. All the experiments are performed using the testbed presented in Figure 4.1.

## 5.1 Single interface tests

Several control tests involve running the device with only one network type used at a time. They are performed to establish basic characteristics of the system and the experimental environment.

### 5.1.1 Latency test

The system performs data tunnelling between the client device and the proxy server to achieve its goals. This solution offers several benefits and has some associated costs as described in other sections of this document. One of them is the increased latency, or round trip time between the applications running on the client device and the remote network servers these applications use. The reason is that the data is not travelling directly to the destination, but is sent through the proxy which, in almost all cases, will increase the length of the network path. The increase in latency depends heavily not only on the placement of the proxy, but also on the location of the specific servers that client applications are using. This means that this factor cannot be "measured". However, there is additional overhead

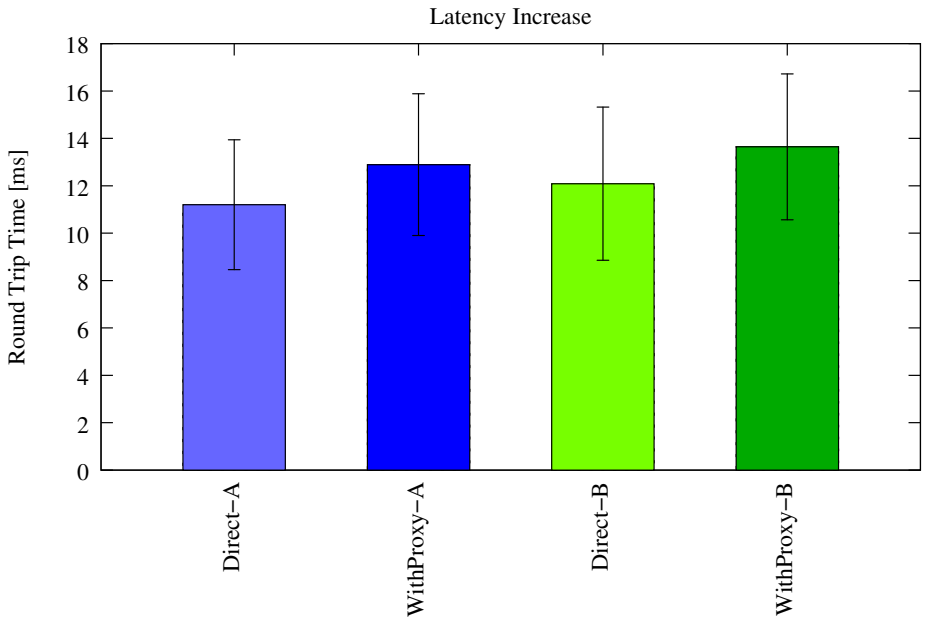


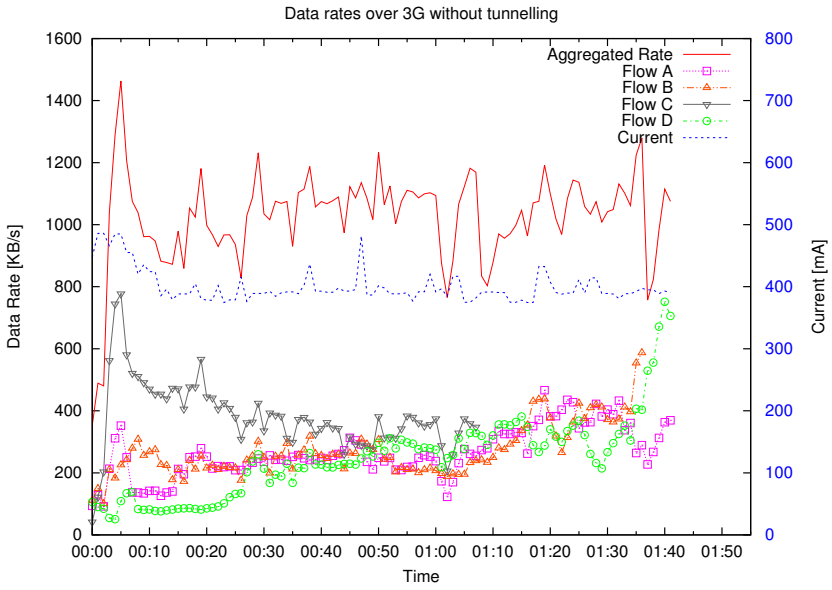

Figure 5.1: Latency increase due to data tunnelling

related to the fact that all data is additionally processed by both the client device and the proxy. This factor does not depend on the location of the proxy or remote servers.

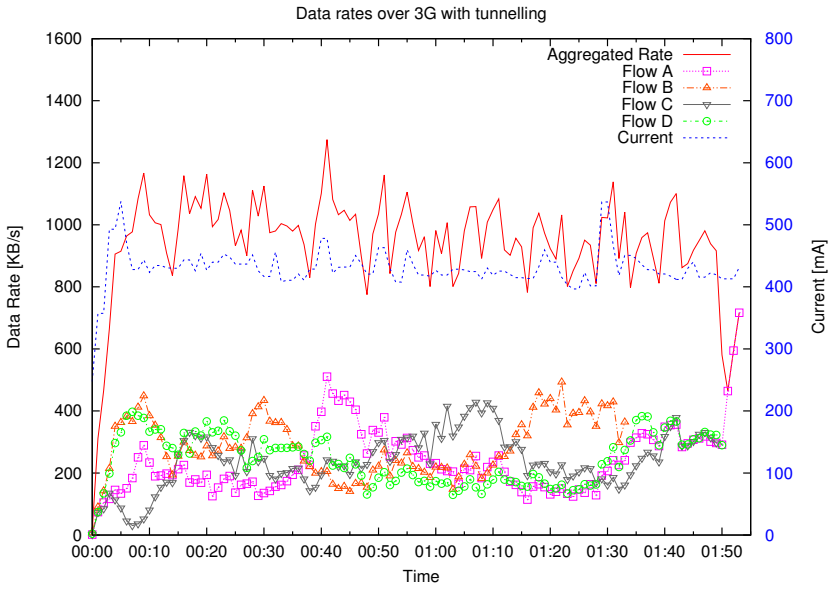
To measure that part of latency increase, a simple test was performed. The client device, Galaxy Nexus, was pinging a public interface of the proxy server - with and without the tunnelling. This means that the ping packets were travelling exactly the same distance in both cases. The difference was the additional processing and packet forwarding between the interfaces.

All the tests were performed only over Wi-Fi interface. The interface choice does not affect the latency caused by increased data processing, and Wi-Fi interface is significantly more stable than 3G interface. Each of the tests involved sending 100 ping packets, both with and without our system running. Two series of tests were performed at different times of day. Results are presented in Figure 5.1.

The measured latency increase is, as expected, very small, and within roughly 2 ms above the latencies observed without the tunnelling. This amount of latency increase should have practically no effect on data flows running over our system.

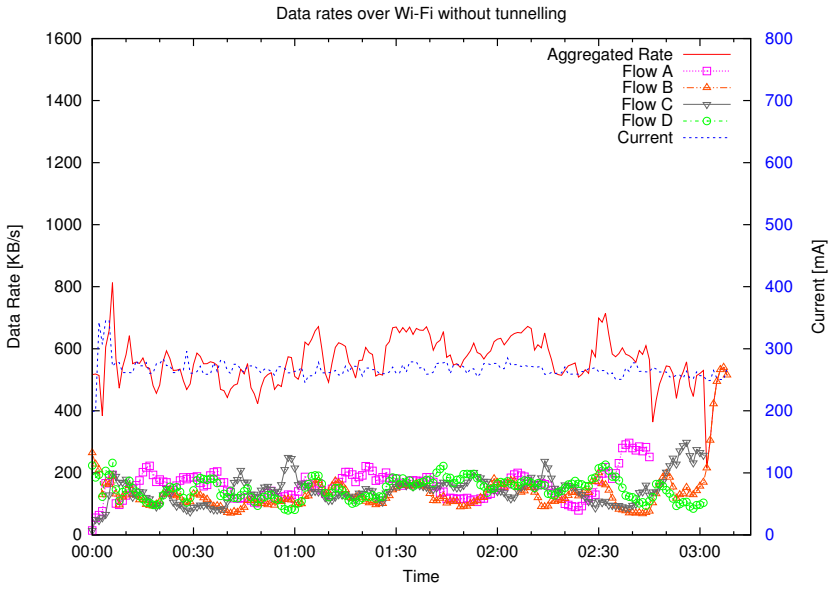



(a) No Tunnelling

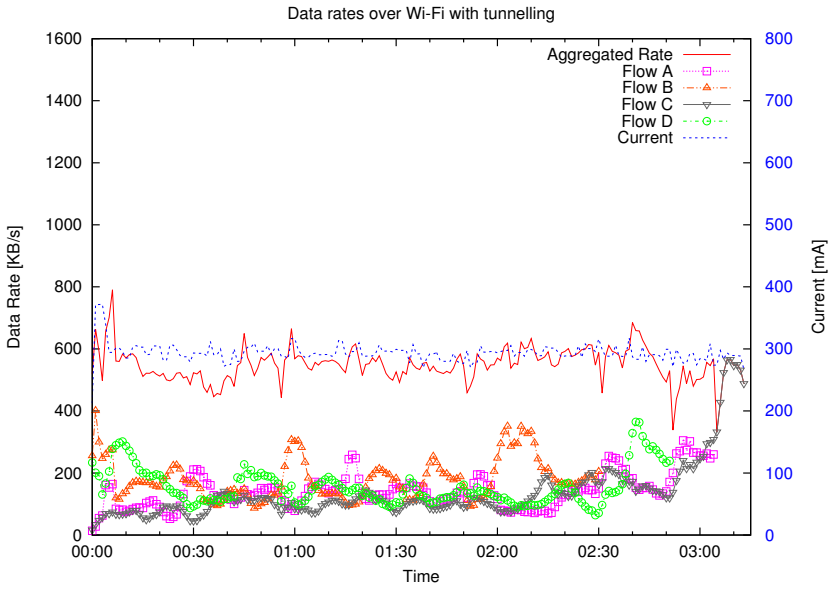



(b) Tunnelling

Figure 5.2: Data transfer over 3G




(a) No Tunnelling




(b) Tunnelling

Figure 5.3: Data transfer over Wi-Fi

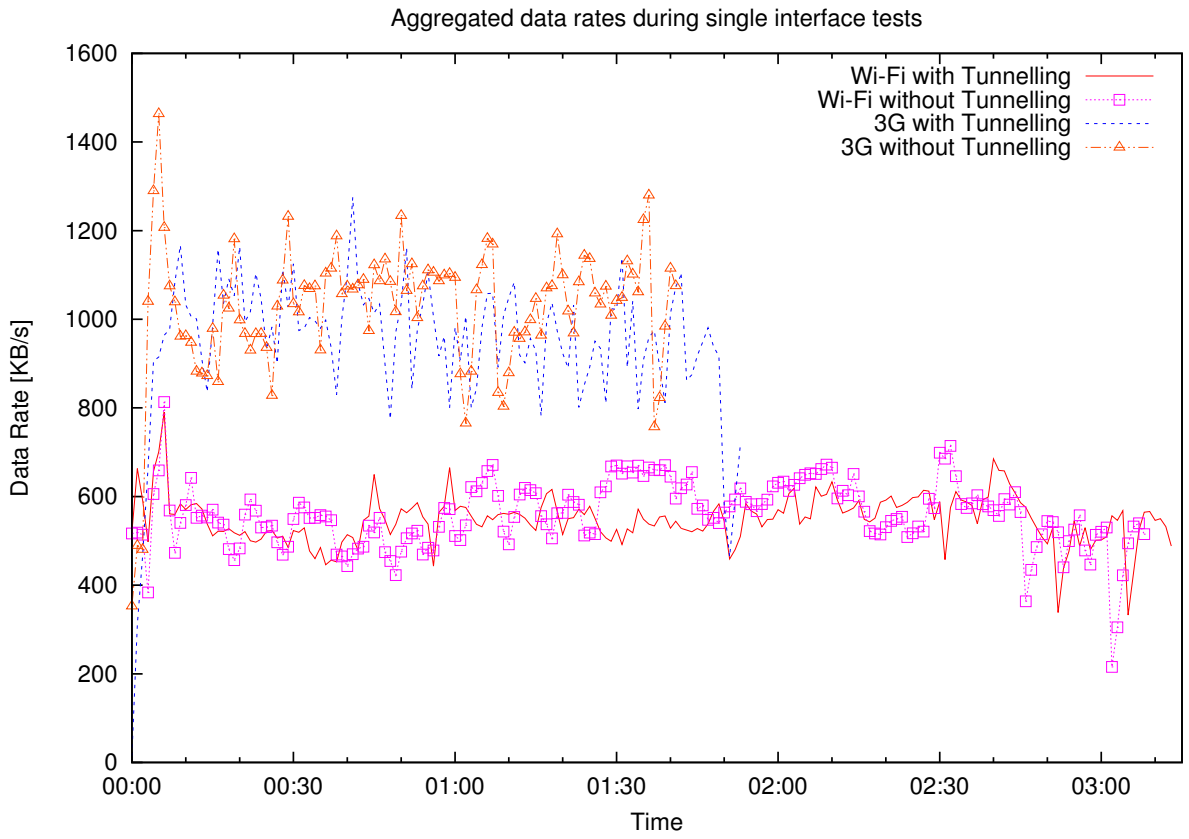



Figure 5.4: Comparison of aggregated data rates during four different single interface scenarios

### 5.1.2 Data transfers

This basic scenario involves downloading the entire test data set - four 25 MB files - over a single interface. This test is performed with and without our system running, to observe basic data and energy overhead of our solution. Both Wi-Fi and 3G interfaces are tested.

Graphs depicting data rates and energy usage during experiments in various scenarios are presented in Figure 5.2 and Figure 5.3. These graphs include data rates of each of the TCP flows, the aggregated, total data rate, as well as energy usage. The X axis shows the time of the test, the left Y axis shows the data rates (in KB/s) and the right one - the energy usage (in milliamps per second).

For easier comparison of data rates achieved in all four cases, all the aggregated rates

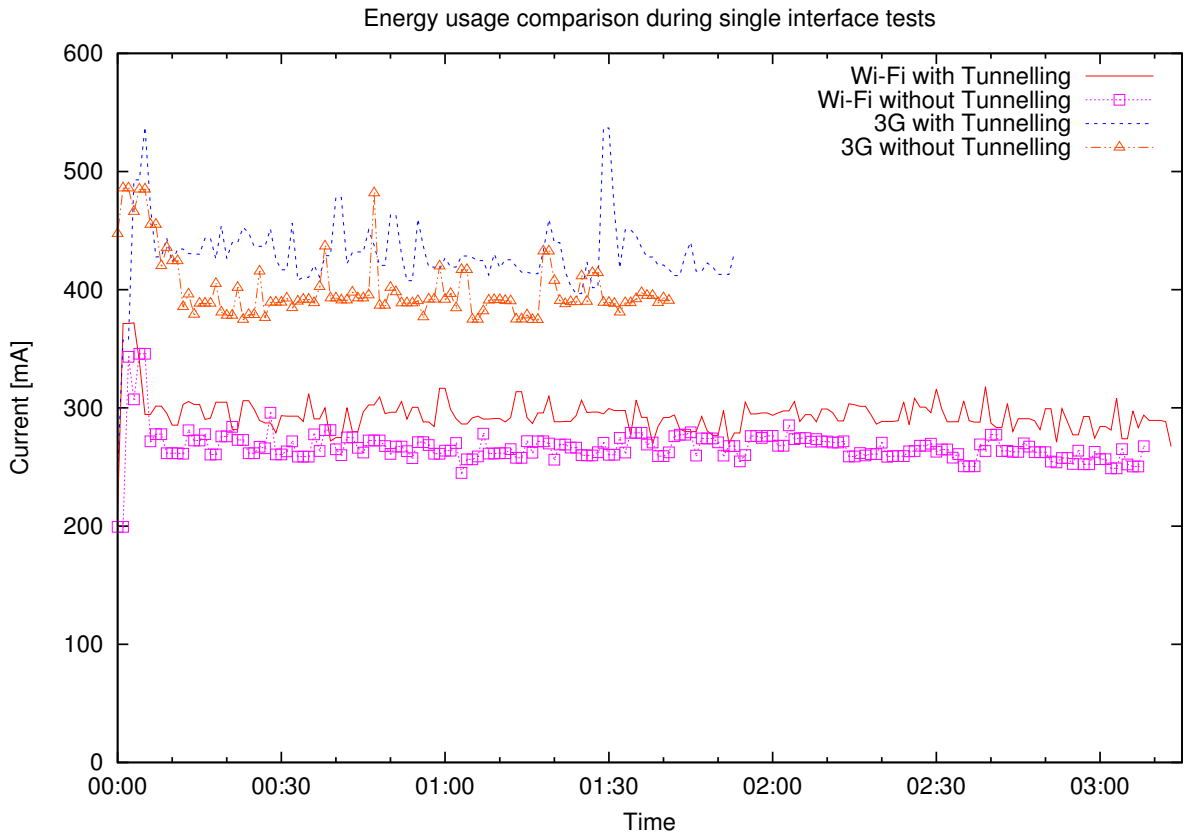



Figure 5.5: Comparison of energy usage during four different single interface scenarios

are also included in Figure 5.4. Analogous comparison of energy usage traces is presented in Figure 5.5.

The general behaviour of the system is presented using figures that include only one trace for each scenario. However, several experiments were performed for each of the scenarios. The aggregated mean values for both data rates and energy usage are presented in Figure 5.6. It includes the completion time of the test task (downloading four 25 MB files), as well as the total energy used to perform it in each of the scenarios considered.

The analysis of the results shows that the 3G network offered greater bandwidth than the Wi-Fi network - this is to be expected given testbed configuration assumptions presented in Chapter 4. At the same time, the detailed traces of individual tests show that the parameters of the 3G network are less stable than those of Wi-Fi network. This is

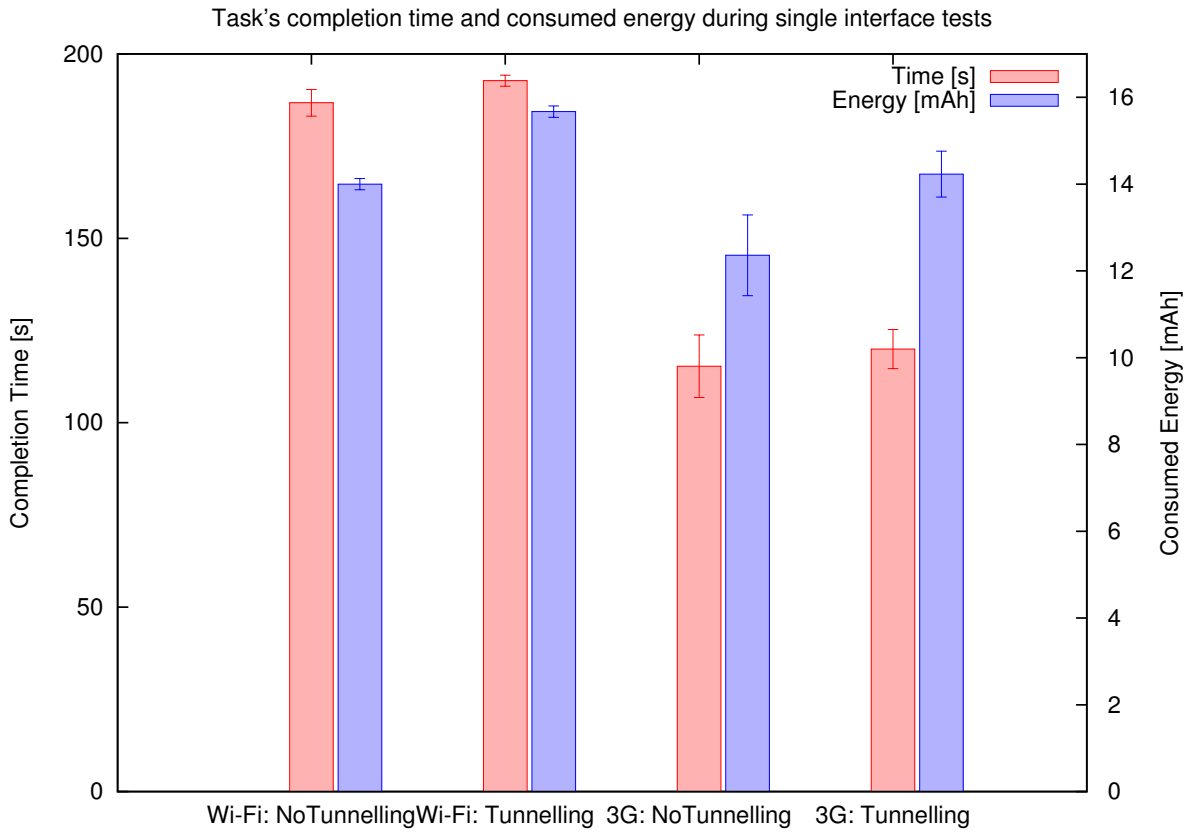



Figure 5.6: Completion time and energy usage of single interface experiments

consistent with the general observations of the behaviour of mobile networks.

The comparison of analogous scenarios with and without tunnelling (with and without our system being used) shows the cost of using the tunnelling - slightly increased completion time and increased energy usage.

The completion time increase is the result of slightly decreased maximal data rates. This is expected consequence of using any tunnelling mechanism, due to additional headers and overall increase in protocol overhead. In this case the completion time differences are within 6 seconds.

Increased energy usage is partially the consequence of the increased task completion time. Because it takes longer to download the test data set, the phone used more energy. Another reason for increased energy consumption is the increased amount of data processing that phone has to do when running our system. The total difference in energy consumed is less than 2 mAh, which is about 16% of increase. This difference may be seen as significant, but the system is meant for multi-interface environment, where it shows its strength. Also, downloading 100 MB of data from the network is very data-intensive operation, so any additional data processing will significantly increase the energy usage.

### 5.1.3 Simple mobility scenario

The simple mobility scenario is meant to present system's behaviour during network hand-overs. It only makes sense to perform this experiment with our system running, since switching networks without it simply interrupts any data connection that was started before the hand-over.

During this test the test applications performs one long TCP data download, while the device is switched from Wi-Fi network to 3G network and back. The focus of this experiment is to examine the behaviour of TCP connections during hand-overs (and whether these TCP connections "survive" such an operation or not), and not the time to complete the test task. Multiple time and energy results gathered during separate runs of this test would be hard to compare, as the exact timing and behaviour of the device during switching between different networks is very difficult to replicate.

The behaviour of a single TCP flow is presented in Figure 5.7. It is easy to spot the times at which the hand-overs took place. First, the device is using the Wi-Fi network. The data rates are between 600 and 800 KB/s. Around 45th second of the experiment, the device was switched to 3G network. The increased energy usage is the consequence of turning on the 3G interface and additional operations the device had to perform. During

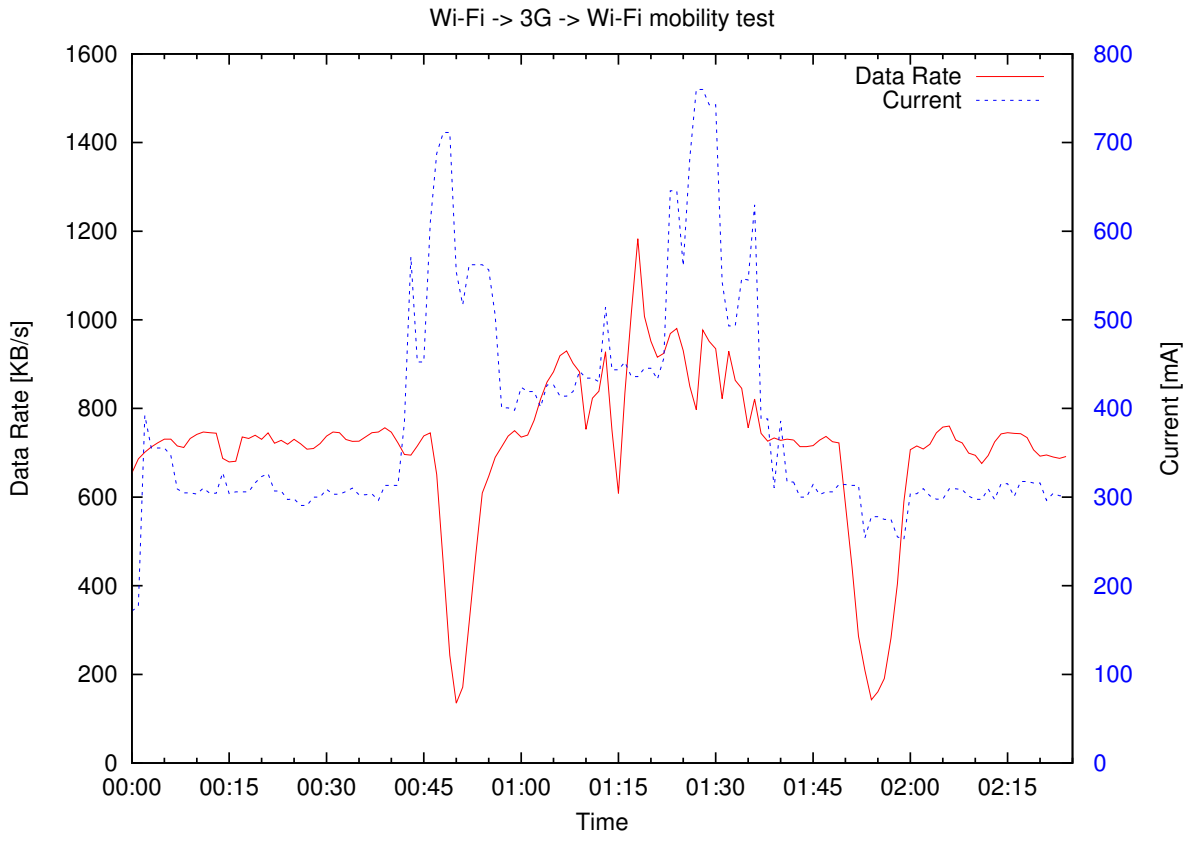



Figure 5.7: TCP flow behaviour during network hand-overs

the hand-over the data rate drops significantly. This is because the switch between the networks was very rapid. In real life conditions, where networks do not disappear so abruptly but rather degrade over some period of time. For those situations it is possible to develop a scheduler that monitors the network quality and handles hand-overs in more seamless way. But even this test, when the switch happened instantly, the TCP flow survived this operation and was able to recover quickly. If our system was not in place, the TCP data transfer would be interrupted completely just after switching the networks.

After the 50th second of the experiment, the TCP flow recovers and reaches the bandwidth offered by the 3G network - higher than bandwidth of the Wi-Fi network, but less consistent. After another 45 seconds the device was switched back to Wi-Fi network. This operation results in another drop in the data rate but, again, the TCP flow recovers quickly and restores the rate observed before the first network switch.

This test proves, that despite short disturbances in the TCP performance, our system can handle transparent flow migration between different networks.

## 5.2 Multi-interface scenarios

The experiments in this category are performed with the device connected to both networks at the same time. They are performed using both "bare" client that is not running our system, as well as with different schedulers that offer bandwidth aggregation.

### 5.2.1 The "simulated" multi-interface scenario

This experiment simulates the multi-interface connectivity, without running the actual multi-interface system we developed. We use it to observe the performance of the network operations while using multiple interfaces in parallel without any overhead that may be introduced by our system. We also want to use optimal static flow assignment. We can do that because we know, in advance, the remote addresses of the TCP flows that will be opened during the experiment. This test requires the device to be forced to keep both (Wi-Fi and 3G) interfaces active. During the experiment four TCP flows were used to two different remote servers (two TCP flows to each of the servers). System routing rules were manually added on the client device to force these TCP flows to use different physical interfaces.

This test shows the costs (in terms of increased energy usage) of using both interfaces

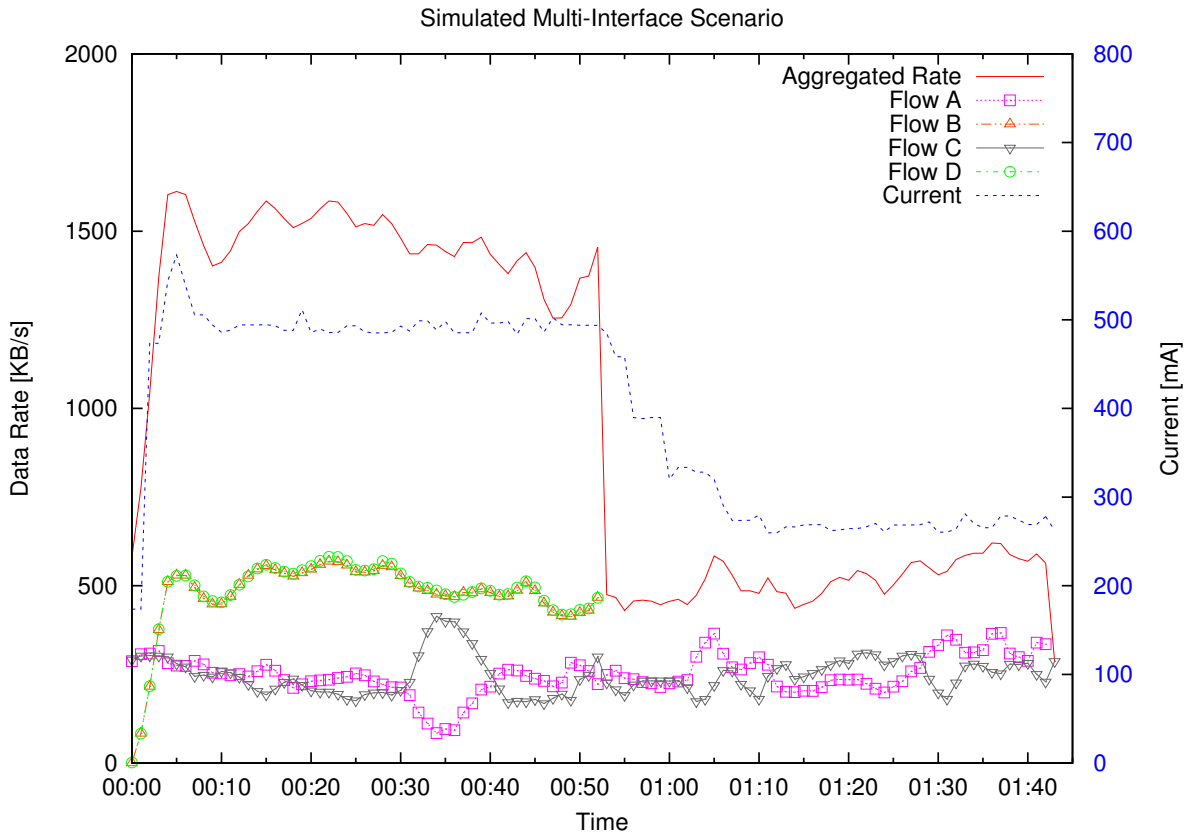



Figure 5.8: TCP flows during simulated multi-interface test

at the same time, without "paying" the price of running any additional services on the client device.

The traces of a single test are presented in Figure 5.8. During this test flows A and C were sharing the Wi-Fi interface, and flows C and D were running over 3G network. The total, aggregated bandwidth is, during the first 50 seconds of the experiment, greater than bandwidth offered by both Wi-Fi and 3G when used individually. Since the interfaces are used independently, this is also the maximum bandwidth available to any bandwidth aggregation mechanism. This, however, does not mean that this method achieves the fastest task completion times, or that no scheduler can do better than that. This is caused by the fact that this is a very static method. During this test flows B and C, that were running over 3G network, finished much earlier than remaining two flows. Because there is nothing that could migrate flows between interfaces, during the second part of the experiment only the flows that started (and continue) running over Wi-Fi network are active. The aggregated data rate is equal to the bandwidth available over that network. During that time, the capacity of the 3G network is wasted.

This test also shows energy usage levels for using both interfaces in parallel (during the first half of the test) and using only the Wi-Fi network (during the second half of the test). It clearly shows that using both interfaces at the same time is significantly more expensive (in terms of per-second energy consumption) than using the Wi-Fi interface on its own. However, the per-task energy usage is not necessarily always greater than when only one interface is used. The comparison of per-task energy and time parameters is included in Section 5.2.5.

Even ignoring the fact that flow migration is impossible, the very nature of this method makes it impractical to use in real life scenario. The extensive configuration requires knowledge of future traffic parameters which is possible in test scenarios, but not in real life. This method is only used as a reference point for other, practical multi-interface approaches.

### 5.2.2 Packet based bandwidth aggregation

This section presents the behaviour of TCP flows when the Round Robin scheduler (discussed in Section 3.6) is used. The traces of a single test are presented in Figure 5.9.

The comparison with other multi-interface methods (based on a number of test runs) is included in Section 5.2.5, but even brief inspection of the results suggests very high energy usage, and relatively low aggregated transmission rate. It also looks like the rates of individual flows are not stable and change a lot. This is what we expected from a simple

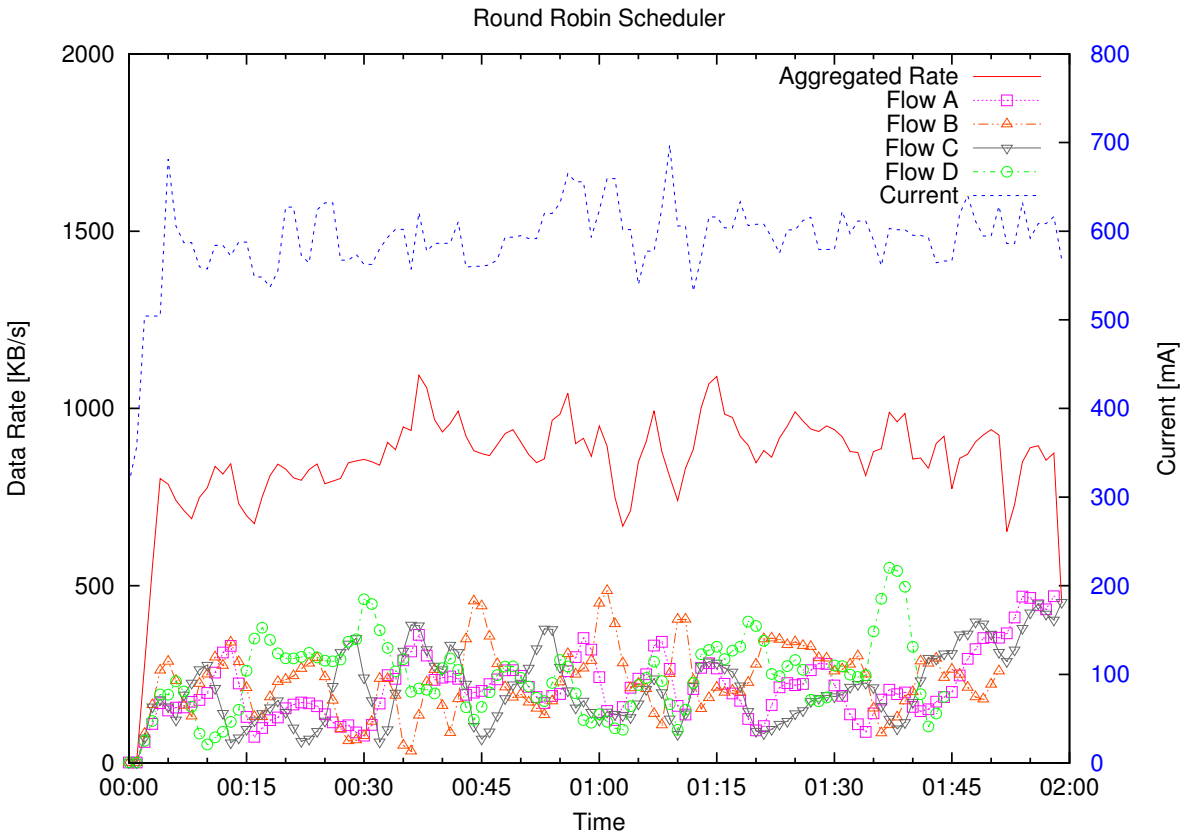



Figure 5.9: Round-Robin scheduler

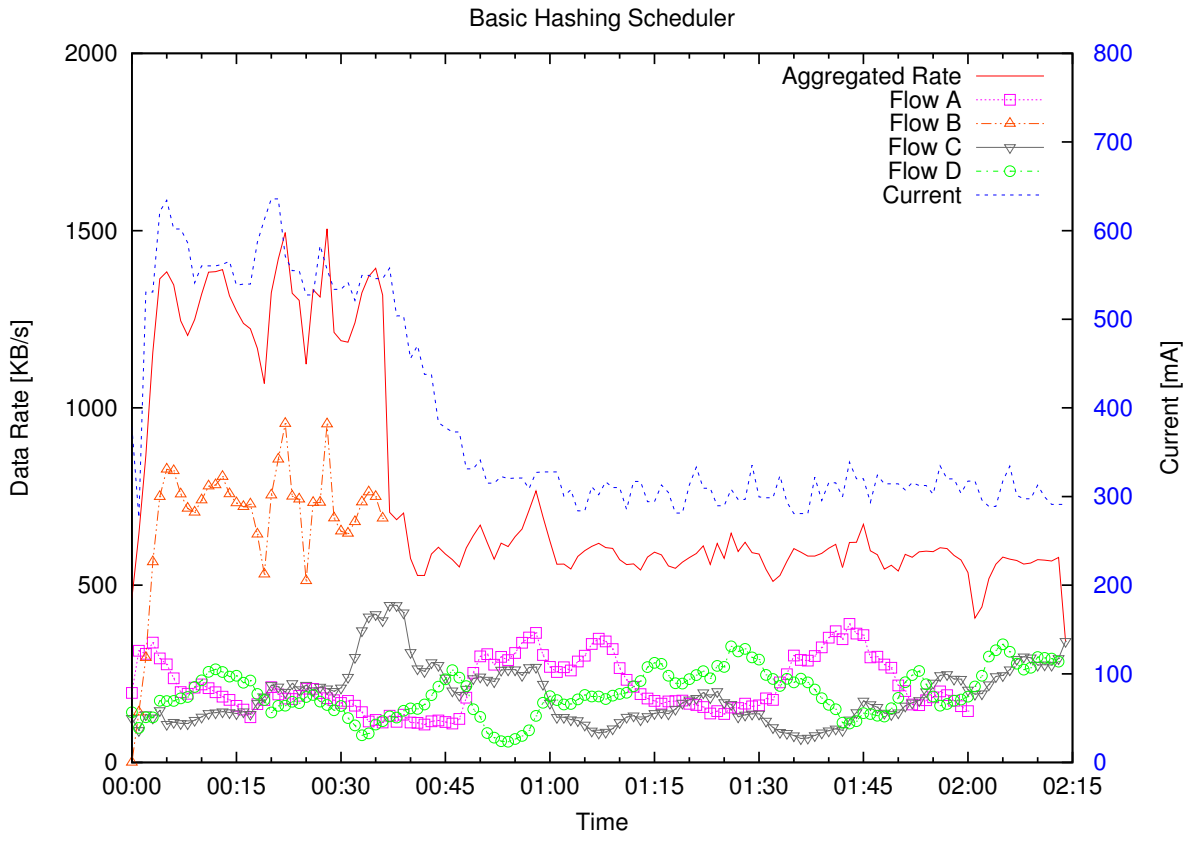



Figure 5.10: Hashing scheduler - "bad" traffic scenario

packet-based scheduler and is a result of TCP back-off algorithm in presence of increased packet reordering.

### 5.2.3 Hash based scheduler

This scenario presents performance of the hashing scheduler, which can achieve bandwidth aggregation in real life scenarios. Contrary to the "simulated" multi-interface connectivity, it does not require knowledge of the future traffic parameters, and yet is able to distribute flows between different interfaces. It also provides all the features of tunnelled approach, which means traffic migration between different networks when they appear and disappear.

This scheduler shares the main disadvantage of the static, simulated solution - it cannot

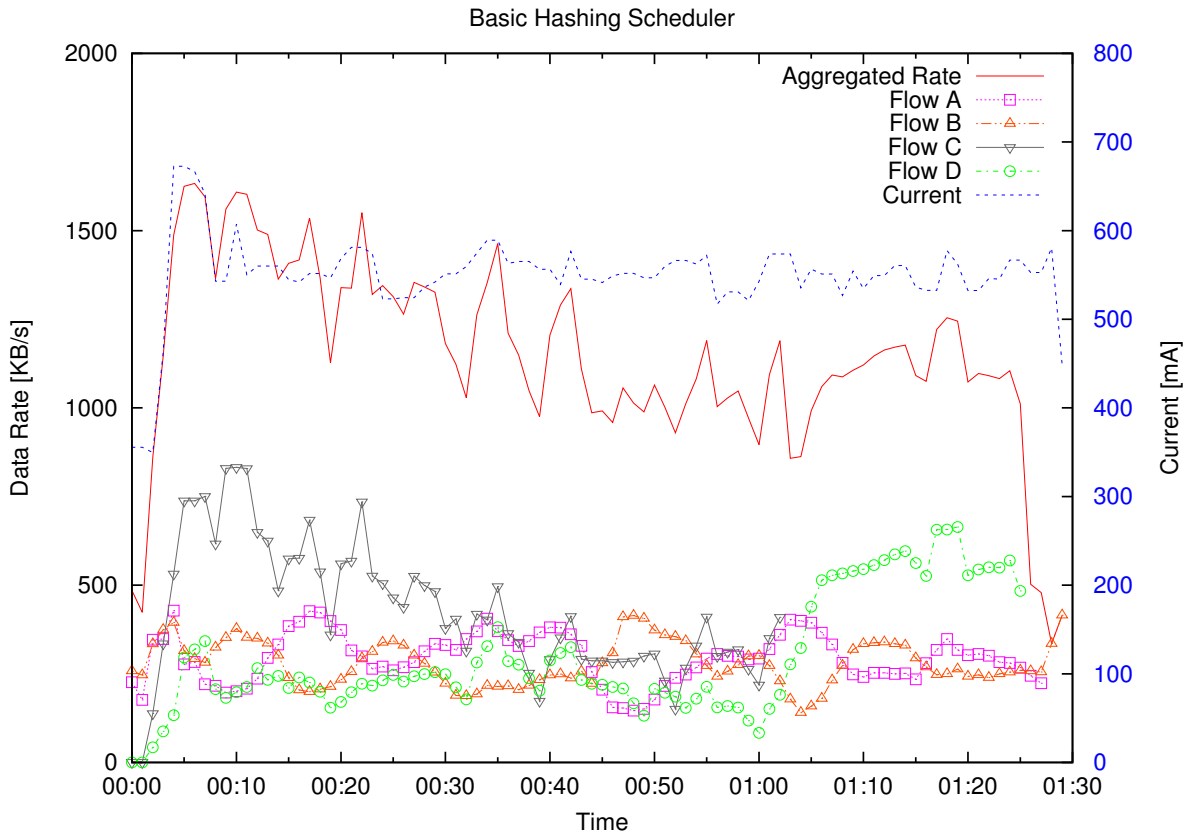



Figure 5.11: Hashing scheduler - typical behaviour

adapt to changing traffic conditions. This problem is presented in Figure 5.10. During this test, only one TCP flow was scheduled over the 3G interface. Once it was finished, the 3G network was not used anymore, causing the remaining TCP flows to run slower, and the task completion time to be longer. The energy usage characteristics are also similar to the "simulated" scenario.

In most cases, however, the traffic assignment is different and allows all available interfaces to be utilized better. The typical behaviour is presented in Figure 5.11.

These two traces were captured using exactly the same configuration of the scheduler. The difference was the traffic characteristics, more specifically - the source port numbers selected for the TCP flows. This is a parameter on which, in real life, we have no control. Typically, as the number of separate connections grows, the flow assignment becomes

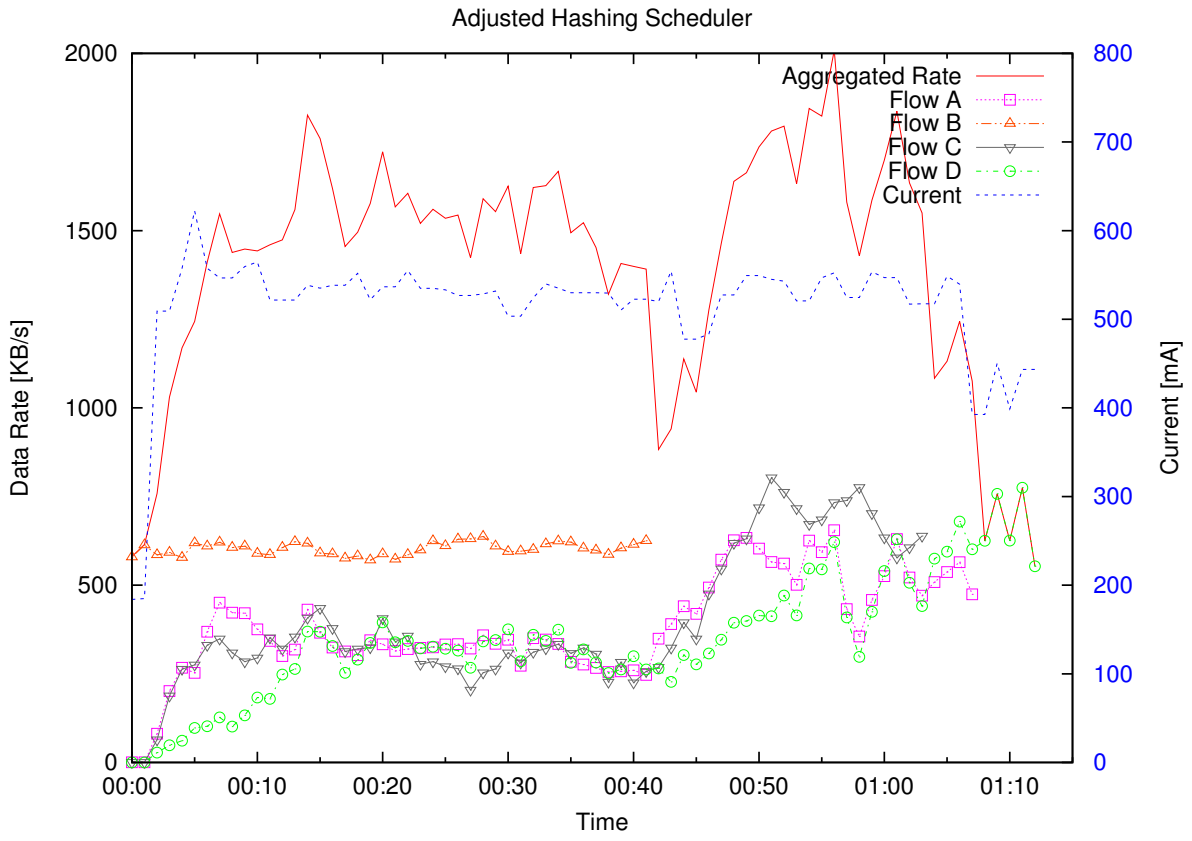



Figure 5.12: Hashing scheduler with weight adjustments

better. And conversely, as the number of different flows decreases, the assignment may become less optimal. Also, when there is only one flow, this scheduler (as well as any other flow-based scheduler) will not provide bandwidth aggregation.

### 5.2.4 Dynamic hashing adjustments

The problem exposed by some of the tests performed with the hashing scheduler, the inability to dynamically adjust to changing, or "bad" traffic characteristics, can be solved by using an additional algorithm. This algorithm is responsible for adjusting interface weights used by the hash scheduler to achieve the desired traffic split between different interfaces. The design of this algorithm is described in detail in Section 3.7.3.

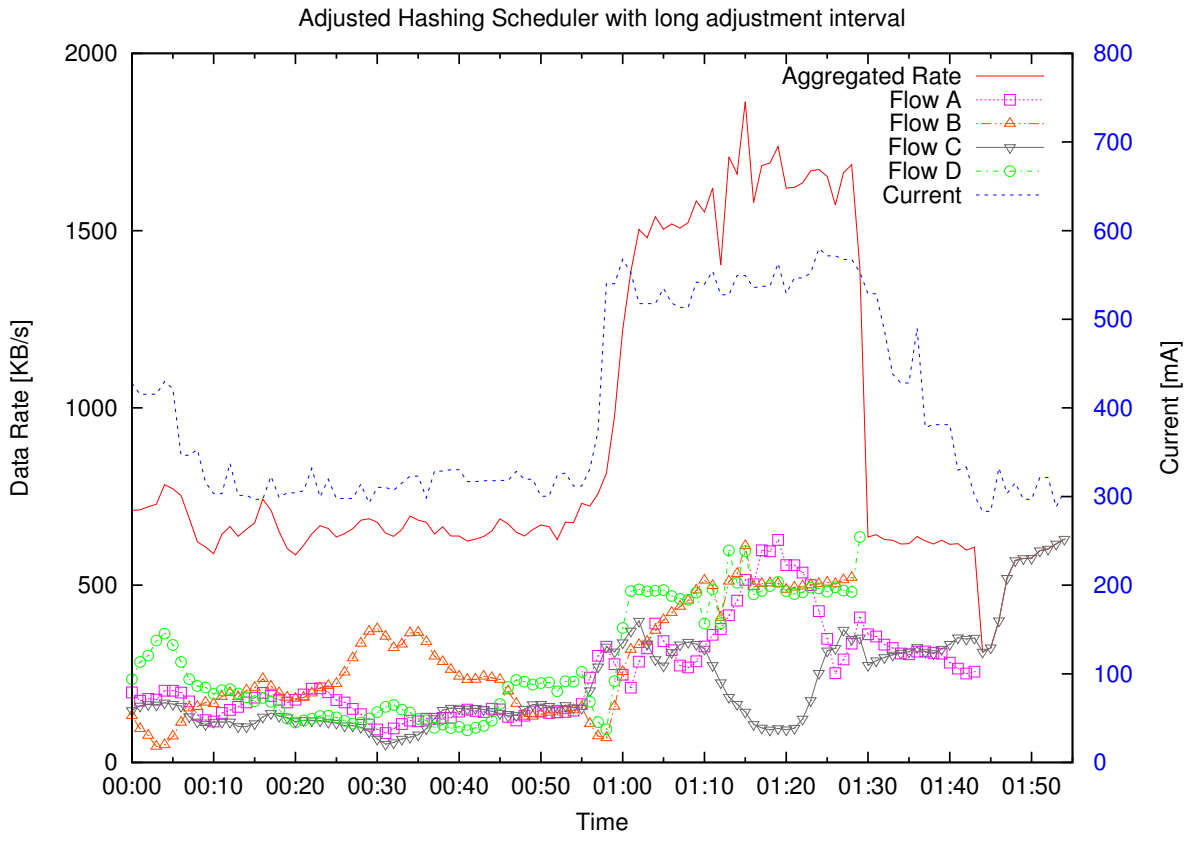



Figure 5.13: Hashing scheduler with infrequent weight adjustments

Figure 5.12 presents the trace of a single test. During that test, only a single flow (flow B) was scheduled over the Wi-Fi network. Remaining three flows were running over 3G network. This is analogous to sub-optimal assignment presented in Figure 5.10. The difference between these two situations can be observed around the 40th second of the experiment. Once the flow B is finished, the aggregated data rate drops. A few seconds later, however, the adjustment algorithm detects the problem, and adjusts hash weights causing one of the remaining flows, to be switched from the 3G network to, now unused, Wi-Fi network. Aggregated data rate goes back up, and both networks are used again, until almost the end of the entire experiment. As a result, the task completion time is almost twice shorter.

The role of the weight adjustment intervals can be observed in Figure 5.13. During this test the weight adjustment algorithm was configured to run less frequently. Instead of

running every 10 to 20 seconds, it was set to run once every minute. To make sure, that the initial flow assignment is sub-optimal, the specifically selected TCP source port numbers are used. We call them "mean" port numbers, as they cause the hashing scheduler to assign all the flows to the same network interface. This trace shows that until the adjustment algorithm was run for the first time, around the 60th second of the experiment, all the TCP flows were indeed using the same network. If this was the "bare" hashing scheduler, this sub-optimal assignment would remain until the end of the experiment. However, once the adjustments were performed, some of the flows were migrated to the second link and until they finished both networks were used at the same time. After these two flows ended, the assignment became sub-optimal again. Since the adjustment did not have the chance to run again, this behaviour did not improve. However, if the simulation continued, as long as there are at least two separate flows, some of them would keep getting migrated to the other network.

This experiment was performed only to better show the way the hash weight adjustments work. Such an infrequent adjustments are not meant to be used in practice, and so this scenario is not used during comparisons of different multi-interface approaches in Section 5.2.5.

The repeated flow reassignments are presented in Figure 5.14. In this experiment, we used specifically selected source port numbers ("mean" port numbers) to force the sub-optimal flow assignment, but increased frequency of weight adjustments. This time the algorithm was run every 5 seconds. The trace shows that at the beginning, for the very brief period of time, all the flows are using the same interface. This is quickly fixed by the hashing balancer, which migrates one of the flows (flow B) to the second interface, causing the aggregated data rate to go up significantly. Once the flow B is finished, around 35th second, the aggregated data rate drops, as the second interface is not used anymore. This assignment is, again, quickly fixed by the balancer, which causes flow D to be migrated to the second interface. This behaviour repeats again, around 60th second of the experiment. It looks like this time it took longer to restore the optimal flow assignment. The adjustment algorithm, described in detail in Section 3.7.3, increases its resolution every time it runs. In this situation the two remaining flows were so close to each other in the hashing space, that it took two, or even three iterations for the adjustment algorithm to be able to fix the flow assignment. Eventually, however, it managed to restore the proper network utilization. The situation presented here is possible, but very unlikely to happen in real life. Exposing this behaviour for this experiment requires very careful, specific source port assignment.

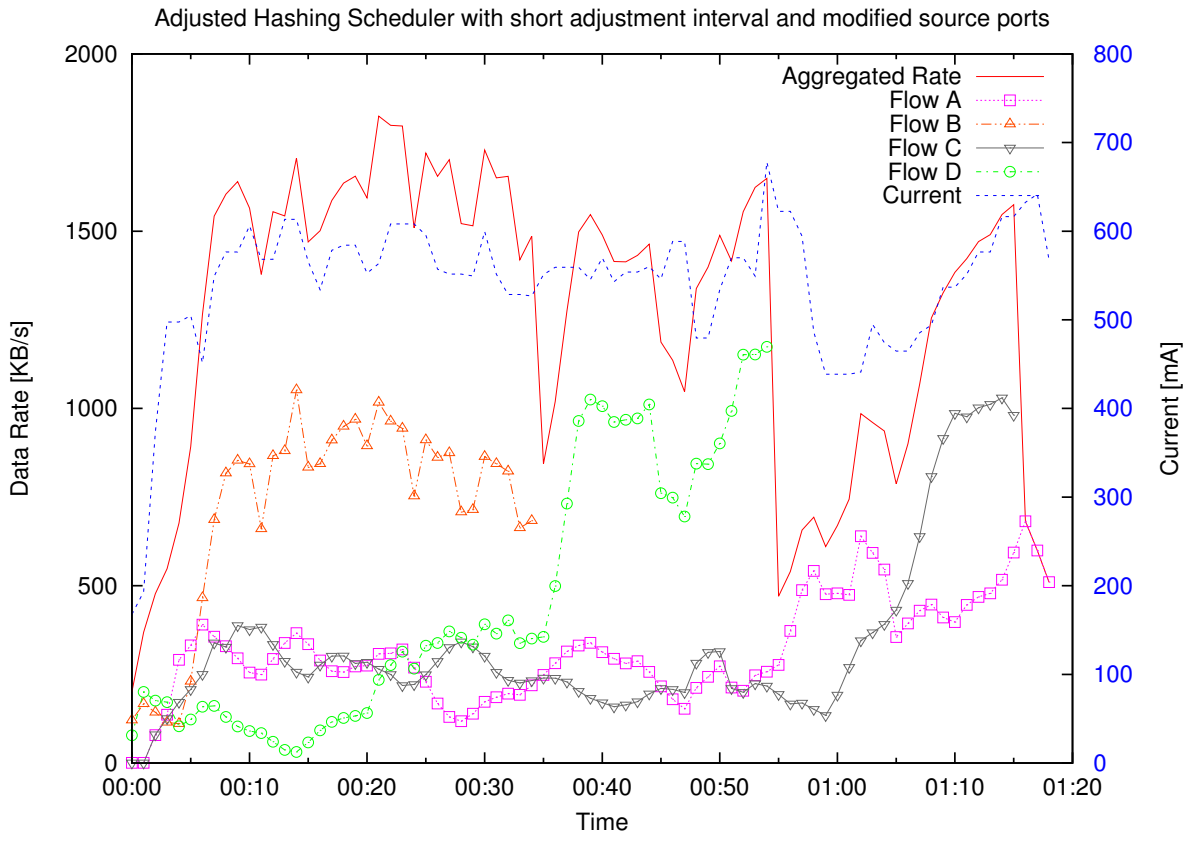



Figure 5.14: Hashing scheduler with very frequent weight adjustments

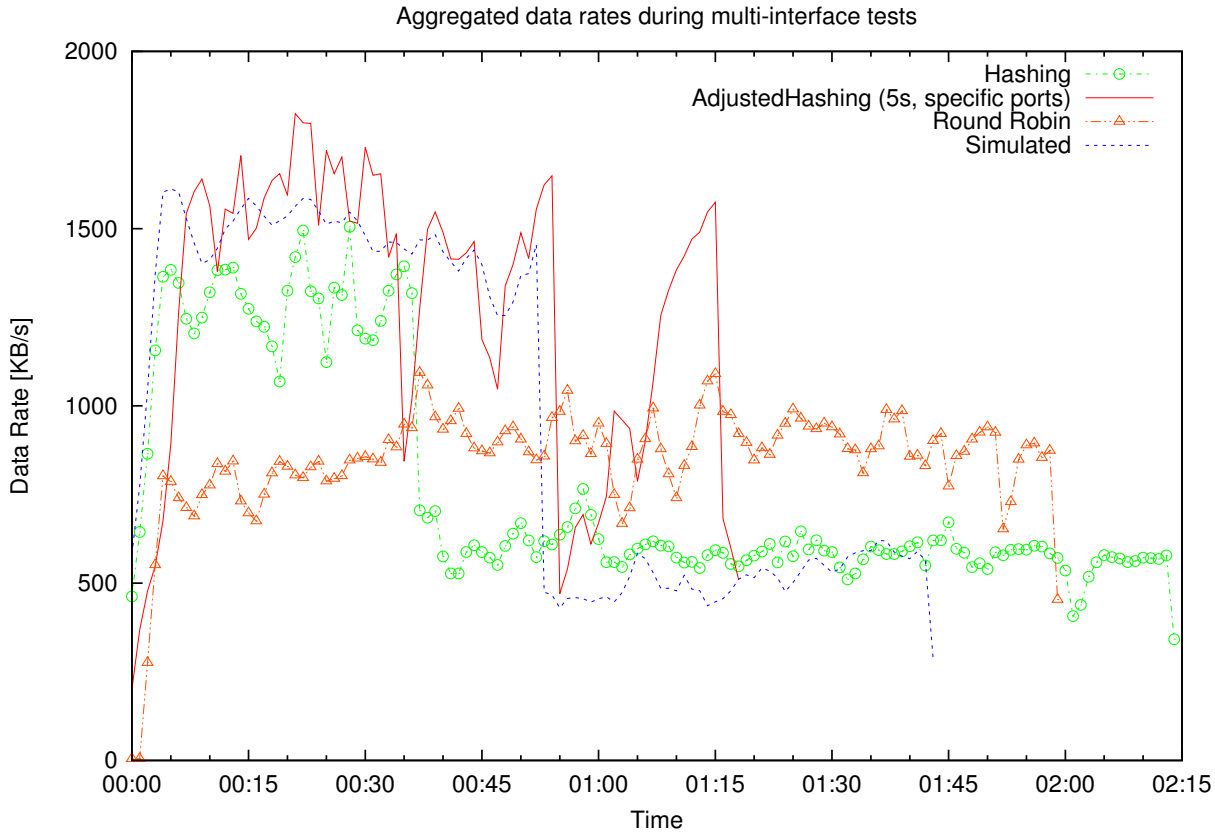



Figure 5.15: Comparison of aggregated data rates during different multi-interface scenarios

### 5.2.5 Comparison

This section presents comparisons of different multi-interface solutions. It also includes some results obtained using single interface experiments.

#### Data rate traces

The first part of the comparison of different multi-interface scenarios includes data rate traces of the experiments discussed in previous sections. Figure 5.15 includes the comparison of aggregated data rates in different scenarios. Different task completion times can also be observed using that graph.

For the basic hashing scheduler the scenario with "bad" port assignment is used to show the similarities with the simulated multi-interface scenario. Both of these scenarios have similar traces. At the beginning both networks are used, and once the flows running over the faster networks are finished, both experiments are reduced to single interface cases. The difference is, that in the simulated scenario, we make sure that two flows are scheduled over each of the interfaces. In the hashing scheduler scenario, it so happened that only one flow was scheduled on one of the interfaces. This caused the aggregated data rate to drop to the level offered by a single interface sooner, resulting in the longest completion time.

The round robin scheduler offered data rates slightly higher than the maximum bandwidth available on the slower link. However, we can see that the slower link offers abut 500 KB/s, and aggregated bandwidth is around 1500 KB/s. This means that during these tests the bandwidth offered by the other network was close to 1 MB/s. The round robin scheduler offers about 800-900 KB/s which is faster than slower of the two networks. However, it is lower than the bandwidth offered by the faster network. Considering that this scheduler was using both of them (which also becomes evident when inspecting the energy usage traces in Figure 5.17), those results are poor. In fact it would be better to only use the 3G network, than using both with round robin scheduler.

To clearly demonstrate the shortcomings of simple packet-based scheduling, Figure 5.16 presents comparison of the round robin scheduler with single interface cases. Even considering that data rates offered by the round robin scheduler ale close to those of the 3G network and significantly higher than those of the Wi-Fi network, there clearly is no bandwidth aggregation.

The simulated multi-interface scenario, even though it was described as one uses "maximum bandwidth available to any bandwidth aggregation mechanism" is not the winner of this comparison. Due to a better handling of dynamically changing conditions, the hashing algorithm with hash weight adjustments is the clear winner. Even though the scenario presented here used the specially configured source port numbers (resulting in sub-optimal flow allocation), this scheduler outperformed other solutions. It offered the highest aggregated data rates for the longest duration, and resulted in the shortest task completion time.

### Energy traces

The second part of the comparison discusses different energy consumption traces gathered during different experiments. Figure 5.17 includes comparison of the same multi-interface solutions as discussed in the first part. The results show energy usage pattern similar to

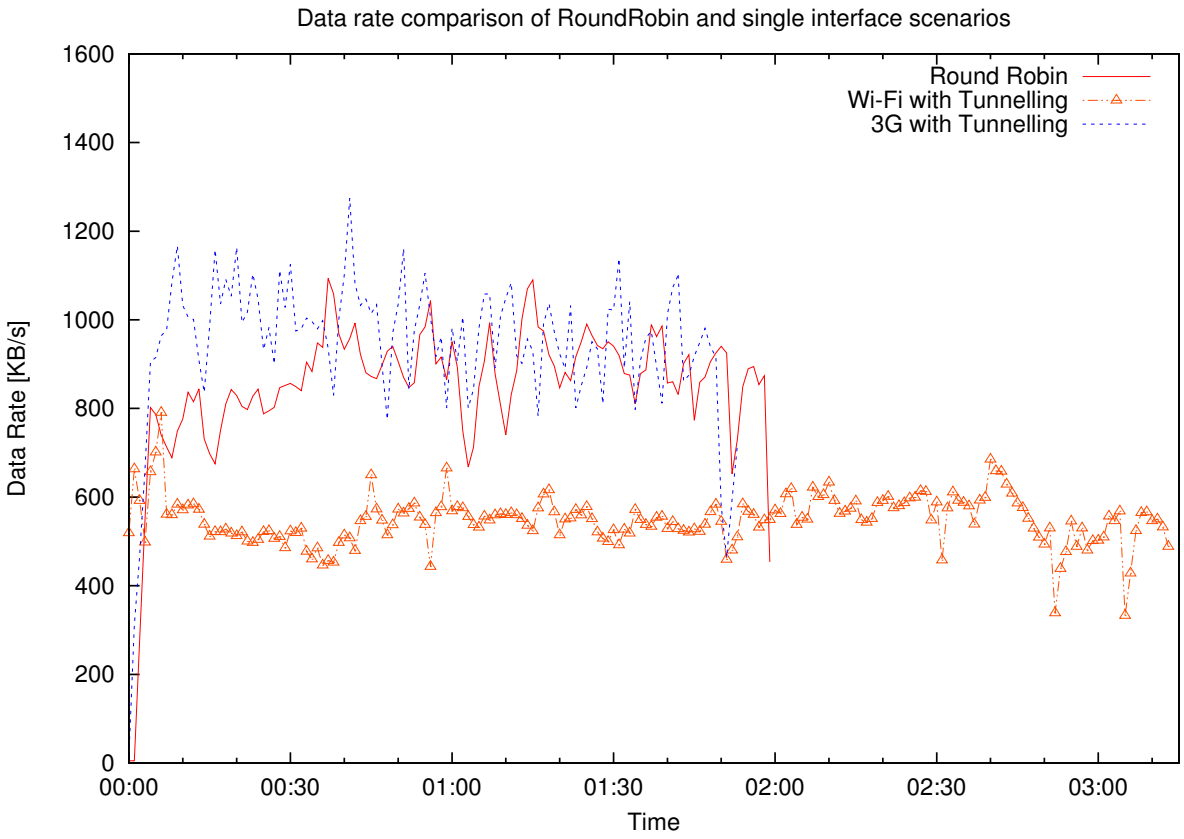



Figure 5.16: Data rate comparison of round-robin and single interface scenarios

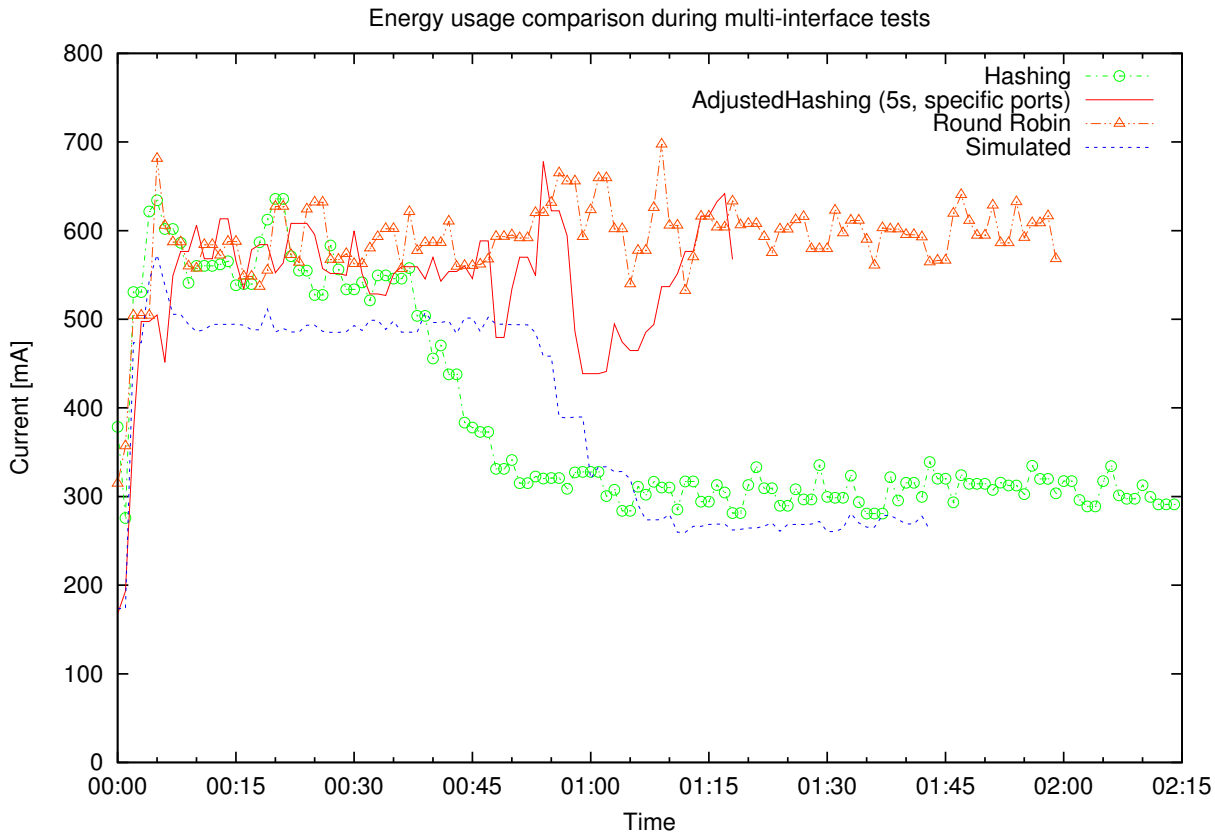



Figure 5.17: Comparison of energy usage during different multi-interface scenarios

the aggregated data rates achieved during the experiments. Whenever given scheduler was able to use both networks, it was using significantly more energy than when it was using only one of them. This is true, to some extent, even during short periods of time when the hash weight adjustments were performed.

The scheduler that resulted in the highest energy usage was the round robin algorithm. It was simply using both interfaces during the entire experiment. To further show problems related to using this scheduler, we include Figure 5.18 with comparison of energy usage of that scheduler, as well as using only a single interface at a time. Using different interfaces results in different energy usage levels, but since the round robin scheduler uses both networks, its energy consumption is significantly higher. Given that round robin scheduler failed to provide increased data rates (higher than those offered by the 3G network alone), simple packet scheduling is clearly not a solution to be used in practice.

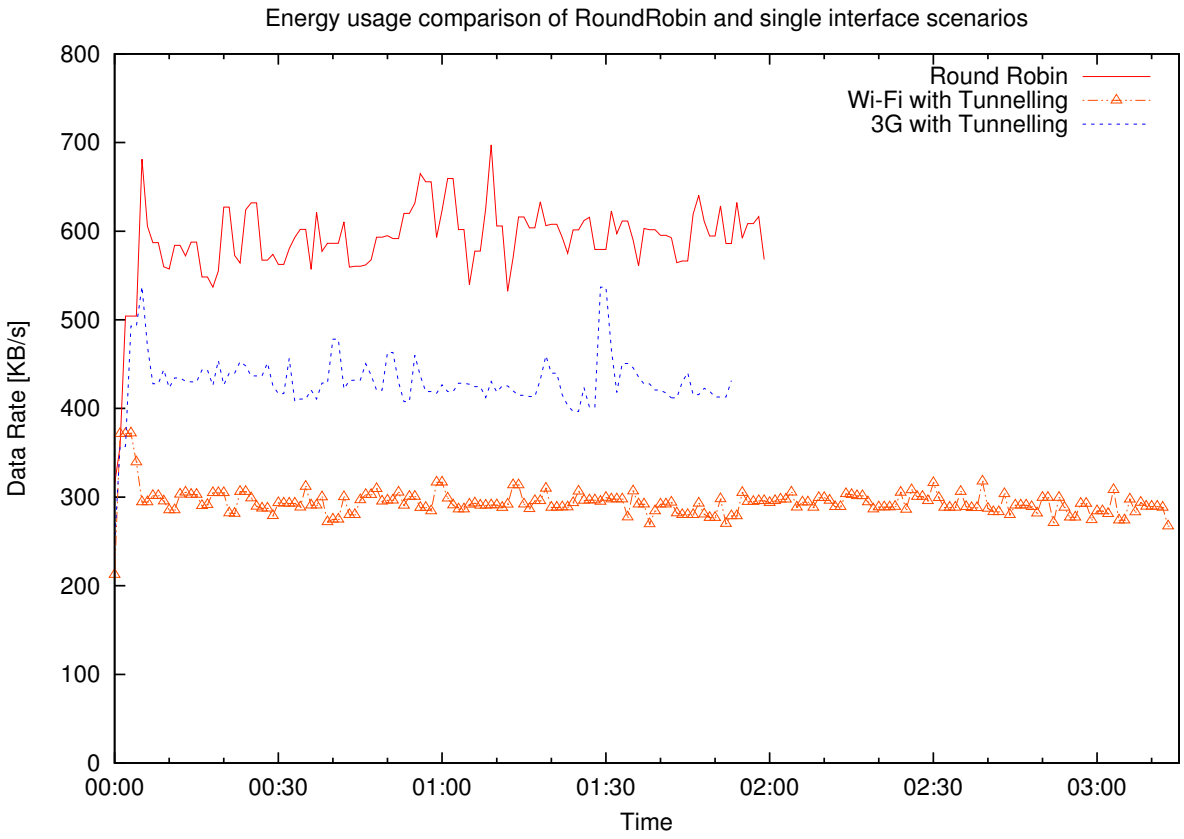



Figure 5.18: Energy rate comparison of round-robin and single interface scenarios

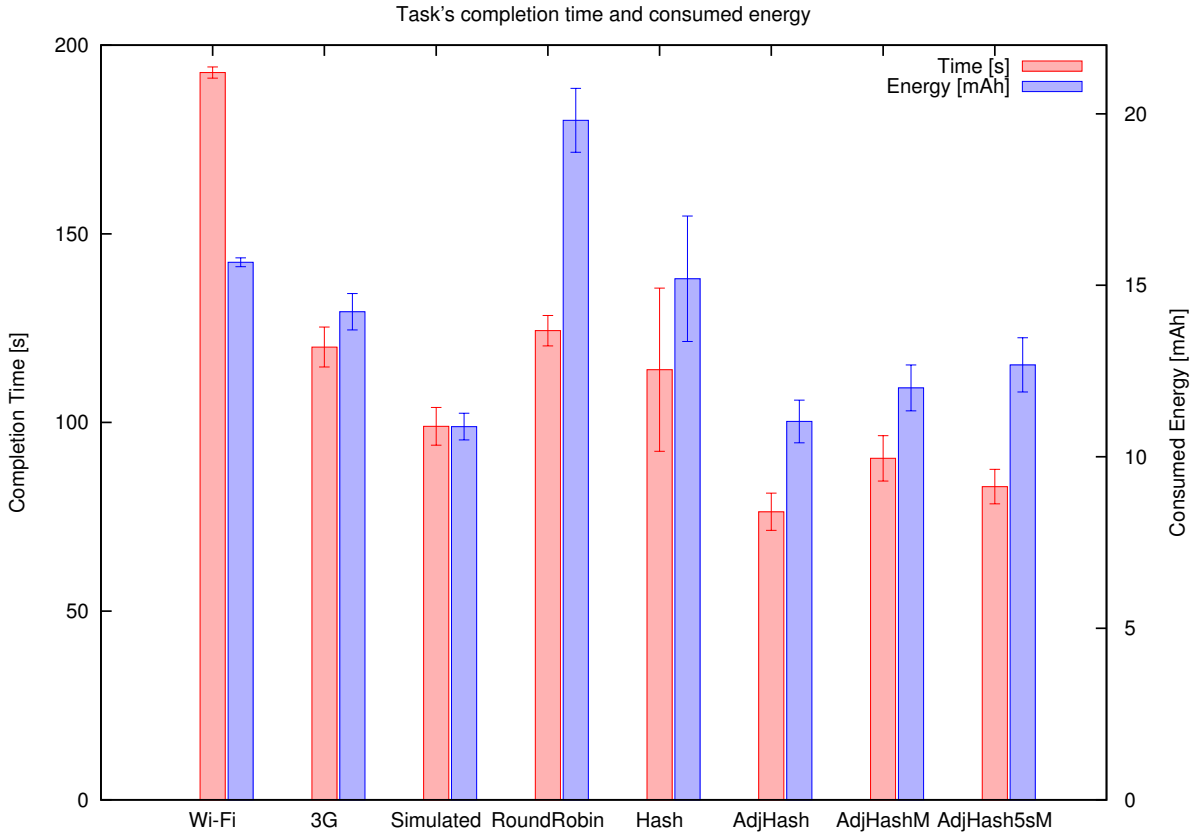



Figure 5.19: Completion time and consumed energy in different single and multi-interface scenarios

### Average results

The comparisons so far operated on selected traces collected during individual experiments. The results from several experiments performed for each scenario were aggregated to allow for comparison of larger data sets. The results from single interface, "tunnelled", scenarios are included as well.

There are several observations that can be made based on the results presented so far - both the individual experiment traces and the aggregated, average results from the larger number of simulations.

First observation is that using multiple interfaces simultaneously is more expensive, in terms of per second energy usage, than using only a single interface at a time. Also, the

Wi-Fi interface has significantly lower energy requirements than the 3G interface.

However, even though per-second energy usage is always higher in multi-interface case, the per-task energy usage is, in most cases, lower than the cost of completing the same task using only a single interface. The reason is the shorter time to complete the task, and shorter time the mobile device stays in the active state.

The time and data rate results, and differences between different scenarios depend heavily on the actual network conditions. Different data rates available over Wi-Fi and 3G networks would affect the actual differences between specific scenarios. However, the assumptions behind configuration of the test bed used in this work are realistic and should be applicable in many cases. Given these network conditions, it is possible to improve network parameters offered to the applications running on the client device.

The first of the multi-interface approaches, the "simulated" solution, offers, on average, significantly shorter task completion times. It also results in the lower per-task energy cost. The main problem with that solution is that it is not applicable in practice. It requires careful device configuration for specific, anticipated traffic. Another problem exposed during the experiments is that it cannot adapt to changing traffic conditions.

The next solution, the per-packet, round robin scheduler in theory could offer bandwidth aggregation even when only one data flow is used. However, results show that not only it does not improve task completion times, but also results in extremely high per-task energy cost - significantly higher than any other single or multi-interface solution considered. For these reasons, this scheduler cannot be considered as a viable solution for practical deployments.

The aggregated results in Figure 5.19 do not suggest that the unmodified, simple hashing scheduler is a viable solution either. However, the standard error of the related measurements is the biggest of all the scenarios examined. This is because the performance of this scheduler depends significantly on the traffic parameters. Some of the test runs performed completed in comparable, or even lower times than even those of the simulated solution. However, the cases when flows were not properly assigned to interfaces resulted in significantly longer completion times. Both these cases were considered together, causing both the time and energy consumption average to grow.

The modified version of the hashing scheduler, that performs weight adjustments at standard intervals of 20-30 seconds (labelled in Figure 5.19 as "AdjHash") performed much better. It not only was able to complete the test task significantly faster than any other solution, including the simulated scenario, but also used relatively low amount of energy - second best after the simulated scenario.

The next scenario presented in Figure 5.19 (labelled as "AdjHashM") represents the "worst case" traffic characteristics. This is the hashing scheduler with weight adjustments at standard, 20-30 second intervals, but with source port numbers selected to skew the flow assignment. Even in these carefully crafted, sub-optimal conditions, the scheduler was able to achieve both time and energy results better than in any of the single interface scenarios. In real life conditions, the behaviour in most cases would be better than in this "worst case" conditions.

The last of the schedulers examined is labelled as "AdjHash5sM". This is the hashing scheduler running in "worst case" traffic conditions, but with the adjustments happening at shorter intervals - every 5 seconds. The completion times achieved in this case are slightly above the "regular" hashing scenario. This is expected, as the traffic used during these experiments were configured to cause problems. Energy usage is greater than in case of other adjusted hashing scenarios. This is related to increased frequency of calculations and operations related to adjusting hash weights. But even this increased level of energy usage remains below the per-task energy used during any of the single interface scenarios.

The scheduler of choice in real life deployment would be the adjusted hashing scheduler, with adjustments performed every 20-30 seconds. Both per-task energy usage and completion times remain low, even in the worst case traffic conditions.

On the other side of the scale, the round robin scheduler should not be used in multi-interface environment, as it does not offer improved completion times, and yet uses significantly more energy than any other solution examined.

# Chapter 6

# Conclusion

The multi-interface system presented in this work has numerous interesting properties. It offers data continuity and flow migration between different physical networks. It is a practical solution, which can be deployed in real-life network without requiring any changes in the existing network infrastructure or the remote servers used by network applications used today. It also does not require significant changes in the client's operating system, or the applications running on the client device. The only requirement is installing the software package providing this service.

Results obtained during experiments with real network show, that using simple packet based scheduling does not improve the networking parameters when multiple networks are used in parallel. It is possible that more advanced, future methods may yield better results, but the straightforward solution does not offer satisfying results. Even though time reduction between Wi-Fi only (completion time: ~187 s) and multi-interface experiments (completion time: ~124 s) was significant, it performed worse than 3G only attempt (completion time: ~115 s). It also resulted in a very significant increase in per-task energy usage (~20 mAh compared to ~14 mAh in case of Wi-Fi and ~12 mAh when only the 3G interface was used).

On the other hand, per-flow scheduling can often significantly improve bandwidth properties offered to the applications running on the client device, whenever more than a single data flow is used. The hashing scheduler with weight adjustments proves to be an effective way to provide increased total bandwidth to the system, and is able to adapt to changing traffic conditions, even in specific, sub-optimal situations. In the typical scenario our algorithm was able to reduce test task completion time from ~187 s (Wi-Fi) or ~115 s (3G) in single interface scenarios to ~76 s in multi-interface case.

Our results show that, as expected, using multiple network interfaces at the same time uses more energy per second. However, they also show that the energy cost of performing specific tasks in single interface scenarios can be reduced. The average completion time was, on average, equal to ~14 mAh for Wi-Fi, and ~12 mAh for 3G scenarios. Using our scheduler with both hashing and weight adjustment algorithms reduced this energy cost to, on average, ~11 mAh. This may not be a big difference but it shows that instead of generating additional energy costs for the benefit of decreased data transmission times, using multiple interfaces and the right scheduling algorithm can actually save energy.

## 6.1 Future work

There are several directions in which this work can be continued and extended. First, the selection of the schedulers available in the system could be extended with algorithms that perform better traffic handling while switching networks. Instead of focusing on improved data rates, they could monitor network conditions and proactively control traffic to provide seamless hand-overs of data flows between different networks.

Another area of future work is flow fairness. The schedulers presented in this work ignore this problem and focus on the total, aggregated bandwidth. If the flow fairness is desired, additional algorithms should be monitoring the traffic and perform additional adjustments to try to provide better fairness between the flows.

Finally, the area of per-packet bandwidth aggregation should be further explored. Results presented in this work show that simple, naive approach does not yield satisfying results. However, it may be possible to build more advanced packet-based scheduler that would be able to offer bandwidth aggregation even in single flow scenarios. One of the ideas is to build upon the idea of the "Indirect TCP" [8]. That mechanism could potentially be used on both sides of the multi-interface part of the network path between the client application and the remote server. This idea, referred to as a "Double Indirect TCP", could make avoiding reordering and delaying TCP packets possible. These issues, introduced by the simple packet-based scheduler, are the reason of degraded performance of TCP flows. The "Double Indirect TCP" approach could be the base for constructing a practical scheduler offering bandwidth aggregation even for a single data flow.

# References


[1] A. Abogharaf, R. Palit, K. Naik, and A. Singh. A methodology for energy performance testing of smartphone applications. In *7th International Workshop on Automation of Software Test - AST 2012*, pages 110–116, June 2012.

[2] H. Adiseshu, G. Parulkar, and G. Varghese. A reliable and scalable striping protocol. *Applications, Technologies, Architectures, and Protocols for Computer Communication*, 26(4):131, 1996.

[3] C. Ahlund and A. Zaslavsky. Multihoming with mobile IP. *6th IEEE International Conference on High Speed Networks and Multimedia Communications*, pages 235–243, 2003.

[4] M. Allman, H. Kruse, and S. Ostermann. An Application-Level Solution to TCP's Satellite Inefficiencies, 1997.

[5] K. Argyraki, S. Baset, B.G. Chun, K. Fall, G. Iannaccone, A. Knies, E. Kohler, M. Manesh, S. Nedevschi, and S. Ratnasamy. Can software routers scale? *Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 21–26, 2008.

[6] A. Argyriou and V. Madisetti. Bandwidth aggregation with SCTP. *IEEE Globecom 2003*, 2003.

[7] R. Arya, R. Palit, and K. Naik. A methodology for selecting experiments to measure energy costs in smartphones. In *2011 7th International Wireless Communications and Mobile Computing Conference*, pages 2087–2092. IEEE, July 2011.

[8] A. Bakre and B.R. Badrinath. I-TCP: indirect TCP for mobile hosts. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, ICDCS '95, page 136, Washington, DC, USA, 1995. IEEE Computer Society.

[9] H. Balakrishnan, H.S. Rahul, and S. Seshan. An integrated congestion management architecture for internet hosts. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 175–187, New York, NY, USA, 1999. ACM.

[10] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference - IMC '09*, page 280, New York, New York, USA, November 2009. ACM Press.

[11] A. Baldini, L. De Carli, and F. Risso. Increasing Performances of TCP Data Transfers Through Multiple Parallel Connections. *IEEE Symposium on Computers and Communications, 2009*, 2009.

[12] J.C.R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking (TON)*, 7(6), 1999.

[13] R.W. Bickhart, P.D. Amer, and R.R. Stewart. Transparent TCP-to-SCTP Translation Shim Layer, 2005.

[14] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM SIGCOMM Computer Communication Review*, 32(1), 2002.

[15] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), 1970.

[16] C. Casetti, C.F. Chiasserini, R. Fracchia, and M. Meo. AISLE: Autonomic Interface SeLEction for Wireless Users. *International Workshop on Wireless Mobile Multimedia*, 2006.

[17] C. Casetti and W. Gaiotto. Westwood SCTP: load balancing over multipaths using bandwidth-aware source scheduling. *IEEE 60th Vehicular Technology Conference*, pages 3025 – 3029, 2004.

[18] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt. Flow Aggregation for Enhanced TCP over Wide-Area Wireless. In *INFOCOM 2003*, pages 1754–1764, 2003.

[19] K. Chebrolu, B. Raman, and R.R. Rao. A Network Layer Approach to Enable TCP Over Multiple Interfaces. *Wireless Networks*, 11(5):637–650, September 2005.

[20] K. Chebrolu and R.R. Rao. Communication Using Multiple Wireless Interfaces. In *WCNC 2002: Wireless Communications and Networking Conference*, volume 1, pages 327–331. IEEE, 2002.

[21] K. Claffy, G. Miller, and K. Thompson. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone. In *International Networking Conference (INET) '98*, 1998.

[22] M. Dobrescu, N. Egi, K. Argyraki, B.G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: exploiting parallelism to scale software routers. *ACM Symposium on Operating Systems Principles*, pages 15–28, 2009.

[23] F.R. Dogar, P. Steenkiste, and K. Papagiannaki. *Catnap: Exploiting High Bandwidth Wireless Interfaces to Save Energy for Mobile Devices*. ACM Press, New York, New York, USA, June 2010.

[24] K. Evensen, D. Kaspar, P. Engelstad, and A.F. Hansen. A Network-Layer Proxy for Bandwidth Aggregation and Reduction of IP Packet Reordering. *The 34th Annual IEEE Conference on Local Computer Networks*, pages 585 – 592, 2009.

[25] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proceedings of the 8th international conference on Mobile systems, applications, and services - MobiSys '10*, page 179, New York, New York, USA, June 2010. ACM Press.

[26] Internet Engineering Task Force. Tunneling of SCTP over Single UDP Port. `http://www.ietf.org/proceedings/48/I-D/sigtran-sctptunnel-00.txt`.

[27] Link Aggregation Task Force. IEEE 802.3ax (IEEE P802.1AX) Link Aggregation. `http://ieee802.org/3/axay/`.

[28] WiMAX Forum. WiMAX FAQ. `http://www.wimaxforum.org/resources/frequently-asked-questions`.

[29] R. Fracchia, C. Casetti, C.F. Chiasserini, and M. Meo. WiSE: Best-Path Selection in Wireless Multihoming Environments. *IEEE Transactions on Mobile Computing*, 6(10), 2007.

[30] L.A. Grieco and S. Mascolo. End-to-End Bandwidth Estimation for Congestion Control in Packet Networks. In *Quality of Service in Multiservice IP Networks: Second International Workshop, QoS-IP 2003, Milano, Italy, February 24-26, 2003. Proceedings*, pages 645–658, 2003.

[31] D.A. Hayes, J. But, and G. Armitage. Issues with network address translation for SCTP. *ACM SIGCOMM Computer Communication Review*, 39(1), 2008.

[32] H.Y. Hsieh and R. Sivakumar. pTCP: An end-to-end transport layer protocol for striped connections. In *Proceedings of IEEE ICNP*, 2002.

[33] C. Huang and Z. Zeng. SCTP-Based Bandwidth Aggregation across Heterogeneous Networks. *Computational Intelligence and Industrial Application*, 1:650–654, 2008.

[34] J.R. Iyengar, P.D. Amer, and R. Stewart. Retransmission Policies for Concurrent Multipath Transfer Using SCTP Multihoming. *ICON 2004*, 2004.

[35] J.R. Iyengar, P.D. Amer, and R. Stewart. Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths. *IEEE/ACM Transactions on Networking*, 14(5):951 – 964, 2006.

[36] J.R. Iyengar, K.C. Shah, and P.D. Amer. Concurrent Multipath Transfer Using SCTP Multihoming, 2004.

[37] V. Jacobson. Compressing TCP/IP Headers. `http://tools.ietf.org/html/rfc1144`.

[38] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. In *Proceedings of the 2nd annual international conference on Mobile computing and networking*, number 3775 in Request for Comments, pages 27–37. IETF, 2004.

[39] L. Kalampoukas, A. Varma, and K.K. Ramakrishnan. Explicit window adaptation: A method to enhance TCP performance. In *Proceedings of IEEE INFOCOM'98*, pages 242–251, 1998.

[40] S. Keshav. A control-theoretic approach to flow control. *ACM SIGCOMM Computer Communication Review*, 25(1):188, 1995.

[41] S.P. Kumar and M. Golash. Efficient flow-aware dynamic link load balancing. In *Proceedings of the First international conference on COMmunication Systems And NETworks*, COMSNETS'09, pages 77–82, Piscataway, NJ, USA, 2009. IEEE Press.

[42] K. Lee, I. Rhee, J. Lee, S. Chong, and Y. Yi. Mobile data offloading. In *Proceedings of the 6th International COnference on - Co-NEXT '10*, page 1, New York, New York, USA, November 2010. ACM Press.

[43] J. Liao, J. Wang, and X. Zhu. cmpSCTP: An extension of SCTP to support concurrent multi-path transfer. *IEEE International Conference on Communications - ICC '08*, 2008.

[44] Y.H. Lin, S.L. Tsao, Y.L. Cheng, and C.M. Yu. Dynamic Bandwidth Aggregation for a Mobile Device with Multiple Interfaces. In *ISCOM 2005: International Symposium on Communications*, Taiwan, 2005.

[45] L. Magalhaes and R. Kravets. Transport Level Mechanisms for Bandwidth Aggregation on Mobile Hosts. *Urbana*, 2008.

[46] B.S. Manoj, R. Mishra, and R.R. Rao. SEBAG: A New Dynamic End-to-End Connection Management Scheme for Multihomed Mobile Hosts. *Distributed Computing IWDC 2005*, 3741:524–535, 2005.

[47] S. Mascolo, C. Casetti, M. Gerla, M.Y. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. *International Conference on Mobile Computing and Networking*, 2001.

[48] E. Oliver and S. Keshav. Data driven smartphone energy level prediction, 2010.

[49] R. Palit, R. Arya, K. Naik, and A. Singh. Selection and execution of user level test cases for energy cost evaluation of smartphones. In *Proceeding of the 6th international workshop on Automation of software test - AST '11*, page 84, New York, New York, USA, May 2011. ACM Press.

[50] C. Perkins. IP Mobility Support for IPv4 (RFC 3220), 2002.

[51] C.E. Perkins. Mobile IP. *IEEE Communications Magazine*, 35(5):84–99, May 1997.

[52] D.S. Phatak and T. Goff. A novel mechanism for data streaming across multiple ip links for improving throughput and reliability in mobile environments. In *INFOCOM*, 2002.

[53] D.S. Phatak, T. Goff, and J. Plusquellic. IP-in-IP tunneling to enable the simultaneous use of multiple IP interfaces for network level connection striping. *Computer Networks*, 2003.

[54] M.R. Ra, J. Paek, A.B. Sharma, R. Govindan, M.H. Krieger, and M.J. Neely. *Energy-delay tradeoffs in smartphone applications*. ACM Press, New York, New York, USA, June 2010.

[55] D. Rubenstein, J. Kurose, and D. Towsley. Detecting shared congestion of flows via end-to-end measurement. *IEEE/ACM Transactions on Networking (TON)*, 10(3), 2002.

[56] A.A.E.A.T. Saadawi and M. Lee. LS-SCTP: a bandwidth aggregation technique for stream control transmission protocol. *Computer Communications*, 2004.

[57] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V.N. Padmanabhan. Bartendr: A Practical Approach to Energy-aware Cellular Data Scheduling. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking - MobiCom '10*, page 85, New York, New York, USA, September 2010. ACM Press.

[58] A. Sharma, V. Navda, R. Ramjee, V.N. Padmanabhan, and E.M. Belding. Cool-Tether: Energy Efficient On-the-fly WiFi Hot-spots using Mobile Phones. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies - CoNEXT '09*, page 109, New York, New York, USA, December 2009. ACM Press.

[59] J. Shi, Y. Jin, W. Guo, and S. Cheng. Performance Evaluation of SCTP as a Transport Layer Solution for Wireless Multi-access Networks. *Wireless Communications and Networking Conference*, pages 453 – 458, 2004.

[60] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti. The PPP Multilink Protocol (MP) (RFC 1990), 1996.

[61] A.C. Snoeren. Adaptive inverse multiplexing for wide-area wireless networks. *Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99. (Cat. No.99CH37042)*, pages 1665–1672, 1999.

[62] 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), 2000.

[63] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, and M. Kozuka. SCTP Dynamic Address Reconfiguration (RFC 5061), 2007.

[64] N. Thompson, G. He, and H. Luo. Flow scheduling for end-host multihoming. *Proceedings of IEEE INFOCOM*, 2006.

[65] O. Titz. Why TCP Over TCP Is A Bad Idea.
`http://sites.inka.de/bigred/devel/tcp-tcp.html`.

[66] C.L. Tsao and R. Sivakumar. On effectively exploiting multiple wireless interfaces in mobile hosts. *5th ACM International Conference on emerging Networking EXperiments and Technologies*, pages 337–348, 2009.

[67] N. Vallina-Rodriguez, P. Hui, J. Crowcroft, and A. Rice. Exhausting battery statistics: understanding the energy demands on mobile handsets. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, MobiHeld '10, pages 9–14, New York, NY, USA, 2010. ACM.

[68] L. Wang and J. Manner. Energy Consumption Analysis of WLAN, 2G and 3G interfaces. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 300–307. IEEE, December 2010.

[69] Smart Wi-Fi. UMA Overview.
`http://www.smart-wi-fi.com/overview.php`.

[70] F. Xie, N. Jiang, Y.H. Ho, and K.A. Hua. Semi-split tcp: Maintaining end-to-end semantics for split tcp. In *Proceedings of the 32nd IEEE Conference on Local Computer Networks*, LCN '07, pages 303–314, Washington, DC, USA, 2007. IEEE Computer Society.

[71] L. Zhang, S. Shenker, and D.D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. *SIGCOMM Comput. Commun. Rev.*, 21(4):133–147, 1991.