Master's Programme in Computer Network Engineering, 60 credits

BANDWIDTH AGGREGATION OF MOBILE BROADBAND LINKS ON RASPBERRY PI BASED ACCESS POINT

Lukas Chrast, Jozef Knaperek, Marek Kovalcik

Computer Network Engineering, 15 credits

Halmstad 2014-06-01
BANDWIDTH AGGREGATION OF MOBILE BROADBAND LINKS ON RASPBERRY PI BASED ACCESS POINT

Master's Thesis in Computer Network Engineering

Lukas Chrast
Jozef Knaperek
Marek Kovalcik

Supervisor: Urban Bilstrup

May 2014
Bandwidth aggregation of mobile broadband links on Raspberry Pi based access point
Lukas Chrast, Jozef Knaperek, Marek Kovalcik

Master thesis report IDE1411
School of Information Science, Computer and Electrical Engineering
Halmstad University
Abstract

This thesis is concerned with the usability of Raspberry Pi as the access point in the mobile broadband network environment. The first part of the thesis is dedicated to Raspberry Pi itself; hardware required to set up WLAN and WAN; and to the analysis of suitable solutions for bandwidth aggregation, particularly the load balancing of mobile broadband connections and their aggregation into one logical link. The second part deals with the implementation of these solutions and subsequently with their testing and verification. The evaluation of results gives an interesting outcome. Load balancing has proven to be resilient and feasible solution for bandwidth aggregation in the mobile broadband network environment where the speed, packet loss and jitter are of main concern. The second scenario, where the connections are bundled into one logical link, has turned out to give variable results. Its performance is susceptible to the changes in the mobile broadband network as the packets across the links in the bundle alternate in the round-robin fashion.

Keywords: WiFi; Linux; Raspberry Pi; Point-to-Point Protocol; load balancing; iptables
Acknowledgement

We would like to thank our supervisor Urban Bilstrup and examiner Tony Larsson for their useful suggestions and advices. Also, we would like to acknowledge Halmstad University for providing resources needed to accomplish this thesis.
Table of Contents

1 Introduction ........................................................................................................................................... 1

2 Review of Key Technologies ................................................................................................................. 2
  2.1 Introducing Raspberry Pi .................................................................................................................. 2
    2.1.1 Platform Specifications ............................................................................................................. 2
    2.1.2 Powering ................................................................................................................................. 2
  2.2 Accessing WLAN ............................................................................................................................... 3
    2.2.1 802.11 Operation Modes ........................................................................................................ 3
    2.2.2 Distinct Roles in Infrastructure Mode ...................................................................................... 4
    2.2.3 Performance vs. Flexibility ...................................................................................................... 4
    2.2.4 Entering Linux Network Stack ............................................................................................... 5
    2.2.5 Identifying Compatible Devices ............................................................................................. 5
  2.3 Software AP ....................................................................................................................................... 6
    2.3.1 WLAN Security ....................................................................................................................... 6
  2.4 Mobile Broadband Modems .............................................................................................................. 7
    2.4.1 Modems with Integrated Storage .......................................................................................... 7
  2.5 Load Balancing Across Multiple WAN Connections ......................................................................... 7
    2.5.1 Linux Netfilter/Iptables .......................................................................................................... 8
    2.5.2 Iptables Operation .................................................................................................................. 8
    2.5.3 Iproute2 .................................................................................................................................. 14
  2.6 Link Aggregation ............................................................................................................................... 15
    2.6.1 Point-to-Point Link ................................................................................................................ 16
    2.6.2 Point-to-Point Protocol .......................................................................................................... 16

3 Implementation ..................................................................................................................................... 22
  3.1 Initializing the Hardware .................................................................................................................... 22
    3.1.1 WiFi Adapter Detection in Linux ............................................................................................. 22
    3.1.2 Broadband Modem Detection in Linux .................................................................................. 23
  3.2 Configuring Software ........................................................................................................................ 25
    3.2.1 Installing OS ........................................................................................................................... 25
    3.2.2 WiFi Access Point Configuration .......................................................................................... 25
    3.2.3 DHCP & DNS Setup ............................................................................................................... 26
    3.2.4 Broadband Modem Setup ....................................................................................................... 27
    3.2.5 Daemonizing with Supervisor ............................................................................................... 28
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>Load Balancing Configuration</td>
<td>29</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Setting Up Routes and Routing Tables</td>
<td>29</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Marking Packets</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>Mobile Broadband Aggregation</td>
<td>31</td>
</tr>
<tr>
<td>3.4.1</td>
<td>A Remote Server</td>
<td>32</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Setting the Point-to-Point Environment</td>
<td>32</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Setting the Multilink PPP</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>Testing and Evaluation</td>
<td>38</td>
</tr>
<tr>
<td>4.1</td>
<td>Methodology</td>
<td>38</td>
</tr>
<tr>
<td>4.1.1</td>
<td>File Download via HTTP</td>
<td>38</td>
</tr>
<tr>
<td>4.1.2</td>
<td>BitTorrent Download</td>
<td>38</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Measuring Tools</td>
<td>39</td>
</tr>
<tr>
<td>4.2</td>
<td>Testing Environment and Appliance</td>
<td>39</td>
</tr>
<tr>
<td>4.3</td>
<td>Measurement Results</td>
<td>39</td>
</tr>
<tr>
<td>4.4</td>
<td>Observations</td>
<td>41</td>
</tr>
<tr>
<td>4.4.1</td>
<td>BitTorrent vs. HTTP</td>
<td>41</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Multilink Tunnel vs. Direct Access</td>
<td>41</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Load Balancing Performance</td>
<td>42</td>
</tr>
<tr>
<td>4.4.4</td>
<td>System Utilization</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>References</td>
<td>44</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1: Packetflow in Netfilter [7]........................................................................................................9
Figure 2: PPP Encapsulation......................................................................................................................17
Figure 3: PPP Phase Diagram....................................................................................................................18
Figure 4: LCP Encapsulation.....................................................................................................................19
Figure 5: Multilink Encapsulation..............................................................................................................20
Figure 6: Network topology in load balancing configuration.....................................................................29
Figure 7: Network topology with link aggregation..................................................................................32
List of Tables

Table 1: Built-in chains in Iptables [8] ........................................................................................................ 10
Table 2: Tables in Iptables [6] ...................................................................................................................... 11
Table 3: List of Iptables commands [6] ......................................................................................................... 13
Table 4: Parameters for Iptables command [6] ............................................................................................ 14
Table 5: Results using file download via HTTP method, in kB/s .................................................................... 40
Table 6: Results using BitTorrent method, in kB/s ..................................................................................... 40
List of Abbreviations

ACFC – Address and Control Field Compression
AP – Access Point
APCP – AppleTalk Control Protocol
cfg80211 – The new Linux kernel API for controlling WNIC
HDLC – High-Level Data Link Control
IPCP – Internet Protocol Control Protocol
IPXCP – Internetwork Packet Exchange Control Protocol
KVM – Kernel-based Virtual Machine
LCP – Link Control Protocol
MRU – Maximum Receive Unit
NAT – Network Address Translation
NCP – Network Control Protocol
nl80211 – The new Linux user space API for universal management of WLAN
PBR – Policy based routing
PPP – Point-to-Point Protocol
RPDB – Routing policy database
RPi (or the Pi) – Raspberry Pi
VPS – Virtual private server
WAN – Wide area network
WEXT (or WE) – Wireless Extensions – It is the old Linux driver API for communicating with WNIC. It has been obsoleted by cfg80211.
WLAN – Wireless local area network
WNIC – Wireless Network Interface Controller – Implements an interface to wireless network. Operates on both Layer 1 and Layer 2 of the OSI model.
1 Introduction

Several months have passed since the first Raspberry Pi model came out on the market. Its size and low price has awoken a huge interest in computer community and several thousand units were ordered on daily basis. As of today, over two million units have reached their owners, individuals as well as educational institutions; yet only a few researches in computer networks have been made with Raspberry Pi as its integral part. So far they have focused on different technologies. For example tracking of people at mass events using Wifi\(^1\) or Bluetooth based applications\(^2\). Some projects used Raspberry Pi as 3G wireless routers\(^3\), but none of them dealt with bandwidth aggregation and they did not set their goal to determine the hardware limitations of Raspberry Pi.

In this thesis, we will examine Raspberry Pi as a wireless access point to a mobile broadband network. One part of the research will aim to find out how suitable Raspberry Pi is as a network device in terms of traffic handling and providing essential network services. The other part will be dedicated to mobile broadband bandwidth aggregation. We will investigate scalability of the AP and examine the ways of accommodating more users connected simultaneously. Two methods will be evaluated: load balancing and link aggregation. The former achieves bandwidth aggregation through load spreading across multiple mobile broadband connections, whereas the latter aggregates these connections into one logical link. As both of these methods use more than one physical connection, they also provide a redundancy that is nowadays highly desirable. We will observe and measure the performance of the system and identify its limits. If Raspberry Pi proves its feasibility for such network operations, it could be deployed in a mobile environment such as public transportation, meetings and conferences. It could even help to provide higher bandwidth connectivity in locations where network connection other than mobile broadband is unavailable or economically infeasible.

The document is divided into five chapters. Chapter 2 covers the analysis of Raspberry Pi, WiFi adapters, mobile broadband modems, and software needed for their operation. Next, solutions for load balancing and link aggregation are covered in detail. Chapter 3 describes the implementation of these solutions. Subsequently, Chapter 4 describes testing methodology and presents results. Finally, Chapter 5 provides the conclusion where results from the previous chapter as well as overall evaluation were taken in to account.

\(^1\) WiFiPi: Involuntary tracking of visitors at mass events
(http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6583443)

\(^2\) Bluetooth communication using a touchscreen interface with the Raspberry Pi
(http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6567448)

\(^3\) Raspberry Pi as a 3g (Huawei E303) wireless (Edimax EW-7811Un) router
(http://bigcowpi.blogspot.de/2013/03/raspberry-pi-as-3g-huawei-e303-wireless.html)
2 Review of Key Technologies

2.1 Introducing Raspberry Pi

Despite being relatively new, RPi is already an established embedded Linux platform, well-known for its credit card size and low cost. Affordability of this single-board computer and maturity of the supported operating system make it suitable for a wide range of use cases. This project aims to evaluate the performance and overall suitability of Raspberry Pi as a mobile wireless AP.

2.1.1 Platform Specifications

The core of the device is Broadcom BCM2835 system on a chip (SoC), featuring single core 700 MHz ARM1176JZF-S processor and 512 MB of main memory. Older versions were equipped with only 128 or 256 MB of main memory, so we have picked the latest “B” model (with 512 MB) for this project. Officially supported operating system is Raspbian but other alternative systems supported by the community or third parties are available as well. Since Raspbian is a modified version of Debian GNU/Linux, it benefits from its broad software availability and cross-platform support. Vast majority of Debian packages can be installed and run on Raspbian without any modification in their source code, as ARM architecture is (among others) officially supported by Debian.

There is no hard disk or flash memory built-in on the Pi. The only persistent storage is an external SDHC card, partitioned and used for both firmware and the operating system with custom software and data.

External devices can be connected via standard High Speed USB (2.0) interface. There is also wired one Fast Ethernet interface, internally connected to the USB bus. Video output is available through HDMI or RCA / composite ports.

2.1.2 Powering

A dedicated micro USB port with no data pins is used for powering the device. Power supply with standard voltage (5V) capable of providing at least 700mA on the output is recommended, but the total power consumption tightly depends on connected peripherals as well. It is generally safer to use a powered USB hub when connecting devices with the total power consumption over 150mA. Because of sensitive circuitry in RPi, susceptible to fluctuations in the input voltage, it is essential to use high-quality power supply for RPi itself and USB hub that does not “backfeed” the power through the upstream port [1]. Using inappropriate accessories can result in malfunction or brick the device entirely.
2.2 Accessing WLAN

Connecting mobile users to the local RPi based gateway is the first step towards providing Internet access. LAN technology chosen for this purpose must be well supported on all targeted devices, such as laptops and smartphones, and must provide sufficient flexibility with respect to mobility. For obvious reasons and widespread use of wireless technology standardized in the IEEE 802.11, commonly known as "WiFi", this was chosen as the key WLAN technology for the RPi based Mobile AP. It is however possible to extend the support with another WLAN technology and still transparently leverage the rest of the system, without the need of further changes in downstream components.

2.2.1 802.11 Operation Modes

The 802.11 standard recognizes two modes in which a wireless station can operate:

- Ad hoc,
- Infrastructure.

Ah hoc mode is for its simplicity intended to be used for temporary connections between two or more stations. There is no central station controlling the network or providing additional services like DHCP or DNS. All transmitted frames are flowing directly between the communicating stations. This mode is not suitable for the purpose of Mobile AP and thus will not be a subject of further analysis in this document.

The Infrastructure mode is predominantly used in wireless networks managed by a dedicated central station usually serving also as a gateway to other networks, such as campus network or the Internet. The role of the central station – Access Point – is to control the channel and authenticate users requesting network access. It can enable advanced usage of WiFi network, such as roaming, and often integrates various additional features and services on multiple OSI layers (e.g. routing, DHCP, DNS, firewall, NAT).

Having these modes standardized enables many different devices to communicate regardless of their manufacturer or type. Hence the idea of using an ordinary consumer WiFi adapter and turning it into a full-featured Access Point seems technically viable. Other stations should then be able to connect to it the same way they connect to hardware Access Points manufactured specifically for this purpose.

In reality, though, this is not that straightforward and often impossible or unpractical with most of the commercially available WiFi adapters. In order to better understand the reasons causing
the issues and to get the necessary insight in the technology, a brief explanation of some Wireless Network Interface Controller (WNIC) internals is beneficial.

2.2.2 Distinct Roles in Infrastructure Mode

The major pitfall comes from the difference in operation of a station in the Infrastructure mode depending on its role. Access Point working in the Infrastructure mode has different responsibilities and it needs to perform different operations compared to a client station operating in the Infrastructure mode. Therefore it is only logical to have several distinct implementations for particular features related to the Infrastructure mode rather than a monolithic all-in-one solution. On the downside, customary USB WiFi dongles designed to serve as client stations often support only necessary subset of features and don't target to have a full implementation of the standard.

Common feature sets implemented in 802.11 network adapters are divided into a number of operating modes. The most common are:

- Ad-Hoc mode (also known as IBSS),
- Station Infrastructure mode (also known as Managed),
- Access Point Infrastructure mode,
- Monitor mode,
- Wireless Distribution System (WDS),
- Mesh. [2]

At one time, a network interface is only able to operate in one of the modes listed and this must be supported by both the hardware and the software (traditionally implemented as device drivers in common operating systems).

While Station Infrastructure mode is the default mode and thus well supported by all the devices and their respective drivers, AP Infrastructure mode is for its complexity often neglected and not supported by the device manufacturer. Unlike PCI based WNICs, usually capable of working in both, client and AP modes, USB adapters with this support are extremely rare to find.

2.2.3 Performance vs. Flexibility

Wireless cards are complex devices with lots of different feature sets implemented at various levels. While hardware implementations have the advantage of generally better performance, software implementations provide better control and customization. It's worth noting that different platforms have different requirements and while it is more suitable to let WNIC hardware manage most of the network related work when used in embedded systems, it might
not be as advantageous on modern PC architectures. The reason behind this is obvious: embedded devices usually functioning as a single-purpose appliance are more likely to benefit from off-loading their scarce CPU resources, whilst powerful PC systems might prefer to endure some additional IO overhead and get more flexible access in return. Entering the software network stack at lower levels gives more possibilities to control the behavior on software side and enables various software overrides. Different use cases for the same hardware make both chipset and driver development more complex and are one of the reasons behind several incompatibilities in implementations.

As AP mode is not considered a standard use of WiFi USB dongle and requires more specific control depending on the use (e.g. encryption, authentication and client management), it needs to be implemented partially in software and use proper interface to delegate only the lower level functions, such as packet generation and media access, to the hardware chipset.

### 2.2.4 Entering Linux Network Stack

In Linux, this has traditionally been accomplished using Wireless Extensions (WEXT) interface in the kernel. There was not much functionality implemented in the kernel itself so every driver was responsible for implementing everything by itself, which led to fragmentation and upstream incompatibility. Thus it was not possible to create a single application that could use some advanced features with every WNIC, even though it technically had hardware support for it. Some manufacturers even used to supply their own forks of common user-space software in order to leverage advanced features available through their WEXT drivers.

To make things better, new standard interfaces were introduced in the kernel. Common wireless features can now be set using cfg80211 kernel interface and manufacturers are encouraged to make their drivers compliant with this new interface. This in turn enables user-space software, such as AP implementations, to use the standard nl80211 (netlink) interface to manage wireless interfaces, regardless of their manufacturer or model. With the new netlink interface, much more flexible setups are possible and one can even create virtual interfaces operating in different modes simultaneously. The most important feature in the context of this project is certainly the unified interface to AP Infrastructure mode.

### 2.2.5 Identifying Compatible Devices

Despite the huge progress made with the standardization of WNIC interfaces in the Linux network stack, there's still a lot of devices with only the obsolete WEXT drivers available. It is therefore important to carefully choose a USB dongle and make sure the AP mode is supported in hardware and that it is compatible with the current Linux driver stack (nl80211, cfg80211).
No less important is the stability of the device when operating in a nonstandard mode, such as AP. Crucial thing to identify on an USB WiFi dongle is the chipset manufacturer and model number, as this solely determines the supported features and required drivers, regardless of the dongle manufacturer or model.

In the context of this document, devices fully compliant with the new netlink interface are assumed to be used in the setup. Simple guidance for identifying and distinguishing compatible devices in Linux will be provided in the chapter 3.1.1 WiFi Adapter Detection in Linux.

2.3 Software AP

Leveraging modern WiFi adapters and their netlink compatible drivers, it is possible to create a full featured WiFi Access Point with just a user-space application. Hostapd is well-known open source daemon with comprehensive implementation of 802.11 AP features, making it a great fit for this project's purpose. It takes the responsibility for the following operation:

- broadcasting beacon frames,
- client association,
- client authentication (according to 802.11i),
- encryption keys setup and rotation.

For better performance and the ability to connect many mobile stations to the AP, we have decided to use a set of three identical WiFi dongles to operate on non-overlapping channels (1, 6, and 11). They are all managed by the same hostapd instance and are internally connected to a virtual LAN bridge (implemented in the kernel). DHCP and DNS service are provided by dnsmasq daemon attached to the virtual bridge. This setup makes all the wireless clients belong to the same IP network, regardless of the WiFi adapter they are connected to. Local communication between individual clients thus flows only through the virtual bridge without the need of IP routing. Packets destined for remote machines connected via the Internet exit the bridge and are further routed according to active configuration setup, described in chapters 3.3 Load Balancing Configuration and 3.4 Mobile Broadband Aggregation.

2.3.1 WLAN Security

Hostapd supports all common state-of-the art security measures for hardening WLAN networks. It implements IEEE 802.1X/WPA/WPA2 authenticator and EAP server as well as RADIUS functionality suitable for enterprise deployments [3]. There is also a comprehensive support for security schemes commonly deployed in SOHO setups, such as WPA-PSK authentication with TKIP encryption mode or WPA2-PSK with CCMP.
2.4 Mobile Broadband Modems

In order to provide locally connected clients with the Internet access, it first needs to be delivered to the RPi itself. Since the whole appliance needs to be mobile, wired connections can't be considered a viable solution. We have decided to adopt commercially available mobile networks with broadband Internet access. Combining multiple Internet connections is essential in order to provide sufficient bandwidth for many local clients connected at the same time.

Although RPi itself has no built-in modem for mobile broadband networks, external modems can be easily connected to the USB bus. These are available in various designs with internal or external antennas and contain a SIM card for enabling communication with a provider's network.

Unlike WiFi network adapters these are much easier to use with Linux as no uncommon usage is necessary in this setup. Communication with the modem is conveyed using a sequence of AT commands, in a fashion similar to dialing legacy ISDN modems. After a circuit is created at the physical level, Point-to-Point Protocol negotiates the link and IP address leased from ISP is assigned to the client. From this point on, it is registered in the kernel as an ordinary network interface which can be used to route traffic out.

2.4.1 Modems with Integrated Storage

It is common that most of the USB broadband modems come with an integrated block storage device, typically used to hold filesystem with drivers for MS Windows operating system and for arbitrary user data as well. While this may be a useful feature, it sometimes prevents the modem device from being automatically registered in the Linux kernel. This can be fixed by manually configuring the device ID that should be registered when the USB dongle is plugged in. The proper ID value differs amid device models, but can be looked up in various databases. An example will be provided in chapter 3.1.2 Broadband Modem Detection in Linux.

2.5 Load Balancing Across Multiple WAN Connections

When the throughput of a single network connection is not sufficient, usually there are two options; either we can change the connection for a faster one, or we can add more connections to meet the needs we have. The former is usually an expensive option, thus the latter often comes out in favor.

There are two possibilities how this can be done. The first and simpler option is to load balance traffic from hosts connected to the Raspberry Pi across multiple broadband links. When there is
a higher throughput requirement for a single connection than a single broadband link can provide, links need to be aggregated. The chapter 2.6 Link Aggregation will discuss this setup.

Load balancing can be done per connection, but some applications may require per host load balancing, i.e., all communication from one host is routed through one wide area network (WAN) link. We can use Iptables together with Iproute2 software package to achieve this.

2.5.1 Linux Netfilter/Iptables
Iptables and Netfilter are used together for a packet filtering, network address translation (NAT), packet mangling and manipulation, application layer filtering and more [4]. Netfilter and Iptables are replacement for ipchains (Linux kernel 2.2.x) and ipfwadm (Linux kernel 2.0.x) with added functionality.

Netfilter is a Linux kernel framework providing a set of hooks for kernel modules, which allows them to register functions with the network stack at various stages as a packet traverses these hooks. It does not filter the traffic by itself. Netfilter is present in Linux kernel from version 2.4.

Iptables comprise of kernel modules and user-space binaries. Kernel modules use the hooks provided by Netfilter. These modules provide a table-based system for maintaining the rules for the packet filtering and manipulation. Examples of the main modules are ip_tables, ip6_tables, arp_tables, ebtables, including modules responsible for specific tasks, such as iptables_filter, iptables_nat and iptables_mangle. User-space binaries iptables, ip6tables, arptables and ebtables are then used for administering tables maintained by the kernel modules. [5]

Term “Iptables” will be used in this document to refer to the user-space tools, kernel modules and the whole Netfilter/Iptables framework interchangeably, depending on the context.

2.5.2 Iptables Operation
Iptables provide mechanism for maintaining packet filtering and manipulating rules in Linux kernel. These rules are added to a specified table and to a specified chain. Each rule can have different options, which are divided into commands and parameters. Commands specify action to be performed and parameters specify protocols, interfaces, matches, targets etc. [6]

Following is an example of the iptables command structure:

```
iptables -t <table> -A <CHAIN> -i <interface> -p <protocol> -m <match> -j <TARGET>
```

Figure 1 provides an overview of all possible packet paths through IP stack and all built-in chains in Iptables/Netfilter.
Packet flow in Netfilter and General Networking

Figure 1: Packetflow in Netfilter [7]
2.5.2.1 Chains

Iptables provide five built-in chains (Table 1), which are present by default and provide hooks to Netfilter. The source of the packet determines which chain is hit initially.

Table 1: Built-in chains in Iptables [8]

<table>
<thead>
<tr>
<th>Chain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREROUTING</td>
<td>Packets enter this chain after being received on an interface and before the routing decision is made.</td>
</tr>
<tr>
<td>INPUT</td>
<td>Packets enter this chain after the routing decision and before being delivered to local process.</td>
</tr>
<tr>
<td>FORWARD</td>
<td>After the routing decision is made and packets are not designated for a local delivery. Packets enter this chain when traversing the system from one interface to another.</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Packets enter this chain after being created by a local process and before the routing decision.</td>
</tr>
<tr>
<td>POSTROUTING</td>
<td>Packets enter this chain after the routing decision, before they leave a network interface.</td>
</tr>
</tbody>
</table>

Chain is a list of rules, which are examined sequentially when a packet enters a chain. When the rule matches the packet, action specified in its target is executed. When no rule matches the packet and the packet reaches the end of the chain, a default policy is applied. The default policy for all built-in chains is to accept the packet. This can be changed with the \(-P\) command.

In addition to the built-in chains, administrator can define an arbitrary number of user-defined chains. These can be used as targets to other rules within the same table. User-defined chains can be nested without a limit and their default policy is to return to the calling chain. This default policy cannot be changed, but a rule can be added at the end of the chain to serve the same purpose.

2.5.2.2 Tables

Iptables provide five independent tables and each holds rules to serve a different functionality. Several chains are associated with each table as shown in Table 2.
### Table 2: Tables in Iptables [6]

<table>
<thead>
<tr>
<th>Table</th>
<th>Associated chains</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>INPUT FORWARD OUTPUT</td>
<td>This is the default table (when no other table is specified using <code>-t</code> option). Defines which packets are allowed to enter, leave or traverse the system.</td>
</tr>
<tr>
<td>nat</td>
<td>PREROUTING OUTPUT POSTROUTING</td>
<td>This table is mainly used for NAT. Rules in the nat table chains are evaluated only once for the first packet of a new connection. Rest of the packets within the same connection will have the same action taken on them as the first packet of the connection.</td>
</tr>
<tr>
<td>mangle</td>
<td>INPUT FORWARD OUTPUT POSTROUTING</td>
<td>Table mangle is used for a packet alternation. Header information of packets or a kernel-space marking can be changed here.</td>
</tr>
<tr>
<td>raw</td>
<td>PREROUTING OUTPUT</td>
<td>This table is used to exempt packets/connections from being tracked or for debugging purposes.</td>
</tr>
<tr>
<td>security</td>
<td>INPUT FORWARD OUTPUT</td>
<td>Table security is used for MAC networking rules.</td>
</tr>
</tbody>
</table>

#### 2.5.2.3 Packet flow

Packets traverse specific chains, depending on the source and the destination. A packet that hits a chain is presented to its rules sequentially. If a rule does not match the packet, the packet moves to the next rule in the chain. If the packet does not match any of the rules, the default chain policy is applied.

Based on the source of the packet and the routing decision, packet can take four different paths through Iptables:

- from one network interface to another (forwarding)
- from local process to a network interface (output)
- from a network interface to a local process (input)
- from a local process to a local process (local)

For the purpose of this thesis forwarding and output paths are interesting. Following are chronologically ordered table – chain pairs traversed when packet takes one of these paths.
Forwarding:
- raw – PREROUTING
- mangle – PREROUTING
- nat – PREROUTING
- mangle – FORWARD
- filter – FORWARD
- mangle – POSTROUTING
- nat – POSTROUTING

Output:
- raw – OUTPUT
- mangle – OUTPUT
- nat – OUTPUT
- filter – OUTPUT
- mangle – POSTROUTING
- nat – POSTROUTING

2.5.2.4 Rules

Each Iptables rule has a defined position within a specific Iptables chain. An Iptables rule consists of one or more matches and a target which determines fate of the packet if the packet is matched. Each Iptables rule has a packet and a byte counter. When the packet is matched by the rule the packet counter is incremented by one and the byte counter is incremented by the size of the packet.

Matches

Rules can be matched by the source and the destination address, input and output interfaces and the protocol.

When option `-m` is specified on the command line, extended modules for matching can be used. This option is followed by the name of an extended matching module and module options. Following example shows the use of the module statistic, which is used to match packets based on a statistic condition:

```
-m statistic --mode random --probability 0.5
```

Targets

There are three types of targets in an Iptables rule:

- built-in targets
- user defined chains
- extended target modules

Built in targets are ACCEPT, DROP, QUEUE and RETURN. ACCEPT lets the packet through a specific chain while DROP drops the packet. QUEUE target passes the packet to the user space. RETURN target in user-defined chain stops traversing the current chain and continues with the
calling chain. In a built-in chain, RETURN target applies the default chain policy, as if the packet reached the end of the chain.

When a user-defined chain is used as a target, traversing current chain is stopped and packed is matched against the rules in a specified user-defined chain.

Extended target modules can be used to specify additional actions on packets. Example of extended target modules are MARK/CONNMARK, which manages Netfilter marks with connections, and MASQUERADE, which is used for NAT. Extended targets can usually have an additional options specifying actions. Connection tracking using MARK/CONNMARK targets is especially useful with the per connection load balancing.

2.5.2.5 Options

Options for iptables command are divided into 2 groups, commands and parameters. Option -t selects a specific table of Iptables. If this option is not provided, the default table (filter) is assumed.

Commands

Commands specify an action to be performed upon a rule and are followed by chain. Only one command can be specified on the command line. Some of the commands are listed in Table 3.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A chain rule-specification</td>
<td>Append command is used to append a rule to the end of the specified chain.</td>
</tr>
<tr>
<td>-D chain {rule-specification</td>
<td>Deletes a rule or rules from the specified chain. A rule can be specified as a rule to match or as a rule number, where the first rule is numbered as 1.</td>
</tr>
<tr>
<td></td>
<td>rulenum}</td>
</tr>
<tr>
<td>-L [chain]</td>
<td>Lists rules in the chain. If no chain is specified, rules in all the chains in a specified table are listed.</td>
</tr>
<tr>
<td>-F [chain]</td>
<td>Flushes all rules in the chain. If no chain is specified, rules in all the chains in a specified table are flushed.</td>
</tr>
<tr>
<td>-P chain target</td>
<td>Changes default policy of the chain to the specified target. Only the built-in targets can be specified.</td>
</tr>
</tbody>
</table>

Parameters

Rule specification consists of parameters. Table 4 lists the most important parameters. An exclamation mark ("!") can be used before some parameter flags to invert their sense.
**Table 4: Parameters for Iptables command [6]**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ] -p protocol</td>
<td>Protocol name or number to match. Protocol names from /etc/protocols and following protocol names can be used: tcp, udp, udplite, icmp, esp, ah, sctp. Special keyword “all” or number 0 is used to match all protocols.</td>
</tr>
<tr>
<td>[ ] -s address[/mask][,...]</td>
<td>Specifies the source of a packet. Address can be a network name, hostname, network IP address or host IP address. For network IP address, mask is specified either as a network mask or a prefix length.</td>
</tr>
<tr>
<td>[ ] -d address[/mask][,...]</td>
<td>Specifies the destination of a packet. The syntax is same as for the -s parameter.</td>
</tr>
<tr>
<td>-m match</td>
<td>Specifies the match to use. Match is extension module. In order for a rule to match a packet, all matches must be evaluated as true.</td>
</tr>
<tr>
<td>-j target</td>
<td>Jump parameter specifies the target of the rule.</td>
</tr>
<tr>
<td>[ ] -i name</td>
<td>Specifies the name of the interface on which the packet was received. A special wildcard “+” can be used at the end of the name to match all interfaces that start with the name.</td>
</tr>
<tr>
<td>[ ] -o name</td>
<td>Specifies the interface out of which the packet is going to be sent. Wildcard “+” can be used the same way as with -i parameter.</td>
</tr>
</tbody>
</table>

### 2.5.3 Iproute2

Iproute2 is a software package that provides utilities for an advanced routing, tunnels and traffic control under Linux OS. It is intended to substitute standard net-tools and unify their syntax. [4]

Iproute2 consists of a several commands, out of which ip and tc are the most important. ip is used for managing routing, network devices, policy based routing (PBR) and tunnels. tc command is used for a traffic control.

Ip utility has following structure [9]:

```plaintext
ip [ OPTIONS ] OBJECT { COMMAND | help }
OBJECT := { link | addr | route | rule | neigh | tunnel | maddr | mroute | monitor }
```

### 2.5.3.1 Ip Command Objects

**Link**

Link object is used for listing and changing an interface configuration and properties.
**Address**

Address object serves to show, add and delete the IP addresses on a specified interface.

**Route**

Route object is used to manipulate routes in kernel routing tables. From Linux kernel 2.2 multiple routing tables can be used. These are specified in the file `/etc/iproute2/rt_tables`. Except from the main and the local table, administrator can define up to 252 additional routing tables. Route object has commands to add, change, replace, delete, show, flush, and get the routes.

**Rule**

Rule object is used for PBR. This allows to route packets based on the source IP address, IP protocol, ports and payload in addition to the destination IP address. It can also match packets based on fwmark, previously set with Iptables.

To enable PBR the conventional destination based routing table is replaced with the Routing policy database (RPDB). Rules in the RPDB are ordered based on their priority and consist of a selector and an action predicate. If the selector matches the packet the action is performed. If the action is successful a route is selected or the packet is dropped, depending on the action, otherwise, PBR continues with the next lower priority rule. Additionally each rule is pointing to some routing table.

### 2.6 Link Aggregation

As opposed to load balancing, when the throughput requirement for a single connection exceeds bandwidth of a single network link, network links can be aggregated.

Two or more network connections aggregated together form a bundle. Depending on an implementation, there can be additional connections in standby mode that provide means for failover. When one of the aggregated connections fails, the bundle can still operate, although with a lower throughput when no failover is available. This provides redundancy that is nowadays highly desirable. Link aggregation can be implemented at any of the lowest three layers of the OSI model: at the physical layer, at the data-link layer, in both LAN and WAN environment, and at the network layer.

For the purposes of our master thesis, we want to aggregate the bandwidth of 3G modems attached to the Raspberry Pi to find out how suitable it is to bundle mobile broadband connections. We have decided to use an extension of the original Point-to-Point Protocol (PPP), Multilink PPP. As the name of the protocol indicates, it is designed to establish a direct
connection between two points over various types of physical networks such as serial link, phone line, cellular network and fiber optic link. The Internet is no such network, and due to that, we have to set an environment that resembles a point-to-point network. In the next subsections, the tools for setting the environment as well as PPP protocol are discussed.

2.6.1 Point-to-Point Link

2.6.1.1 Socat

Socat, or Socket CAT, is a command line based network utility that establishes two bidirectional byte streams and transfers data between them. These streams do not have to be of the same address type. Among others, there are various address types such as raw byte stream, pipe, socket (IPv4, IPv6, TCP, UDP, UNIX), SSL or file. Each of these address types has numerous options to tune them precisely for our needs. Thanks to the variety of address types and their respective options, usage of Socat is multipurpose. It can be used for TCP port forwarding, IPv6 relay, redirection of various streams, and for security testing and research of various network solutions.

Because of Multilink PPP, we need to set up point-to-point connections, serial links particularly. Socat’s address type “raw” can be conveniently used for serial link emulation. Start, stop, parity and data bits can be set via address type’s options. By this we can create a couple of serial interfaces on the local, as well as on the remote point. To bridge these two points, we will use another network utility called Netcat which is discussed in the next subsection.

2.6.1.2 Netcat

Netcat is another useful utility used for various network operations involving TCP, UDP, or UNIX-domain sockets. It can listen on UDP and TCP ports, open TCP connections, send UDP packets, do port scanning, and much more. The capability of reading and writing data across network connections using TCP/UDP is the missing part we need. We have decided to make TCP connections via Netcat to relay data over Internet between the local and remote point. Some may wonder why to use another tool as Socat provides means to redirect a virtual serial interface to TCP/UDP socket; however, we have found out that Socat’s TCP/UDP socket was causing a bottleneck in communication for an unknown reason; therefore we have decided to use Netcat as an alternative.

2.6.2 Point-to-Point Protocol

The Point-to-Point Protocol (PPP) [10] is designed for point-to-point links that transport packets from one point of a link to the remote one and vice versa. These links are assumed to provide full-duplex simultaneous bi-directional operation and to deliver packets in order. PPP is
designed to easily interconnect a wide variety of hosts, routers and switches. It provides a standard method for transporting multi-protocol datagrams over point-to-point links. PPP can be considered rather as a set of protocols then a single protocol. It consists of three main components: encapsulation, Link Control Protocol (LCP), and Network Control Protocol (NCP). Moreover, a number of compression, encryption, and authentication protocols can be employed to provide means for high speed implementations and security issues.

### 2.6.2.1 Encapsulation

The PPP encapsulation enables multiplexing of different network-layer protocols simultaneously over the same link. By default, 8 octets are necessary to form the encapsulation. In the case of environments where bandwidth is scarce, the encapsulation can be shortened to 2 or 4 octets. Simple fields were designed within the encapsulation to support high speed implementations where merely one field needs to be checked for demultiplexing. In the Figure 2 we can see the framing format for PPP. It is based on the High-Level Data Link Control (HDLC) protocol which was well-established by the time when PPP was designed.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Addr</th>
<th>Control</th>
<th>Protocol</th>
<th>Information</th>
<th>Padding</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7E</td>
<td>0xFF</td>
<td>0x03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x7E</td>
</tr>
</tbody>
</table>

**Figure 2: PPP Encapsulation**

With HDLC-like framing used, the PPP frame is surrounded by two 1-octet flags called Flag. It is used by both sides of a point-to-point link to find the start and end of the frame.

After the front Flag field, PPP adopts two HDLC fields. In HDLC, the Address field is used to specify the recipient. As PPP is concerned only with a single destination, the fixed value of 0xFF, which means “all stations”, is always used.

The Control field in HDLC is used to specify frame sequencing and frame behavior. PPP implementation is not commonly enhanced by these functions and the Control field is set to the constant value of 0x03. As the Address and Control field use a constant value, they are often omitted by an option called Address and Control Field Compression (ACFC).

The Protocol field is 2-octet field which contains information about data encapsulated in the Information field of the PPP frame. Except for LCP-carrying frames, the Protocol field can be compressed to 1-octet. Protocol field values identify the network layer protocol of the encapsulated packets, data belonging to an associated NCP, low volume traffic which has no associated NCP, or control protocols such as LCP.
The Information field carries data for the protocol specified in the Protocol field. It has from zero to Maximum Receive Unit (MRU) octets including Padding and excluding Protocol field. If not specifically stated, MRU defaults to 1500 octets.

The Padding field can be utilized to pad PPP frame with non-zero values [11]. Some encryption and compression algorithms need minimum number of octets, a block size, when operating. In such case, padding is used to provide this minimum number of octets or its multiples.

The FCS field stays for Frame Check Sequence and defaults to a 16-bit Cyclic Redundancy Check error detection code. It covers the entire PPP frame except for the Flag fields and the FCS field itself. It can be extended to 32 bits using the same CRC32 polynomial that is used for Ethernet.

2.6.2.2 PPP Link Operation

Link operation starts with an initial exchange of LCP packets to configure and test the data link. If successful, the link is established and authentication may proceed. Then, PPP must send NCP packets to configure one or more network layer protocols. Once the particular network layer has been configured, datagrams from that protocol can be sent over the link. The link for particular network layer protocol will remain up until NCP packets of this protocol close the link down, or the link as a whole will be terminated by LCP packets. Additionally, the link can be terminated when idle timer expires or by network administrator intervention. The following phase diagram in Figure 3 illustrates the basic state transitions of PPP protocol.

![Figure 3: PPP Phase Diagram](image-url)
2.6.2.3 LCP

LCP protocol provides a means to establish a low-level two-party communication over a point-to-point link. As PPP's operation is concerned only with two ends of the point-to-point link, there is no need to handle the problem of access to shared resources like in the case of MAC-layer protocols of Ethernet and WiFi. The communication starts by an initial exchange of messages with a proposal of options to be configured. Unlike general PPP packets carrying data for upper network-layer protocols, LCP uses three additional fields: Code, Ident and Length field, as illustrated in Figure 4 below.

![Figure 4: LCP Encapsulation](image)

In the case of LCP, the Protocol field is always set to the value of 0xC021 and cannot be compressed. It means it has always 2 octets.

The Ident field is used by a sender for packet identification. The value of this field is set by the sender of a request and exactly the same value is used by the replying peer. In this way, the PPP peer can assign each reply to the right request. The Ident field is incremented one by one as subsequent packets are sent by a sender.

The Code field identifies either requesting or responding operation. The LCP operation always starts with Configure-Request sent by one side of a point-to-point link. The peer from the other side may reply with Configure-Ack if it accepts the requested options. If some of the values in the request are not acceptable, the Configure-Nak reply is sent with unacceptable options specified. If some option is neither recognizable nor available for negotiation, Configure-Reject is sent as a reply. Terminate-Request and Terminate-Ack are used to close a connection. Code-Reject is replied to the request with an unknown Code field value. Protocol-Reject is used in similar fashion when evaluating the Protocol field. The famous Echo-Request and Echo-Reply is used by LCP too. A Discard-Request packet serves for debugging, performance testing and numerous other functions.

The Length field contains the number of octets in an LCP packet and indicates the end of configuration options listed in the LCP Data field. A typical option is Maximum Receive Unit (MRU) which indicates the maximum size of packets that are acceptable by the peer. Magic-
Number is used for loop prevention. The other examples of configuration options are data required by authentication and compression protocols. The Multilink option, which is essential for our thesis, is discussed later.

### 2.6.2.4 NCPs

Once LCP establishes a link and its optional authentication, both ends of the point-to-point link reach the Network state (depicted in Figure 3) and they can start to negotiate a network-layer association using zero or more NCP protocols. There are many NCP protocols that can be run atop PPP, even simultaneously. Most of them, such as AppleTalk Control Protocol (APCP) and Internetwork Packet Exchange Control Protocol (IPXCP) are not used more than their respective network-layer counterparts. The most commonly used NCP is Internet Protocol Control Protocol (IPCP) [12]. The Protocol field of IPCP is set to 0x8021. IPCP uses the same packet format and exchange of requests, acknowledgements and rejects as LCP. Lot of options can be negotiated via IPCP, such as number of compression protocols and DNS information. It is interesting to note Van Jacobson header compression. The 40 bytes of TCP and IP headers can be compressed to as little as 4 bytes. Non-changing parts of these headers are stored at both sides of point-to-point link; in transmission they are replaced by 1-byte connection identifiers. Values that change as transmission proceeds, such as sequence numbers, are encoded in remaining 3 bytes.

### 2.6.2.5 Multilink PPP

Multilink PPP [13] is an extension of the original PPP protocol. Both PPP peers need to support this extension, which is indicated by an LCP option. It means that the system offering this option is capable of combining multiple physical links into one aggregated logical link. Moreover, the system is able to receive upper layer protocol data units; both fragmented PDUs (using the multilink header) and larger PDUs than MRU of a single physical link.

Once enabled and agreed upon, two or more links can be aggregated and act as one logical link, called bundle. The simplest possible solution would be to alternate PPP packets between actual physical links. However, this solution may lead to undesirable side effects in the case of packet reordering. The actual solution amends the original PPP encapsulation and introduces two additional fields shown in the Figure 5 below.

![Multilink Encapsulation Diagram](image)

*Figure 5: Multilink Encapsulation*
The Flags field contains two 1-bit flags, beginning fragment bit (B) and ending fragment bit (E). If a frame is not fragmented, then both bits are set to 1, indicating that the fragment is the first one and the last one and represents the whole frame. In all other cases, the first fragment has the BE bits set to 10 and the last fragment to 01. All the fragments in between have these bits set to 00. The subsequent bits between E bit and the Sequence Number field are not defined and must be set to zero. The Flags field is 4 bits long, when a short sequence number has been negotiated, otherwise it has 8 bits.

The Sequence Number field, used to address a packet reordering, is incremented for each fragment transmitted. It is either 24 bits long or reduced to 12 long when a short sequence number has been negotiated.

Multilink PPP introduces additional three LCP options; The Max-Receive-Reconstructed Unit option, or MRRU which specifies the maximum number of octets in the Information fields of reassembled packets; the Short Sequence Number Header Format option which is self-explanatory; and Endpoint Discriminator option, which identifies the system transmitting the packet, is used to decide whether the link will join an existing bundle or a new bundle will be established.

2.6.2.6 MP Operation

At first sight, load balancing across member links in a bundle might seem trivial; however, things can get tricky when links of different speeds are aggregated together. In theory, a possible way to address the load balancing across links of different transmission rates would be to divide the packets into segments proportional to the speeds of the links. Another option would be to divide segments into many equal fragments and distribute them across links in numbers proportional to the transmission rates. From the open-source solutions point of view, there are two options that can be deployed. With round-robin delivery, the bandwidth and latency are ignored and the PPP peer alternates the frames across all the links in the bundle. Another option, though more complex, is optimal delivery. In this case, each link's bandwidth, latency, and current queue length is taken into account. It is not appropriate for very slow machines and/or very fast links as the algorithm does require some computation. Both of these approaches are implemented for BSD operating systems; however, in the case of Linux, only round-robin fashion delivery is currently available.
3 Implementation

3.1 Initializing the Hardware

Our project setup requires a handful of peripherals to be used with RPi, significantly increasing the total power consumption of the system. Broadband USB modems are power-hungry devices able to utilize the whole 500mA output available from a standard USB 2.0 port, depending on a signal level and connection speed. WiFi USB adapters are less demanding in this regard: devices that we have used do not require more than 110mA each. With three 3G modems and three WiFi adapters the total power consumptions had quickly added up to 1.83A, which made using USB hub with an external power supply unavoidable. We have used 7-port backfeed-safe USB hub capable of providing 2A altogether, specifically designed for Raspberry Pi.

3.1.1 WiFi Adapter Detection in Linux

As already explained in the chapter 2.2.5 Identifying Compatible Devices, it is essential to have netlink compatible USB WNIC with AP support. After plugging the device in, the kernel should recognize the wireless chipset inside and load the proper network driver according to its identification. Running `lsusb` command shows if the chipset was recognized by the USB kernel subsystem. It is the first prerequisite; any valid USB device should pass this step.

```
(root@pi)# lsusb
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 010: ID 148f:5370 Ralink Technology, Corp. RT5370 Wireless Adapter
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 05e3:0608 Genesys Logic, Inc. USB-2.0 4-Port HUB
Bus 001 Device 005: ID 05e3:0608 Genesys Logic, Inc. USB-2.0 4-Port HUB
```

After the device has been detected in the system we may proceed with capability check. Supported 802.11 modes of any netlink compatible WNIC can be listed via `iw` command.\(^4\)

\(^4\) `iw` is the successor of legacy `iwconfig` utility, which used the old WEXT interface to communicate with WNIC drivers. Unlike `iw`, `iwconfig` does not work with the new `nl80211` (netlink) API and thus cannot be used for capability listing.
It is essential that AP mode be supported in order to use this device with hostapd. If it is not listed here, then either the device is not capable of running as soft AP or a compatible driver is missing in the operating system. We have tested several different devices that were not operational or required serious driver patching and were not even working reliable after it. The decision was then made to get adapters with Ralink RT5370 chipset, specifically designed for Raspberry Pi. Any adapters with netlink compatible Linux drivers capable of AP mode should be sufficient though.

### 3.1.2 Broadband Modem Detection in Linux

USB modems are often manufactured as integrated compound devices, which often prevents Linux from automatically detecting the modem, as we have mentioned in the chapter 2.4.1 Modems with Integrated Storage. In order to register the modem as a Linux device it is necessary to configure a proper product ID value to be selected when the dongle is plugged in. This can be achieved with `usb-modeswitch` [14] program that automatically switches the device to modem mode when it is detected by the USB subsystem.

First, `usb-modeswitch` needs to be installed as it is not a part of the standard Raspbian installation.

```
(root@pi)# apt-get install usb-modeswitch
```

Then, running `lsusb` should list the dongle plugged in once it is detected:
Important information highlighted in the `lsusb` output is device Vendor ID and Product ID (two numbers separated by a colon). The Product ID value refers to currently active mode (usb storage) which needs to be changed to modem mode. The proper value can be found in the archive file that comes with `usb-modeswitch` installation, located in `/usr/share/usb_modeswitch/configPack.tar.gz`. The archive contains text files with the same naming scheme as the highlighted identification in `lsusb` output above. Every text file lists all possible Product IDs for a particular device.

Using this information we create a new configuration file in `/etc/usb_modeswitch.d/` for each broadband modem model we want to use. An example configuration for ZTE MF102 model is provided below.

```plaintext
DefaultVendor=0x19d2
DefaultProduct=0x2000
TargetVendor=0x19d2
TargetProduct=0x0031
```

After the configuration file is created, dongle needs to be unplugged and then plugged in again for `usb-modeswitch` to take action. Running `lsusb` again should list the modem with the new Product ID number and possibly with a different description as well.

---

5 This identification can be further used to obtain an additional information about the device via `lsusb -v -d 19d2:2000`. 
If the dongle is successfully switched to modem mode, a device file named `gsmmodemX` should be created in `/dev`, where X is a sequential number of the device, distinguishing multiple modems connected simultaneously. Broadband modem is now ready to be used for the Internet connection.

### 3.2 Configuring Software

#### 3.2.1 Installing OS

Raspbian is the default operating system supported by Raspberry Pi manufacturers. It is being actively developed with updates provided through its own central repository. As this fits our purpose and there was no reason to use an alternative system, we have decided to install Raspbian in its most recent revision #557 which comes with Linux 3.6.11+

We have copied the downloaded image to a 4 GB SD card, plugged it in and proceeded with the installation. The whole process was straightforward and similar to standard Debian installation, with the exception that the SD card itself was used both as an installation medium and a target drive, since there is only one slot available in the Pi. This is however flawlessly handled by the prepared installation script and does not cause any issues.

#### 3.2.2 WiFi Access Point Configuration

Wireless Access Point advertises the network via “beacon” frames and provides an access for local clients requesting association with the advertised network. The most mature software implementation for Linux platform is certainly Hostapd. It is free software available as an optional package in the official repository and can be installed with the following command:
Hostapd works perfectly with modern WNICs, leveraging the netlink interface and implementing full-featured software Access Point on top of it. It runs as a daemon process managing a wireless interface configured as an AP. It is also possible to let a single instance of the daemon manage multiple WNICs simultaneously, passing a list of configuration files on the command line when starting the program:

```
(root@pi)# hostapd wlan0.conf wlan1.conf wlan2.conf
```

All WNICs are configured to use the same SSID advertising themselves as a part of the same wireless network, but are tuned to different non-overlapping channels (1, 6 and 11). Clients can choose which BSSID they want to associate with and thus use the least occupied channel. Connection is secured using WPA2-PSK authentication with AES encryption in CCMP mode, as it is currently considered the most secure scheme for non-enterprise networks. A sample `hostapd` configuration for a single WNIC is shown below.

```
interface=wlan0
bridge=lan-br
driver=nl80211
ssid=MobileAP
hw_mode=g
ieee80211n=1
channel=1
wpa=2
wpa_passphrase=secret123
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP
rsn_pairwise=CCMP
beacon_int=100
auth_algs=1
wmm_enabled=1
```

### 3.2.3 DHCP & DNS Setup

Interconnecting all WNICs into a single LAN through virtual bridge simplifies the topology and makes further processing easier. Regardless of BSSID a client chooses to connect to, it becomes a
part of the same IP subnet and receives an IP address from the single DHCP server running on the local network (virtual bridge). We have used dnsmasq daemon for its ability to integrate both DNS and DHCP services in a way that enables local client names resolution. A configuration example shows DHCP server listening on the virtual LAN bridge, leasing a range of RFC1918 private addresses for the period of one hour.

```
interface=lan-br
dhcp-range=172.17.2.100,172.17.2.150,255.255.255.0,1h
```

### 3.2.4 Broadband Modem Setup

Once a broadband modem is recognized in the kernel, it accepts AT commands for circuit creation and management. There is however no unified standard defining all the commands and order in which they should be sent for the circuit to be created. The set and usage of accepted commands differ among individual manufacturers and their various models. There is a free program called sakis3g [15] maintaining a comprehensive database of AT command sequences for a broad range of 3G modems. It is also capable of managing the modems and provides a rich text interface for selecting custom preferences. Unfortunately, sakis3g does not work with multiple modems connected at the same time so it is unsuitable for our purpose. We have however used its source code and extracted the AT command sequence for our Huawei and ZTE modems.

For broadband connection management we opted for a well-known open source utility called wvdial, which is faster and can operate multiple modems simultaneously. Not only is it capable of sending predefined AT commands to the modem, it also starts PPP daemon (pppd) right after the circuit with ISP is established. PPP then goes through standard LCP and NCP phases where the link parameters are negotiated and IP address is assigned to the client. Separate instance of wvdial needs to be launched for every modem.

Both pppd and wvdial are optional packages that can be added to the system from the official Raspbian repository:

```
(root@pi)# apt-get install ppp wvdial
```

When the packages are installed, we can create configuration profiles for all our modems. Creating a section in wvdial configuration file `/etc/wvdial.conf` we define AT command sequences to be used for initiating the connection on a particular modem. The following snippet
shows an example configuration for ZTE MF102 modem with Comviq SIM card. PIN code on the SIM card is disabled to avoid unnecessary issues.

```
[Dialer Defaults]
New PPPD = yes

[Dialer comviq1]
Init1 = ATZ
Init2 = ATQ0 V1 E1 SO=0 &C1 &D2 +FCLASS=0
Init3 = AT+CGDCONT=1,"IP","data.comviq.se"
Stupid Mode = 1
ISDN = 0
Phone = *99#
Modem = /dev/gsmmodem
Username = { }
Password = { }
```

### 3.2.5 Daemonizing with Supervisor

In the previous chapters we have described programs useful for providing network services needed in our setup. We have not mentioned the way they are run though and what their life cycle is. While some programs are predetermined to be used as daemons and thus are well integrated into the standard Debian init system, others do not have any standard way of auto-launching at startup, managing detached instances or rotating the logs.

Since standard Debian init scripts are too cumbersome to maintain (and will soon become obsolete) we have delegated this responsibility to a process control system called Supervisor [16]. Configuration is declarative and thus much simpler, daemonized processes can be managed via lightweight frontend program supervisorctl, logs are rotated automatically. It does even serve as watchdog, restarting crashed processes, which is particularly useful for embedded devices that need to operate for a long time without any administrator intervention or scheduled full reboot.

We have configured Supervisor to manage both wvdial and hostapd. It was not needed for dnsmasq as it is already a daemon by itself.

---

6 https://lists.debian.org/debian-devel-announce/2014/02/msg00005.html
3.3 Load Balancing Configuration

After Raspberry Pi is set up, we can focus on load balancing configuration. In our case we decided to perform per connection load balancing of incoming traffic from hosts on WLAN. This provides a better granularity and in our case also more accurate results.

The network topology can be seen in Figure 6. Hosts are connected wirelessly to WiFi adapters on Raspberry Pi. Routing is performed between the local LAN, represented by a virtual bridge, and the individual broadband interfaces.

![Network Topology](image)

**Figure 6: Network topology in load balancing configuration**

3.3.1 Setting Up Routes and Routing Tables

Before any traffic from hosts can be forwarded to the internet, IP forwarding must be enabled on Raspberry Pi with the following command:

```
(root@pi)# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Iproute2 package comes with most Linux distributions and Raspbian running on our Raspberry Pi is no exception.

In order to route traffic out all three WAN interfaces, we need to enable three additional routing tables. This is done by simply appending entries for these routing tables to `/etc/iproute2/rt_tables` file. The first field of the entry is arbitrary but unique number of the routing table and the second field is its name. This is not entirely necessary and we can use numbers (0-255) of the routing tables instead of their names. But for the sake of clarity it is a good practice to add these entries and address the routing tables by their descriptive names.
Each routing table needs to have a route to the local subnet (172.17.2.0/24), which resides on the virtual bridge interface lan-br. Once a packet is being routed using a custom routing table, routing decision is not affected by entries in the main routing table. This is important for returning packets.

And finally, each routing table has a single default route to one of the WAN interfaces:

3.3.2 Marking Packets

To achieve the load balancing, each new connection initiated by a host on WLAN is marked randomly with fwmark by Iptables. A different value of fwmark will result in the packet being routed using a different routing table and ultimately out a different interface.

First, Netfilter connection tracker checks whether the processed packet belongs to an existing connection. If so, the saved value of fwmark is assigned to this packet.

Otherwise, the series of Iptables rules assigns the value 1, 2 or 3 with $\frac{1}{3}$ probability and save that value for this connection.
Once we have our connections marked, we can route packets belonging to them using different routing tables, which are already set up:

```
(root@pi)# ip rule add fwmark 1 table T1
(root@pi)# ip rule add fwmark 2 table T2
(root@pi)# ip rule add fwmark 3 table T3
```

Last step is to configure NAT on each WAN interface, because we are using private IP addresses on our local network. We use MASQUERADE target, which translates a source address to the actual address on the interface, because IP addresses are assigned to interfaces by DHCP.

```
(root@pi)# iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
(root@pi)# iptables -t nat -A POSTROUTING -o ppp1 -j MASQUERADE
(root@pi)# iptables -t nat -A POSTROUTING -o ppp2 -j MASQUERADE
```

### 3.4 Mobile Broadband Aggregation

In the case of multilink, a bit different setup is needed. As Point-to-Point Protocol needs two peers to communicate with each other, a remote peer has to be deployed. A remote server, as the PPP peer to our Raspberry Pi, is our missing part we need. The actual topology scenario is shown in Figure 7. The wireless LAN part remains the same as in the case of load balancing. Each of our three mobile broadband connections is to be used to form a PPP connection with our remote server. These PPP connections are to be aggregated together to create a logical link, a multilink bundle. However, to achieve this, a proper environment has to be set in order to get the PPP protocol running over the Internet.

---

7 Since packets originating from Raspberry Pi itself do not hit PREROUTING chain of Iptables, these packets are routed using main routing table.
3.4.1 A Remote Server

We have chosen to use a virtual private server (VPS) based on Kernel-based Virtual Machine (KVM) virtualization. KVM is a full virtualization solution and provides private virtualized hardware: a network card, disk, graphics adapter, etc. The VPS has 512 MB of memory, 20 GB SSD disk, and 1 Gb/s Tier-1 connection. We have decided to use Debian as an operating system for its compatibility with Raspbian run on the Raspberry Pi. We can use the same Debian software packages, as Raspbian is an operating system based on Debian. The version of Debian used is Debian Wheezy 7.3 with kernel Linux 3.2.

3.4.2 Setting the Point-to-Point Environment

The Internet is not a point-to-point network and as a result, first thing to do is to prepare a virtual physical layer environment atop of which PPP protocol can run. We have decided to use Socat and Netcat. These can be installed to the system from the official Debian and Raspbian repositories:

```
(root@pi)# apt-get install socat netcat
```

Once installed, we can proceed to the configuration. The actual configuration is being automated using scripts in bash environment. In this document, we illustrate the main parts of these scripts to provide a picture how the things are being done.

3.4.2.1 Creating the Virtual Serial Ports

First, we have to start with the part that is local to both devices: the Raspberry Pi and the remote server. The following snippet shows the Socat configuration commands:
These commands create three pairs of device nodes with subsequent properties: local echo is disabled; mode is set to raw, which means passing input and output almost unprocessed; baud rate is set to 4000000 baud; parity generation on output and parity checking for input is disabled; and character size is set to 8 bits. With these properties, three pairs of virtual serial ports are created: serA1, serA2; serB1, serB2; and serC1, serC2. Two bidirectional byte streams transfer data between each of these pairs. All of these virtual serial ports are on the same device. Both Raspberry Pi and the remote server will need to have such three pairs of ports. Note that initially we wanted to create three pairs of “virtual serial port – TCP socket”, however, during the initial tests we have found out that TCP connections created via Socat were causing a bottleneck in communication between Raspberry Pi and the remote server for an unknown reason.

### 3.4.2.2 Bridging the Virtual Serial Ports

Due to the aforementioned bottleneck problem, we have chosen Netcat to interconnect the respective pairs of virtual serial ports on both systems. The following three Netcat commands need to be configured on the remote server. These commands create three TCP sockets bound to Netcat. As this is the server side, TCP sockets are put into the listening state. In this case, the server listens on ports 2001, 2002, and 2003. The data that are destined for these TCP ports are redirected to the respective serial ports and vice versa.

```
(root@server)# nc -l 2001 > /dev/serA1 < /dev/serA1
(root@server)# nc -l 2002 > /dev/serB1 < /dev/serB2
(root@server)# nc -l 2003 > /dev/serC1 < /dev/serC2
```

In the case of Raspberry Pi, we need the remote IP address and TCP port numbers to establish the TCP connections. Moreover, we have to specify the IP address of a WAN interface to establish a TCP connection through the appropriate interface along with the use of PBR. There are two options that can happen when other than the IP address from a mobile broadband ISP pool is used. If the ISP is complaint with the BCP38 standard, such traffic will be dropped. If the provider is not complaint, such traffic will not be dropped and on its way back, it will come
through the WAN interface whose IP address was for that traffic used. However, this would cause asymmetric TCP streams and we definitely do not want that to happen.

As already mentioned, we will use PBR to fix this situation. We have decided to use Iptables which will mark TCP traffic that uses destination ports 2001, 2002, and 2003 with fwmark 1, 2, and 3 respectively:

```
(root@pi)# iptables -t mangle -A OUTPUT -p tcp --dport 2001 -j MARK --set-mark 1
(root@pi)# iptables -t mangle -A OUTPUT -p tcp --dport 2002 -j MARK --set-mark 2
(root@pi)# iptables -t mangle -A OUTPUT -p tcp --dport 2003 -j MARK --set-mark 3
```

Next, three custom routing tables with default routes, set for each of the WAN interfaces, are used:

```
(root@pi)# ip route add default dev ppp0 table T1
(root@pi)# ip route add default dev ppp1 table T2
(root@pi)# ip route add default dev ppp2 table T3
```

The last piece of puzzle for our PBR is to add ip rules. All the traffic set with marks 1, 2, and 3 will be routed according to routing information stored in the custom routing tables T1, T2, and T3 respectively:

```
(root@pi)# ip rule add fwmark 1 table T1
(root@pi)# ip rule add fwmark 2 table T2
(root@pi)# ip rule add fwmark 3 table T3
```

When the IP address and TCP ports of the remote server are known, and the PBR is settled, the configuration for the Raspberry Pi can come to pass; three TCP streams between the Raspberry Pi and the remote server are to be established:

```
(root@pi)# nc $remoteIP 2001 -s $ip_wan0 > /dev/serA1 < /dev/serA1
(root@pi)# nc $remoteIP 2002 -s $ip_wan1 > /dev/serB1 < /dev/serB1
(root@pi)# nc $remoteIP 2003 -s $ip_wan2 > /dev/serC1 < /dev/serC1
```

Note that the IP addresses of WAN interfaces are dynamically assigned by the provider of a mobile broadband network. As a result, these IP addresses can vary each time the connections are dialed. For that very reason, we have scripted a handy function:
After all of this has been configured, the environment is prepared for a PPP deployment.

### 3.4.3 Setting the Multilink PPP

First, we need to download and install a PPP daemon. Note that on the Raspberry Pi it has been already installed due to its need for the mobile broadband connection setup. In the case of our remote server, we can download it from the official Debian repository:

```
(root@server)# apt-get install ppp
```

In the snippet below we can see the configuration for our remote server. It is almost identical with the configuration for the Raspberry Pi. The passive option means that PPP daemon will initially send an LCP configuration and if no reply is received from the peer, the daemon will wait passively for a valid LCP packet. Without this option, the LCP would terminate upon timeout expiration (30 seconds by default) when no reply is received. The pair of IP addresses is meant for the multilink bundle. Next, the device or interface is selected. It is one of the local virtual serial ports for each interface pair. All of the PPP traffic is either received on these ports or is sent through them. The ports from the other side of the bidirectional byte stream receive this traffic and hand it over to Netcat. Netcat relays all of the PPP traffic, via already established TCP streams, from the Raspberry Pi to the remote server and vice versa. The speed of 4000000 baud is set in compliance with the underlying virtual serial ports' speed. The `call <profile>` is the last option where the set of options to be requested is specified. These profiles are located in `/etc/ppp/peers` directory. For each peer a unique set of options can be specified. The configuration options on both devices are the same as we want configuration requests to be acknowledged; the interesting options to note are: multilink, MTU 1450 and noauth. The first mentioned option is self-explanatory, the size of MTU is set to be lower than 1500 – PPP overhead in order to avoid fragmentation, and authentication is disabled as the Internet is a public network. The Address, Control, and Protocol field compression as well as Van Jacobson compression are enabled by default.
The following snippet shows the configuration for the Raspberry Pi. The similar configuration is used, although without the passive option:

```
(root@pi)# pppd -detach 172.16.1.2:172.16.1.1 /dev/serA2 4000000 call remote
(root@pi)# pppd -detach 172.16.1.2:172.16.1.1 /dev/serB2 4000000 call remote
(root@pi)# pppd -detach 172.16.1.2:172.16.1.1 /dev/serC2 4000000 call remote
```

Once the remote server is configured, the Raspberry Pi can initiate the multilink bundle establishment. Three separate LCP configuration sessions are held over the three virtual serial lines. The first completed configuration session will establish a regular PPP link. The other two links will join the multilink bundle based on the same Endpoint Discriminator option received by PPP daemons on both machines.

### 3.4.3.1 Routing Through the Bundle

After the multilink bundle has been established, all the traffic has to be routed through it. This is easily accomplished by setting the default route in the main routing table:

```
ip route add default dev $bundle_if
```

In the script, the `$bundle_if` variable is holding the actual interface name of the bundle. The final step is to enable NAT on both machines. In the case of the Raspberry Pi, all the traffic that goes through the bundle interface will be translated:

```
iptables -t nat -A POSTROUTING -o $bundle_if -j MASQUERADE
```

With this NAT we ensure that the remote server will be able to route the traffic back. Likewise, on the remote server we need to translate all the traffic from our private address space that is to be routed to the Internet:
Once the bundle is established and the routing along with the NAT is properly set, we have a fully operational aggregated virtual link based on mobile broadband connections.
4 Testing and Evaluation

Providing Internet access for many local clients is inarguably a bandwidth-intensive task. We have combined multiple broadband connections in order to increase the total speed of our Mobile AP. Thus it is expectable that the speed be the primary subject of our measurements when evaluating the final solution. Other interesting factors are latency, system load and CPU usage.

4.1 Methodology

In the context of wireless networks where the connection is often unstable, speed becomes a volatile measure that is hard to estimate objectively. Variance of instantaneous speed increases as the measurement intervals shorten. It is therefore fundamental to clearly define the parameters of measurement process and take it into account when evaluating the results.

We have carried out two different types of speed tests, each having its informative value for a particular purpose. The speed was measured in kilobytes per second (kB/s) in both cases.

4.1.1 File Download via HTTP

The first method of testing comprised of downloading a 25 MB file from a dedicated HTTP server connected to the Internet via 1 Gbps link. Total time of downloading was measured and used for speed calculation.

This method has an advantage of testing a real use-case scenario when a user is downloading a file via single TCP connection and thus is unable to directly exploit multiple WAN links. This is the primary use-case addressed by the PPP multilink tunnel feature.

4.1.2 BitTorrent Download

The second method simulates many clients demanding a small portion of the bandwidth ad hoc, or a few clients running more network demanding applications. It uses BitTorrent protocol to download a 25 MB portion of a high-availability file, calculating the speed in the same way as in the case of the first method. Since BitTorrent is a peer to peer protocol easily scaling up to hundreds of connections per a single file download, it is a great tool to measure the performance of our setup with traffic load balancing.
4.1.3 Measuring Tools

In both cases, the measurement was carried out from a laptop connected to Raspberry Pi via its WiFi network. Standard wget utility was used for the file download via HTTP, Deluge\(^8\) client for the download via BitTorrent.

We have selected various different setups to test the individual links both separately and together. Comparing the speed of a single WAN link with the combined solution (be it load balancer or tunnel) can prove its efficiency.

4.2 Testing Environment and Appliance

Final solution was tested on real 3G networks available in the urban area of Halmstad, Sweden. We have signed up for common commercial plans at Comviq and Telenor providers and used their networks for transmission speed testing. In order to eliminate arbitrary abnormalities in the networks (e.g. oversubscription) as much as possible, testing was always performed overnight.

Throughout the testing we have used three customary USB 3G modems. Two identical ZTE MF102 HSPA modems were used with Comviq and one Huawei E367 HSPA+ with Telenor\(^9\). All three modems had only built-in antennas.

4.3 Measurement Results

Measuring was done repeatedly for every setup and speed test method. Table 5 and Table 6 show measurement data listing ten independent samples for every test setup. Average and median values are displayed below for easier comparison. Samples with extremely low or high values were disregarded and are not included in the datasets.

Column headings are labeled with the following shortcut names of selected test cases:

- 1xC direct – single direct (without tunnel) Comviq connection,
- 1xT direct – single direct Telenor connection,
- 1xC MT – multilink tunnel over a single Comviq connection,
- 2xC MT – two Comviq connections bundled into multilink tunnel,
- 1C+1T MT – one Comviq and one Telenor connection bundled into multilink tunnel,
- 3x3G MT – all three (2x Comviq + 1x Telenor) connections bundled into multilink tunnel,

\(^8\)http://deluge-torrent.org/
\(^9\)All the modems were vendor-locked for their particular networks so we did not perform tests in different combinations of modem and provider
• 3x3G LB – load balancing across all three mobile broadband connections (without tunnel).

Table 5: Results using file download via HTTP method, in kB/s

<table>
<thead>
<tr>
<th>HTTP</th>
<th>1xC direct</th>
<th>1xT direct</th>
<th>1xC MT</th>
<th>2xC MT</th>
<th>1C+1T MT</th>
<th>3x3G MT</th>
<th>3x3G LB</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>87</td>
<td>286</td>
<td>110</td>
<td>244</td>
<td>205</td>
<td>295</td>
<td>n/a</td>
</tr>
<tr>
<td>#2</td>
<td>136</td>
<td>309</td>
<td>93</td>
<td>225</td>
<td>217</td>
<td>223</td>
<td>n/a</td>
</tr>
<tr>
<td>#3</td>
<td>94</td>
<td>292</td>
<td>83</td>
<td>192</td>
<td>200</td>
<td>288</td>
<td>n/a</td>
</tr>
<tr>
<td>#4</td>
<td>156</td>
<td>328</td>
<td>97</td>
<td>218</td>
<td>216</td>
<td>260</td>
<td>n/a</td>
</tr>
<tr>
<td>#5</td>
<td>128</td>
<td>311</td>
<td>144</td>
<td>206</td>
<td>227</td>
<td>243</td>
<td>n/a</td>
</tr>
<tr>
<td>#6</td>
<td>164</td>
<td>281</td>
<td>126</td>
<td>163</td>
<td>230</td>
<td>274</td>
<td>n/a</td>
</tr>
<tr>
<td>#7</td>
<td>112</td>
<td>316</td>
<td>92</td>
<td>188</td>
<td>221</td>
<td>269</td>
<td>n/a</td>
</tr>
<tr>
<td>#8</td>
<td>81</td>
<td>330</td>
<td>81</td>
<td>210</td>
<td>204</td>
<td>251</td>
<td>n/a</td>
</tr>
<tr>
<td>#9</td>
<td>96</td>
<td>322</td>
<td>155</td>
<td>159</td>
<td>219</td>
<td>321</td>
<td>n/a</td>
</tr>
<tr>
<td>#10</td>
<td>161</td>
<td>304</td>
<td>98</td>
<td>236</td>
<td>201</td>
<td>235</td>
<td>n/a</td>
</tr>
<tr>
<td>Average</td>
<td>122</td>
<td>308</td>
<td>108</td>
<td>204</td>
<td>214</td>
<td>266</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>120</td>
<td>310</td>
<td>98</td>
<td>208</td>
<td>217</td>
<td>265</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Results using BitTorrent method, in kB/s

<table>
<thead>
<tr>
<th>BitTorrent</th>
<th>1xC direct</th>
<th>1xT direct</th>
<th>1xC MT</th>
<th>2xC MT</th>
<th>1C+1T MT</th>
<th>3x3G MT</th>
<th>3x3G LB</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>157</td>
<td>315</td>
<td>204</td>
<td>265</td>
<td>303</td>
<td>340</td>
<td>675</td>
</tr>
<tr>
<td>#2</td>
<td>126</td>
<td>432</td>
<td>92</td>
<td>290</td>
<td>296</td>
<td>367</td>
<td>813</td>
</tr>
<tr>
<td>#3</td>
<td>212</td>
<td>248</td>
<td>151</td>
<td>248</td>
<td>315</td>
<td>392</td>
<td>717</td>
</tr>
<tr>
<td>#4</td>
<td>225</td>
<td>481</td>
<td>146</td>
<td>261</td>
<td>331</td>
<td>335</td>
<td>797</td>
</tr>
<tr>
<td>#5</td>
<td>94</td>
<td>446</td>
<td>160</td>
<td>314</td>
<td>308</td>
<td>384</td>
<td>778</td>
</tr>
<tr>
<td>#6</td>
<td>168</td>
<td>516</td>
<td>188</td>
<td>225</td>
<td>284</td>
<td>407</td>
<td>681</td>
</tr>
<tr>
<td>#7</td>
<td>162</td>
<td>489</td>
<td>159</td>
<td>286</td>
<td>271</td>
<td>396</td>
<td>722</td>
</tr>
<tr>
<td>#8</td>
<td>145</td>
<td>393</td>
<td>166</td>
<td>328</td>
<td>293</td>
<td>375</td>
<td>796</td>
</tr>
<tr>
<td>#9</td>
<td>158</td>
<td>357</td>
<td>133</td>
<td>311</td>
<td>320</td>
<td>388</td>
<td>789</td>
</tr>
<tr>
<td>#10</td>
<td>127</td>
<td>422</td>
<td>129</td>
<td>273</td>
<td>288</td>
<td>305</td>
<td>758</td>
</tr>
<tr>
<td>Average</td>
<td>157</td>
<td>410</td>
<td>153</td>
<td>280</td>
<td>301</td>
<td>369</td>
<td>753</td>
</tr>
<tr>
<td>Median</td>
<td>158</td>
<td>427</td>
<td>155</td>
<td>280</td>
<td>300</td>
<td>380</td>
<td>768</td>
</tr>
</tbody>
</table>

While testing the speed, we have also been watching the system load and CPU usage to see how the flowing traffic consumes system resources. With a single WAN link connected, system load stayed below 0.1 and CPU usage did not even reach 20%. Two WAN links caused the load to go up to 0.5 and CPU usage sometimes exceeded 40%. Finally, traffic flowing through all three links at the same time occasionally resulted in system overload, with the actual load values ranging between 0.8 and 1.3 and CPU usage around 90%.

To see if there is any delay introduced with the multilink tunnel we have also tried to test pings with and without the tunnel in place. The results were however too volatile, ranging between
100ms and 1s (peeking at 5s) regardless of the tunnel existence. Since our multilink concentrator server was located in a data center with direct connectivity to Internet backbone networks, the delay introduced by this extra hop (including the processing necessary to reassemble the packets) was minimal. In comparison with the mobile broadband network latency and its variance this was a negligible value, practically impossible to measure.

4.4 Observations

Looking at the results there are multiple interesting outcomes to notice.

First of all, we can see that despite having the most extreme samples excluded, the results are still very variable. The variance of our Internet connection speed is nowhere near the common speed variance of wired or even WiFi connections. On the other hand this was quite expectable as 3G networks are known to be prone to various external factors that we cannot influence, especially bandwidth demand of other customers connected at the same time.

4.4.1 Bittorrent vs. HTTP

Another interesting thing to notice is the difference in speeds measured via HTTP and BitTorrent. The latter shows consistently better results in all setups. We believe the most plausible explanation of this disparity lies in TCP flow control logic which does not perform well over too lossy wireless channels. Tens of independent streams initiated by BitTorrent can possibly adapt to link speed changes more quickly than a single one initiated by HTTP file download. Also, a random error only affects a single stream (causing it to back off) while preserving the speed of others used by the BitTorrent client. The impact on total speed is thus substantially minimized.

4.4.2 Multilink Tunnel vs. Direct Access

A positive observation is that the multilink tunnel itself does not have any significant adverse impact on the speed, at least when used only over a single WAN link. PPP field compression is apparently quite efficiently limiting the inevitable overhead imposed by the tunnel.

However, the regression starts to be bold in the setup with three WAN links. While one could expect the total speed to be approaching the sum of the individual speeds of all three WAN links, it is certainly not the case. In fact, the higher boundary is limited by the slowest link in the bundle, which has to bear the same load as any other link due to round-robin algorithm used in pppd multilink implementation.
Moreover, the limitation applies at any given instant of time, which has further negative consequences. If we think of the probability of a single link experiencing a short outage in some time interval, then it is obvious that the chance of any link having outage during the interval increases with the total number of links. Any single link can then drag the total speed down at any instant of time, which surely has its implications in global merit.

4.4.3 Load Balancing Performance
The best results were achieved in the load balancing setup, which is after all not surprising. Individual streams are spread amongst all the WAN links with no additional tunnel overhead. The total speed is thus truly determined by the sum of individual speeds of all available WAN links. Obviously, this can only be fully exploited with more parallel connections, which is however perfectly expectable on a network with many connected clients demanding Internet access.

4.4.4 System Utilization
With download speeds approaching 800 KB/s, achieved while load balancing the traffic across three WAN links, system resources of the Pi were being almost fully utilized. Even though the CPU usage hit 100% only sporadically and oscillated around 90% most of the time, system load was constantly exceeding the 1.0 reference value. Considering the nature of the tasks the system was running (demanding multiple I/O operations for every packet), these observations are consistent with reasonable expectations.

It also suggests that the maximum bandwidth RPi could handle in this setup is expected to be close to the best values we have measured. Adding a fourth WAN link (or increasing the speed of existing ones) would thus most probably not lead to overall improvement in bandwidth provided for end clients. These assumptions are, however, only relevant to the configuration we have used and do not express any theoretical limit. It is certainly possible that more efficient setup would remove the bottleneck and allow for higher maximum speeds.
5 Conclusion

Throughout the thesis we have experimented with integration of various networking technologies, their respective software implementations and custom configurations for specific setups. We have explored some interesting Linux interfaces and capabilities in order to build a solution comprised of numerous individual components fit together. This includes setting up 3G modems, WiFi access points, DHCP and DNS server, traffic load balancing across multiple outgoing interfaces and link bundling via PPP multilink feature. We have also worked with an external server supporting our infrastructure by concentrating the traffic coming to and from the multilink bundle.

Getting all these technologies working on the Raspberry Pi, we have shown the universality of the platform and its maturity as a flexible general purpose system, with respect to both hardware and software extensibility. Trying out various configurations we have created test cases and proceeded with the performance testing focused on speed tests.

Joining the WAN links into one PPP multilink bundle turned out to be quite error-prone with limited use cases, but we still consider it a vital exercise and a useful proof-of-concept solution. Perhaps a PPP multilink implementation with different (more adaptive) algorithm would be more efficient than the simple round-robin scheme employed by pppd. Other technologies such as L2TP or other VPNs come to consideration for future research and improvements as well.

On the other hand, the load balancing scenario provided satisfactory results and has proven to be efficient in terms of bandwidth aggregation. We believe it would make a good fit for use in real production environment, e.g. trains, buses or as a compact portable device to bring to business meetings or conferences taking place in external or outpost locations, and set up a temporary WiFi AP for attendants.

Finally, we would like to state that, even though our solution was not inherently limited to any specific number of mobile broadband interfaces employed, we have not tested it with more than three dongles because of limited resources. Nonetheless, it was enough to almost exhaust Raspberry Pi computational resources. It would certainly be interesting to see the setup perform on other compatible but more powerful platforms, such as CuBox\textsuperscript{10}, Hummingboard\textsuperscript{11} developed by Solid-Run or Banana Pi\textsuperscript{12} from OSSUG company. We believe there are still many ways to explore the topic beyond this scope, stretching the goals and pushing the limits even further ahead.

\textsuperscript{10} http://imx.solid-run.com/wiki/index.php?title=CuBox_i_Hardware
\textsuperscript{11} http://imx.solid-run.com/wiki/index.php?title=HummingBoard_Hardware
\textsuperscript{12} http://www.lemaker.org/
6 References


3. Developers’ documentation for wpa_supplicant and hostapd. [Online] [Cited: 5 4, 2014.]

4. Gheorghe, Lucian. Designing and Implementing Linux Firewalls and QoS using netfilter,


    https://github.com/Trixarian/sakis3g-source.

Lukas Chrast
Computer engineer interested mainly in networks and Unix. Obtained bachelor degree in study program Computer and Communication Systems and Networks at Slovak University of Technology.

Jozef Knaperek
Free software enthusiast, passionate developer, linux and networking fella. He has done his bachelor thesis dealing with network service monitoring at Slovak University of Technology in Bratislava.

Marek Kovalcik
Sort of a guy focused on computer systems and networks, and yet he loves woods and mountains. He has earned a bachelor degree in Computer Engineering from Slovak University of Technology in Bratislava.