JACOBS UNIVERSITY

Traffic Differentiation Detection in Mobile Networks using Android Phones

by

Vitali Bashko

A thesis for conferral of a Master of Science in Computer Science

Prof. Dr. J. Schönwälder
Name and title of first reviewer

Dr. Bendick Mahleko
Name and title of second reviewer

Date of Submission: July 29, 2012

Jacobs University — School of Engineering and Science

Declaration

I, Vitali Bashko, born 1986-06-25, hereby declare that this thesis is the result of my independent work, has not been previously accepted in substance for any degree and is not concurrently submitted for any degree.

This thesis is being submitted in fulfillment of the requirements for the degree of Master of Science in Computer Science.

. . . .

Place, Date Vitali Bashko

Acknowledgments

I would like to thanks to Prof. Dr. Jürgen Schönwälder for many helpful comments and his guidance throughout this thesis. Special thanks to Anuj Sehgal and Nikolay Melnikov for helping me setup the experimental environment, and warm thanks to Lizaveta Mikhalchanka for copy editing and supporting me during my studies.

Abstract

One of the interesting topics in the networking area is so-called ”network neutrality”. Many Internet Service Providers (ISPs) want to restrict bandwidth-hungry applications that can hurt other applications in the network or want to control applications, such as VoIP, that reduce ISPs’ ability to profit from competing services of their own. Multiple research projects have been done and a number of measurement tools have been developed in order to investigate how different traffic regulation policies are deployed by ISPs. However, the situation with traffic shaping in mobile networks is still unexplored, mainly because of a lack of a proper measurement infrastructure for mobile devices. In this thesis, a new measurement tool for Android-powered mobile devices is presented, which is suitable for detecting the presence of content-based traffic shaping on mobile networks. A series of experiments on mobile networks has been done using the new measurement tool. As it turned out, most tested mobile providers do not block SIP and VoIP traffic, even despite the fact they claim that VoIP is not supported in their mobile networks.

Contents

Chapter 1

Introduction

The Internet is a large distributed system that involves the network infrastructure, the hosts it connects, the traffic generated by the hosts and the protocols that govern the transmission of packets between hosts across the network. Proper monitoring, performance assessment, maintenance, and troubleshooting of networks play a fundamental role in assuring the desired quality level of the offered service. Performance critical applications already exist, from financial transactions to streaming multimedia, which require some Quality of Service quantities to be measured as accurately as possible. Plenty of tools have been developed to perform active and passive Internet measurement tests and to retrieve useful information about the current network state.

Nowadays, the number of mobile devices, which can connect to the Internet, and the amount of mobile traffic are increasing rapidly. During the last couple of years, so-called smartphone devices became very popular with the users around the world. There is no standard definition of the term smartphone across the industry. However, the popular explanation defines a smartphone as a device that allows users to make telephone calls, and also provides the functions of a personal digital assistants (PDAs), portable media players, compact digital cameras and GPS navigation units [1]. In addition, the following features are typical for smartphones [2]:

– **Operating System**: a smartphone is based on an operating system that allows it to run third-party applications;

– **Software**: while almost all the cell phones include some sort of software, a smartphone provides an advanced application programmer interfaces (APIs) for running third-party applications, which allow those applications to have better integration with the smartphone's operating system and hardware (e.g., interaction with video cameras, different built-in sensors, GPS, gaming controls, etc.). This allows to run more complex applications which provide additional functionality;

– **Internet Access**: smartphones can access the Internet via Wi-Fi and mobile broadband;

– **Keyboard**: smartphones usually include a QWERTY keyboard which can be hardware or software (users type on a touch screen).

– **Messaging**: All cell phones can send and receive text messages, but what sets a smartphone apart is its handling of e-mail. A smartphone can sync with multiple e-mail accounts.

The smartphone users become active members of the Internet community. The users of mobile devices use various services such as web surfing, email, video and audio streaming. Network traffic generated by mobile devices is increasing at this very moment along with the number of smartphone users and mobile services. In our opinion, being able to run network measurement tests from smartphone devices in order to discover whether provided connectivity performance satisfies users expectations is a very useful ability to end-users. However, the existing network measurement tools for mobile platforms (Android, iPhone, Windows Phone OS) mostly allow only to obtain the basic network information (e.g., local and global IP addresses of the mobile device) and to measure the network performance (e.g., downlink/uplink throughput, latency, round trip time).

One of the interesting topics is the so-called ”network neutrality”. Many Internet Service Providers (ISPs) want to restrict bandwidth-hungry applications that can hurt other applications in the network. Some also want to control applications, such as VoIP, that reduce ISPs ability to profit from competing services of their own. In contrast, some other content providers are against traffic differentiation because it gives the ISPs arbitrary control over the quality of service experienced by users. Many research projects have been done [3][4][5] and a number of measurement tools [6][7][8] have been developed in order to investigate how the different traffic shaping policies are deployed by ISPs. However, the situation with traffic shaping in mobile networks is still unexplored, mainly because of a lack of a proper measurement infrastructure for mobile devices.

Thereby, during this master thesis project, we focus on developing a measurement tool suitable for running active Internet measurements on Android-powered mobile devices. The main goal of our Master Thesis project is to develop an application for an Android Operating System (OS) [9] that helps to reveal the presence of traffic differentiation based on deep-packet inspection techniques. The solution is based on a client-server architecture. Client connects to a measurement server to download and run various tests. During each test, the server and the client components exchange messages that carry application-level data, which conforms to the application protocol that we want to test. This data is carefully constructed to detect traffic differentiation along the path. The client component can be installed on any Android-powered smartphone. And the server component runs on a Linux machine, which has a static global IP address. In order to distribute the client component among users, it has been published on Google Play [10] digital-distribution multimedia-content service.

In addition, we evaluate the results obtained from the experiments that have been made on mobile networks in order to learn how and to which extent content-based traffic differentiation policies are deployed by different mobile service providers. When we talk about mobile networks, we can speculate that mobile network operators might want to manipulate on the VoIP application's traffic performance. VoIP applications for smartphone devices (Skype, Viber, Sipgate, MobileVoip) actually are competitors for mobile service operators. To reduce financial losses it is tempting to restrict the performance of traffic flows that carry VoIP data. In fact, many mobile Internet providers (e.g., *NettoKOM*, *O_2*, *Congstar*) claim [11][12][13] that they do not support VoIP and Peer-to-Peer traffic on their networks.

By analyzing the obtained measurement results, we will try to answer the following questions about the presence of the content-based traffic shaping in mobile networks:

a) Do mobile Internet Service Providers apply traffic shaping techniques based on the deep-packet inspection in order to restrict the performance of "unwanted" applications. Does it depend on the type of the network? For example, the same mobile operator provides Internet access using different network types (e.g. HSPA, EGDE). Thereby, it is interesting to know whether and how content-based traffic differentiation policies differ on different network types within the same provider.

b) Does the performance of certain application protocols in mobile networks depend on the time of the day? Can we observe the difference in applications' performances between day time and night time? Do mobile operators perform application-depend traffic shaping policies during the peak hours, when the number of concurrent users is larger than usually?

c) Most of mobile providers reduce the maximum network bandwidth for those users who exceeded the monthly data download limit defined by a contract. Therefore, we will try to determine whether the mobile operators perform content-based traffic shaping in case when the user has crossed that data limit.

Chapter 2

State of the Art

2.1 Network Measurements Overview

Internet measurements techniques play an important role for network operators and providers. They are an efficient and robust way to determine how well a network performs and what kind of Quality of Service guarantees Internet providers are able to offer to their customers. By gathering data from the measurement tests, network providers can retrieve useful information which lately can be used for [14]:

- Performance adjustment: identifying and reducing bottlenecks, balancing resource use, etc.
- Troubleshooting: identifying and repairing faults end misbehavior
- Planning: predicting the scale and required hardware resources
- Characterization of traffic for providing data for modeling and simulation
- Understanding and controlling complexity: understanding the interaction between components of the network and to confirm that functioning, innovation and new technologies perform as predicted and required

Network measurement methods can fall into two categories depending on the way how data has been collected: active and passive measurements. These two types of measurements generally focus on different aspects of network behavior.

In passive measurements, routers or other hosts measure existing traffic passing through or destined to them. Typically data is collected in three ways: polling management information base (MIB) data from routers, packet monitoring and flow monitoring. Passive network measurements are useful in order to learn the network topology, analyze the network workload or measure some end-to-end performance metrics like packet loss and delay [15].

Unlike the passive methods, in active networking measurements we additionally inject probe packets into the network and measure network characteristics at different points. The key idea is that we know the initial structure of the probe traffic, and so by measuring how it is affected by the portion of the network it traverses, network conditions can be inferred. Usually, by active measurements we mean end-to-end measurements, when the probes are observed only at the origin and destination hosts, so there is no need to deploy additional monitors in between. The drawback of probing is that injected packets may possibly disturb the normal traffic flow. This is why active measurements need to be carefully planned before execution and usually the bandwidth reserved for the probe packets is limited. Typically, active measurements are used in order to determine network performance characteristics such as latency, one-way or two-way packet delay, round-trip time, stream or upload speed or hosts availability.

In addition, combining active and passive measurements is called hybrid measurement. An example of a hybrid measurement could be a scenario where active probes are sent over a network and their progress is monitored by passive means during the measurement. This type of arrangement allows the measurer to track the path of the probes and record the intermediate and end-to-end delays.

The Internet performance measurements have a long history, and during the last 30 years plenty of methodologies and techniques have been proposed and a large number of measurement tools have been developed. Thus, we can distinguish several categories of network measurement types accordingly to their intentions: *network performance measurements, routing and topology measurements, audio and video streaming characteristic measurements, measurements in case of multicast communication, application and policy measurements*.

2.1.1 Performance Measurements

One of the goals of Internet measurement is to determine the network performance. For example, some applications benefit from knowing the amount of bandwidth available on a network path, so that they can adapt their sending rate and share the bandwidth more fairly. Performance measurements focus on the analysis of end-to-end behavior and on the diagnosis of network problems. These efforts typically include the collection of end-to-end packet loss, delay and round trip time statistics, most often performed by injecting test traffic into network, thus active measurements techniques are usually used in this case.

The most common metrics typically measured are latency, packet loss, throughput, utilization and availability. In order to standardize the networking measurement metrics to make the comparison of obtained results from different sources possible, the Internet Engineering Task Force's (IETF) IP Performance Metrics Working Group (IPPM) develops standards, called Requests For Comments (RFC), for the measurement of network performance. By July 2012 the following metrics had already been defined by RFC documents:

- IPPM Metrics for Measuring Connectivity (RFC 2678)
- A One-way Delay Metric for IPPM (RFC 2679)
- A One-way Packet Loss Metric for IPPM (RFC 2680)
- A Round-trip Delay Metric for IPPM (RFC 2681)
- A Framework for Defining Empirical Bulk Transfer Capacity Metrics (RFC 3148)
- One-way Loss Pattern Sample Metrics (RFC 3357)
- IP Packet Delay Variation Metric for IPPM (RFC 3393)

Determining the current network performance presents a quite mature field of interests, and a large number of measurement techniques have been proposed. The mechanism of determining the Maximum Transmission Unit (MTU) is defined in RFC 1191 [16]. The core idea is to use the IP header's *"Don't Fragment"* bit to discover the Path MTU (PMTU). A source node first assumes that the PMTU is the MTU of the first hop. If a router on the path notices that the datagram cannot be sent to a next hop without fragmentation, the router drops the packet and sends an ICMP Destination Unreachable message back to the source node. When a sender node receives these messages, it automatically reduces the size of the packets and thus the PMTU until it receives no more error messages.

R. Prasad, C. Dovrolis, M. Murray, K. Claffy in [17] provide a detailed review of existent methods to determine the available network bandwidth. There are four major techniques that are used:

1. *The Variable Packet Size (VPS)* technique attempts to estimate the capacity of each link along a path. VPS does this by sending different sized probe packets from the source node to all nodes along the path and measuring the round-trip delay time (RTT) to each hop as a function of packet size.

2. *The Packet Pair/Train Dispersion (PPTD)* technique sends multiple identical (in terms of size) packets back-to-back and measure the dispersion of the packets at the receiver side. The narrow link on the path causes an increase in the dispersion of the packets, so the available bandwidth can be detected.

3. *The Self-Loading Periodic Streams (SLoPS)* technique is able to estimate the available bandwidth. It is based on principle of sending the sequence of equal sized packets at an increasing rate and to monitor the one-way delay variations experienced by the packets.

4. *The Trains of Packet Pairs (TOPP)* technique is much like the *SLoPS* method but it uses different packet stream patterns and focuses on reducing measurement latency. In addition, instead of just estimating the available bandwidth it is also able to estimate the tight link on the path.

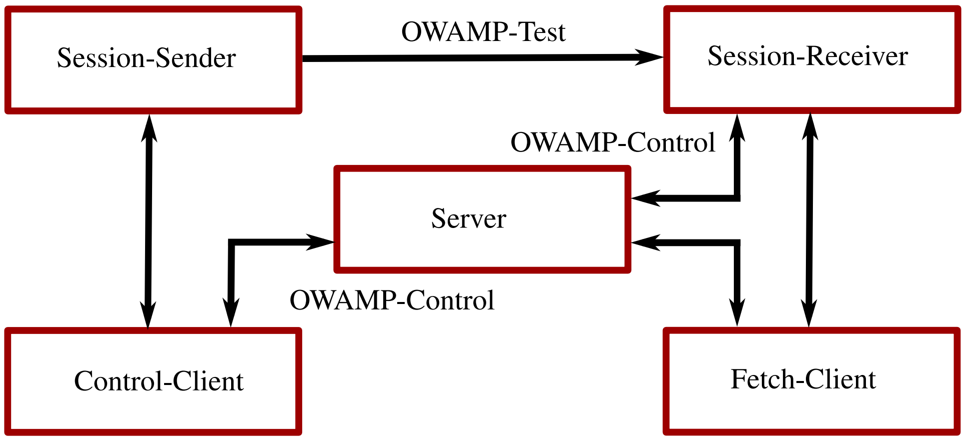


Figure 2.1: OWAMP roles distribution

The One Way Active Measurement Protocol (OWAMP) defined in RFC 4656 [18] provides a high precision mechanism to measure one-way delay and latency in the Internet. Additional design goals of OWAMP include: being hard to detect and manipulate, security, logical separation of control and test functionality, and support for small test packets. OWAMP also supports an encrypted mode that further obscures the traffic and makes it impossible to alter timestamps undetectably. Technically OWAMP consists of two inter-related protocols: OWAMP-Control and OWAMP-Test. OWAMP-Control is used to initiate, start, and stop test sessions and to fetch their results, whereas OWAMP-Test is used to exchange test packets between two measurement nodes. The OWAMP architecture separates different roles in order to be more flexible on scheduling and executing performance measurements (see Figure 2.1).

- Session-Sender: the sending host of the test session
- Session-Receiver: the receiving host of the test session
- Server: manages the test sessions, configures per-session states in the session endpoints, and returns the results of a test session
- Control-Client: initiates requests for test sessions, triggers the start or termination of test sessions
- Fetch-Client: initiates requests to fetch the results of completed test sessions

The principle of OWAMP is quite simple: the test packets are sent from the sender to the receiver and the packet's timestamps (send and receive times), sequence numbers and TTLs are recorded on arrival. As OWAMP measures the one-way delay by comparing the timestamps on the sender's and receiver's end, it is clear that the clocks of both the sender and the receiver need to be synchronized.

Similar to OWAMP, the Two-way Active Measurement Protocol (TWAMP)

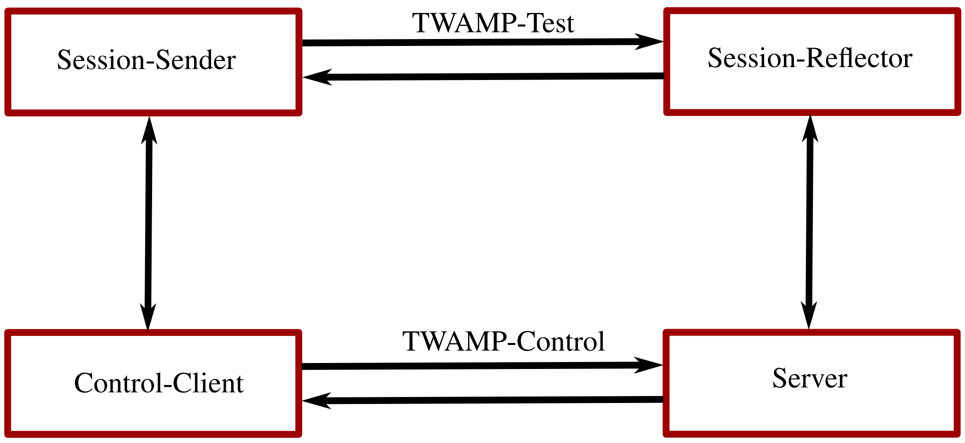


Figure 2.2: TWAMP roles distribution

[19] has been developed in order to measure two-way packet delay and latency characteristics. The TWAMP provides a different role model, the Session-Receiver is replaced by the Session-Reflector which is capable of creating and sending test packets when it receives test packets from a Session-Sender. Unlike the Session-Receiver, it doesn't collect any information from test packets as round-trip delay information is available only after the reflected test packet has been received by the Session-Sender. Another exception is that the Server component doesn't have the capability to return the results of a test session as the Session-Reflector it is associated with doesn't collect any results. Consequently, this means that there is no need for a Fetch-Client and thus it does not exists in the TWAMP architecture (see Figure 2.2).

In addition, plenty of network performance measurement tools have been implemented. The most popular are: *iperf, pathload, traceroute* and *ping*. On the CAIDA [20] web page, a rich collection of performance measurement tools with their descriptions can be found.

2.1.2 Routing and Topologies Measurements

Another goal of Internet measurements is to learn about the network topology and traffic routes from the source to destination hosts. The knowledge about routes is extremely helpful for understanding the current situation in a network and for troubleshooting. For instance, in some cases there might be a fault in the network that causes traffic to be routed the wrong way. Generating an artificial traffic flow through the network and inspecting its behavior can help to troubleshoot routing faults. Running passive measurements and observing data flows can also help to detect bottlenecks in the network configuration.

The global infrastructure of the Internet is continuously changing. It is a really challenging task to track and visualize this complex system. The goal of topology measurements is to collect information on connectivity, which backbone node is connected to which, and to determine their geographical locations. Routing measurements provide insight into the dynamics of routing protocols and routing table updates. These are of great importance as the reliability and robustness of the Internet depends on the stability and efficiency of routing.

The Cooperative Association of Internet Data Analysis (CAIDA) [20] used the *skitter* [21] tool more that 10 years to monitor the Internet's topology. *Skitter* allows to record each hop from a source to many destinations, collect round trip time (RTT) along with path (hop) data, and collect the data to visualize the network connectivity by probing the paths to many destination IP addresses spread throughout the IPv4 address space.

On February 2008, CAIDA deactivated skitter data collection and transitioned to the measurement infrastructure named *Archipelago* [22], which provides a more powerful and flexible IPv4 and IPv6 traceroute active measurement tool. The initial focus of *Archipelago* is coordinated large-scale traceroute-based topology measurements using a process called team probing. In team probing, monitors are grouped into teams and dynamically divide up the measurement work among team members.

NetConfigs.Com [23] provides to ISPs a suite of tools to verify their peering policies, identify errors, and apply (or request) policy changes accordingly. Peering data (IPv4 prefixes and AS-paths) is collected from around the world and processed into a global-view database against which AS reports and BGP tools can be run.

LinkRank [24] is a graphical tool for visualizing BGP routing changes. This tool can be used by BGP operators to understand routing dynamics as well as by people who want to learn more about BGP. *LinkRank* summarizes megabytes of BGP updates received from collection points and produces easy to understand graphs indicating the segments of routes affected.

Similar functionality is provided by the *BGPlay* [25] tool. *BGPlay* is a Java application which displays animated graphs of the routing activity of a certain prefix within a specified time interval. Its graphical nature makes it much easier to understand how BGP updates affect the routing of a specific prefix than by analyzing the updates themselves.

The *iffinder* [26] tool allows to discover which IP addresses belong to interfaces on the same router.

2.1.3 Measurement of Video and Audio Streaming Performance

Internet active measurements also might be useful in more application specific scenarios, for example in measuring channel characteristics in case of voice and video data streaming. Multimedia streaming is a real time application, as a result one of the the main challenges are optimizing video quality by active measurements of links and enhancing the overall Quality of Service (QoS). The QoS itself is a complex function which depends on many factors including the communication channel characteristics like throughput, available bandwidth, jitter, packet delay and loss. So, it is very important to measure those characteristics precisely, in order to dynamically adopt the data packets streaming to achieve the best Quality of Service.

Mubashar Mushtaq and Toufik Ahmed in [27] presented a solution based on active measurement of RTT values, which allows to perform smooth quality adaptation for streaming of IP packet video. They used a receiver-centric mechanism i.e., the receiver peer is in charge for selection of active peers and it also coordinates the overall streaming mechanism by switching from one congested node to another present in the subset of candidate peers offering better QoS.

Reza Rejaie and Antonio Ortega have proposed a framework PALS [28]. PALS is a receiver-driven framework for quality adaptive playback of encoded media streaming, where a receiver coordinates delivery of layer encoded stream from multiple senders. A peer selection criterion has been proposed based on the overall effective throughput. There is no information available in the start so initial peers are selected on random basis.

Prasad Calyam et al, in [29], describe common end-to-end performance problems in the H.323 protocol, that defines how real-time multimedia communications, such as audio and video-conferencing, can be exchanged on packet-switched networks (Internet). Furthermore they developed the H.323 Beacon tool that can be used to measure, monitor and qualify the performance of an H.323 video conference session. It can help an end-user/network engineer operator, as a debugging tool by providing H.323-protocol specific evidence and other information necessary to troubleshoot H.323 application performance problems in the network.

Qian Zhang, Wenwu Zhu and Ya-Qin Zhang in [30] designed an end-to-end distortion-minimized resource allocation scheme for scalable video transmission over 3G wireless network using channel-adaptive hybrid unequal error protection (UEP) and delay-constrained automatic retransmission request (ARQ) error control schemes. They presented dynamic measurements of error rate and throughput for 3G wireless networks, which were used for resource allocation between source and channel coding.

M. Lundeval, B. Olin and others in [31] evaluate different scheduling algorithms for video streaming over High-Speed Downlink Packet Access (HSDPA) networks. The HSDPA communication protocol improves downlink performance on mobile networks and is currently being deployed in networks around the world. The Authors consider performance aspects of different scheduling algorithms with the aim of providing QoS for streaming applications in a scenario with mixed interactive (web browsing) and streaming services.

2.1.4 Multicast Measurements

Another type of network measurements that we can distinguish are measurements in case of multicast communication. In this scenario, multicast datagrams are delivered to all members of their destination host group, which changes dynamically during the time. That means that hosts may join and leave the group at any time, and usually there is no restrictions on the location of members in the host group. The forwarding of IP datagram is handled by so-called ”multicast routers”. The structure of data flow shapes a tree and the data flows from the root (usually source of the data) to the leaves. So, the multicast data monitoring becomes a challenge, because it involves monitoring the distribution trees for each of the senders and all branches of each tree.

One more important factor, that makes multicast measurements difficult is the lack of information about receivers; every node that sends data to a multicast group only knows about the nodes from the next level of the tree. Given these characteristics of multicast communication, the set of measurement metrics as well as measurement methodology is different than the in unicast case. Instead of measuring a single value that is a channel characteristic, it makes more sense to measure the set of the specific values between the source and each receiver. The following metrics can be used to describe the quality of multicast communication channel:

- The set of the one way delay values between the source and each receiver
- The set of the packet loss results between the source and each receiver
- The number of time the measurement packet is repeated by a node (for routers)

T. Saadawi in [32] extends unicast resource measurement techniques to multicast environments to estimate bottleneck link bandwidth. They also introduced a novel way to measure end-to-end queuing delay by using a pair of packets with different priorities. The major benefit of the proposed method is that it doesn’t require time synchronization and does not need to communicate with core routers.

Multicast monitoring tools can be divided into several categories: *RTP monitoring tools, multicast routing diagnostics, multicast traceroute tools and multicast backbone (MBONE) mapping tools*. Tools from the first group (RTP tools) are used to monitor the quality of the data transmission and popularity of individual sessions: *MHealth* [33] is a graphical, near real-time multicast monitoring tool. By using a combination of application level protocol data about group participants, and a multicast route tracing tool for topology information, *MHealth* is able to discover and display the full network tree distribution and delivery quality. *MHealth* also provides data logging functionality for the purpose of isolating and analyzing network faults. Logs can be analyzed to provide information such as receiver lists over time, route histories and changes, and the location, duration, and frequency of loss. *Mlisten* [34] was developed for the collection and processing of MBONE membership information. The tool can be used to generate information about join and leave statistics, connection time characteristics, and multicast tree size and characteristic. *RTPMon* [35] allows network administrators or support personnel to monitor listenership as well as session quality experienced by subscribers. The tool also facilitates tracing the cause of problems resulting in quality degradation. To accomplish this task, *RTPMon* summarizes and analyzes information provided by RTCP source and receiver reports.

Multicast router diagnostics are used to collect information about the state of multicast routers: *mrinfo* [36] displays information about a multicast router to discover the router's physical and virtual interfaces. Routes are queried for their version number, and if this query is successful, for their metrics, thresholds and flags. *Mantra* [37] is a tool for monitoring the multicast traffic at the router level. It periodically collected multicast routing information (e.g., MSDP and MBGP tables) from multicast-enabled backbone routers in the Internet. Then *Mantra* processed this information to generate useful statistics about the deployment and availability of multicast across the inter-domain. The information collected by *Mantra* has helped researchers and network administrators understand multicast operation, routing protocol interaction and evolution of the infrastructure.

Multicast traceroute tools are used to trace the path between the sender and destinations. One of the most popular tools is *Mtrace* [38]. The *Mtrace* tool is used to return a snapshot of the set of links used to connect a particular source with a particular destination. Additional *Mtrace* options allow a user to measure the number of multicast packets flowing across each hop.

MBONE mapping tools are used to map out the topology for a particular group. These tools can show the topology at the level of individual systems, or at the level of autonomous system connections. *Mrtree* [39] uses a combination of IGMP and SNMP queries to discover the actual and potential multicast (sub)trees for a given source and group, rooted at a given router. An actual tree, discovered using the multicast routing MIB, consists of routers which are currently forwarding multicast traffic to a group from a given source.

2.1.5 Application and Policy Measurements

In the real world the traffic management policies usually are driven by business interests (depending on peering and data transition agreements between providers), and many ISPs do not publicly disclose the details of their middlebox deployments. Those middleboxes, such as firewalls, blockers and traffic shapers are used to monitor and manipulate the data flown through. To achieve more revenue, ISPs may use different techniques, for example the traffic shaping to optimize the performance.

Traffic shaping is a network management technique which delays some or all datagrams to bring them into compliance with a desired traffic profile [40], and it can be used to control the volume of traffic being sent into a network in a specified period, or the maximum rate at which traffic is sent. From the one side, using intelligent traffic shaping schemes may guarantee a particular Quality of Service for an applications, and it can help IPSs optimize the use of their network, e.g., to avoid the congestive collapse in the Wi-Fi based protocols. However, from the end-user perspective view, using this technique means that data which is sent by a user into the network might be artificially delayed by its ISP, which naturally is not what users expect from their providers. Recently, it has been reported that certain access ISPs [41] are blocking their customers from uploading data using the popular BitTorrent protocol. The ISPs were found to tear down TCP connections, identified as BitTorrent flows, by sending forged TCP reset packets to the end hosts. Furthermore, not only the P2P traffic can be blocked, but other data like flash video, email or SSH transfer might be restricted as well, depending on the provider's policy configuration.

To differentiate between flows of different types, i.e., belonging to different applications, ISPs must distinguish the packets of one flow from those of other flows. This can be done by examining one of the following [3]:

– *The IP header*. The source or destination addresses can determine how an ISP treats a flow. For example, universities routinely rate-limit only traffic that is going to or coming from their student dorms.

– *The transport protocol header*. ISPs can use port numbers or other transport protocol identifiers to determine a flow's treatment. For example, P2P traffic is sometimes identified based on its port numbers.

– *The packet payload*. ISPs can use deep-packet inspection (DPI) to identify the application generating a packet. For example, ISPs look for P2P protocol messages in packet payload to rate-limit the traffic of P2P applications, such as BitTorrent.

In addition to features of a flow itself, an ISP may use other criteria to determine whether to differentiate. Some of these include:

– *Time of day*. An ISP may differentiate only during peak hours.

– *Network load*. An ISP may differentiate on a link only when the network load on that link is high.

– *User behavior*. An ISP may differentiate only against users with heavy bandwidth usage.

There are a number of ways how an ISP can treat one class of packets differently:

– *Blocking*. One form of differentiation is to terminate a flow, either by blocking its packets or by injecting a connection termination message (e.g., sending a TCP FIN or TCP RST packet).

– *Deprioritizing*. Routers can use multiple priority queues when forwarding packets. ISPs can use this mechanism to assign differentiated flows to lower priority queues and to limit the throughput of certain classes.

– *Packet dropping*. Packets of a flow can be dropped either using a fixed or variable drop rate.

– *Modifying TCP advertised window size*. ISPs can lower the advertised window size of a TCP flow, prompting a sender to slow down.

– *Application-level mechanisms*. ISPs can control an application's behavior by modifying its protocol messages. For example, transparent proxies [42] can redirect HTTP or P2P flows to alternate content servers.

As a result, several techniques were developed, based on the Internet active and passive measurements, which allow to detect if ISPs use traffic blocking or traffic shaping. Marcel Dischinger et al in [4] present a large-scale measurement study of BitTorrent traffic blocking by ISPs. They designed and developed a tool called *BTTest*, which enables end users to test for blocking on their own access links. *BTTest* emulates BitTorrent flows between end hosts and test servers, using the standard BitTorrent protocol and identifies the traffic blocking based on flow characteristics.

Another tool that has been designed and implemented by Marcel Dischinger, Massimiliano Marcon and others is called *Glasnost* [3]. *Glasnost* is a tool that attempts to detect whether your Internet access provider is performing application-specific traffic shaping, It allows you to test if your ISP is throttling or blocking email, HTTP or SSH transfer, Flash video, and P2P applications including BitTorrent, eMule and Gnutella.

NetPolice [6] (previously named NVLens [43]) compares the aggregate loss rates of different flows to infer the presence of ”network neutrality violations” in backbone ISPs.

NANO [5] uses causal inference to infer the presence of traffic performance degradation. NANO relies on a vast amount of passively collected traces from many users to infer if traversing a particular ISP leads to poorer performance for certain kinds of traffic. It uses active measurements and a simple head-to-head comparison of two flows to quickly inform users whether they face traffic differentiation – without relying on other users.

ShaperProbe [7] also detects whether traffic shaping is used in the upload or download directions, and in that case that it is used, ShaperProbe reports the shaping rate and the ”maximum burst size” before shaping begins.

DiffProbe [8] detects whether traffic differentiation based on active queue management (AQM), such as RED and weighted fair queueing, is deployed in the network path. DiffProbe can detect differentiation that leads to small increase in latency and can identify the AQM technique used.

Neubot [44] runs in the background, as a daemon, and periodically performs tests to measure network performance and application-specific throttling. Currently, two tests are implemented: one that emulates HTTP flows and another that emulates BitTorrent flows.

2.2 Internet Measurement Infrastructures

Another important aspect of network measurements is the problem of sharing large Internet measurements datasets between different research projects. While standalone tools that have been developed recently allow to perform accurate measurements to test network performance, which is important to the network administrators, researchers need to collect their own data, and perhaps even develop their own measurement tools before investigating a given question. Furthermore, a network operators also might be interested in obtaining real-time data in order to optimize their network infrastructure, since the data-heavy multimedia network application becomes more and more popular. In an effort to make the gathering of the measurement data in a large scale easier and attract more participants (individual persons, as well as institutions) involved into the measurement process, several projects have been founded by different groups of researchers, that provide a well-designed infrastructure for performing broadband connection measurement and collecting the results.

2.2.1 M-LAB

Measurement Lab (M-Lab) [45] is an open, distributed server platform for researchers to develop, test, and deploy new active measurement tools. The goal of M-Lab is to advance network research and empower the public with useful information about their broadband connections. Currently, instead of focusing on the Internet core, M-Lab focuses on measuring the end-to-end performance and on the characteristics of broadband access links. Measurements capture basic operational characteristics (e.g., TCP throughput, available bandwidth), advanced host diagnostics (e.g., misconfiguration, small socket buffer sizes), and ISP traffic management practices (e.g., BitTorrent blocking, traffic shaping). M-Lab is helping build a common pool of network measurement data, removing the need for every research project to collect its own data and facilitating cross-sample analyses. All data collected through M-Lab is made publicly available and placed in the public domain.

The M-Lab platform uses a number of purpose-built and well-connected measurement servers in strategic locations around the globe. Currently, a total of 45 servers are operational across 15 geographically distributed sites in the United States and Europe. Each tool is allocated dedicated resources on the M-Lab platform to facilitate accurate measurements. Server-side tools are openly licensed to allow third-parties to develop their own client-side measurement software. Researchers and network scientists that are interested in running their tools on the M-Lab platform can contact M-Lab's steering committee, which coordinates research on the M-Lab platform. Once granted access, researchers can login and run their experiments on M-Lab servers. Using the M-LAB platform brings benefits to researchers, since it helps to:

- expose measurement tools and systems to a large number of users, since it is usually quite difficult to deploy a measurement tool to many highly available, well-connected servers around the world;
- validate analytical and simulation models with data from real-world Internet paths; instead of making assumptions about the available capacity, delays, losses, presence of traffic shapers, buffer sizes, etc., an analytical or simulation model can be grounded on measurements derived from the available M-Lab data;
- share and analyze datasets collected on M-Lab;
- avoid the administrative and operational overhead involved in managing a large-scale distributed server platform.

2.2.2 SamKnows

SamKnows [46] provides a broadband performance measurement platform based on special hardware devices called Whiteboxes. SamKnows Whiteboxes are consumer grade, home Wi-Fi routes with additional testing software integrated, that can be deployed onto the home network in order to test and report a range of metrics:

- Web browsing: the total time taken to fetch a page and all of its resources from a popular website;
- Video streaming: the initial time to buffer, the number of buffer under-runs and the total time for buffer delays;
- Voice over IP: upstream packet loss, downstream packet loss, upstream jitter, downstream jitter, round trip latency;
- Downstream speed/Upload speed: throughput in Megabits per second utilising three concurrent TCP connections;
- UDP latency: average round trip time of a series of randomly transmitted UDP packets;
- UDP packet loss: percentage of UDP packets lost from latency test;
- Consumption: volume of data downloaded and uploaded by the panellist;
- Availability: the total time the connection was deemed unavailable;

– DNS resolution: the time taken for the ISP's recursive DNS resolver to return an A record for a popular website domain name;

– ICMP latency: the round trip of five regularity spaced and schedule ICMP packets;

– ICMP packet loss: the percentage of packets lost in the ICMP latency test.

The Whitebox can operate in a two modes: operate as a router, replacing the user's existing Ethernet router (in this case, all wired and wireless devices should connect through the Whitebox) or operate as an Ethernet bridge, co-existing with an existing router (all wired devices should connect through the Whitebox, while the wireless devices should continue to connect to their existing routers).

The SamKnows Whiteboxes on panellists' home networks execute a series of software tests over their broadband Internet connection. A test cycle on the Whitebox occurs once an hour every hour (24 times per day). The timing of the testing is randomized per Whitebox. Once a testing cycle is complete, the results of these tests are reported securely up (over SSL) to a hosted backend infrastructure. Whitebox communicates with a backend via the special gateway, called the data collection service (DCS). The collected data is available to the end-users via the web-based reporting system.

2.2.3 RIPE Atlas

The RIPE Atlas project [47] is a distributed Internet measurement network consisting of thousands, potentially up to tens of thousand of measurements nodes, also called active probes, placed all around the Internet, all connected to a controlling framework. The main goal of RIPE Atlas is to take active measurements in a coordinated fashion, thereby supplying more measurement data for the benefit of the research community, and in general to the Internet community.

The vantage points in RIPE Atlas are called probes. They are tiny hardware devices capable of executing active measurements. They can be deployed anywhere: in residential settings, corporate networks and ISP infrastructures. Probes require virtually no configuration, and they execute their measurement tasks completely autonomously.

The probes are controlled by a hierarchical infrastructure. This infrastructure takes care of coordinating measurements, collecting and visualizing results, interacting with users, etc. Currently the RIPE Atlas measurement system executes built-in measurements, such as ICMP ping to pre-defined destinations (measuring the round trip time), traceroutes to these destinations, current uptime, uptime history and total uptime, DNS (anycast) measurements, where for each root name server, they try to determine which instance of the name servers the probe ends up connecting to. The measurement system takes care about scheduling such measurements, finding the appropriate probes to use, collecting and visualizing the results.

The measurement data generated by the system is recorded and kept as part of the RIPE NCC Internet measurement data set, which is available to researchers and other interested parties for further analysis. Furthermore, in the near future RIPE Atlas will provide the possibility to perform user defined network measurements.

2.3 Network Measurements for Mobile Platforms

Being able to run Internet measurement tests from mobile devices can bring additional benefits, to all interested participants such as network administrators, service providers, researchers or end-users. In our opinion there are several reasons which makes smartphones good candidates for running measurements on them:

1. Smartphones became extremely popular during last couple of years, and the number of devices that have been activated is increasing daily. For example, in March 2011, Apple Inc. reported [48] about more that 100 million iPhones have been sold in the world, and in November of 2011, the total number of Android-based devices (smartphones and tablets) that have been activated exceeded 200 million [49]. So the smartphones market is very developed and mature right now.

2. Deploying a special network measurement application on smartphone, turns it to some kind of a ”probing host” (like SamKnows Whitebox or RIPE Atlas probe). And since there is no need to deploy additional hardware components into the network (no need to produce and deliver them), the number of active probes might grow much faster than in hardware-oriented measurement infrastructures.

3. Since smartphones are mobile devices, now a ”probing host” becomes a mobile host, i.e., the smartphones change their geographical location during the time, which leads to a single device can perform active Internet measurements from different locations (or/and different Internet providers). This property is useful for network researchers, because they can collect measurement results which presents data from more geographical locations much faster. In this way smartphones potentially become are a very attractive infrastructure for performing Internet measurement tests and collecting obtained results.

4. Furthermore, most smartphone devices are able to determine their current geographical location using the GPS system, which is usually is not the case when tests are run from stationary desktops or laptops. This can bring additional benefits to broadband connectivity measurements, since now the actual geographical position can be assigned to a particular measurement test result.

5. Finally, the performance and policy management measurement tools can be useful to end-users to monitor and troubleshoot the current state of the mobile network.

2.3.1 Internet Measurement on Mobile Networks Overview

The existing network measurement tools for mobile platforms (Android, iPhone, Windows Phone OS) mostly allow only to obtain basic network information (e.g., local and global IP addresses of the mobile device) and to measure the network performance (e.g., downlink/uplink throughput, latency, round trip time).

The *Network Diagnostic Tool (NDT)* [50] is an application for Android-based smartphones for running network speed and diagnostic tests. An *NDT* test reports the upload and download speeds, in addition it also attempts to determine what, if any, problems limited these speeds, differentiating between computer configuration and network infrastructure problems. The *NDT* server collects test results, records the user's IP address, upload/download speed, packet headers and TCP variables of the test. Note that *Network Diagnostic Tool* is a part of the M-LAB Internet measurement platform.

The *WindRider* [51] application has been developed for the Windows mobile platform. It attempts to detect whether your mobile broadband provider is performing application or service specific differentiation, i.e., prioritizing or slowing traffic to certain websites, applications, or content. In addition, passive measurements are performed on the mobile device. The application measures the delays experienced by different web pages and records the explicit user feedback about different applications. *WindRider* can be installed on a mobile device such as a Pocket PC.

The *Fing* [52] is a multiplatform (Linux, Mac OS, Windows, Android, iPhone, iPod, iPad) toolkit for network management, which allows to perform service scans (TCP port scan), hosts availability detection, traceroute, MAC address and vendor gathering, TCP connection testing, DNS lookup.

MobiPerf [53] is a handy mobile network measurement tool designed to collect anonymous network measurement information directly from end users. It runs on Android and iOS devices and within 2-3 minutes, users are able to obtain basic network information (e.g., the device's IP address as seen by the server and the network type such as HSDPA), network performance information (e.g., downlink/uplink throughput in kbps) and network policies (e.g., testing which ports are blocked by the cellular ISPs).

The *RadioOpt Traffic Monitor* [54] measures data traffic consumed by your wifi and cellular interface. It also can measure download and upload throughput as well as ping durations via an integrated speed test (available for Android OS based devices).

J. Prokkola, P. Perälä and M. Hanski in [55] analyze the performance measurements (including one-way delay and jitter) in live 3G/HSPA networks by comparing TCP and UDP goodput performance in WCDMA, HSDPA-only, and HSPA mobile networks. Also they discuss the impact of different properties like speed, signal strength, handovers onto the network performance.

In [56] K. Pentikousis, M. Palola, M. Jurvansuu, and P. Pekka present the results of extensive experimentation and performance measurements with public WCDMA 3G/UMTS networks. In this paper the so-called ”first connection goodput phenomenon” is explained. This phenomenon refers to that the observed goodput of the first of a series of back-to-back transfers is consistently below par. In addition, authors pay attention on the precise bandwidth measurement in mobile networks. Particularly they conclude that high goodput rates are only achieved using large payloads.

In [57] a passive methodology for TCP performance evaluation over General Packet Radio Service (GPRS) networks is presented. This technique relies on traffic monitoring at the GPRS ingress/egress router interface. Based on the IP and TCP headers of the packets authors estimate the end-to-end performance of TCP connections such as connection setup behavior and data transfer performance.

2.3.2 Traffic Shaping Detection on Mobile Networks

In section 2.1.5 we discussed the measurement instruments to run application and policy measurements. In particular, several projects for traffic shaping detection were mentioned above. Let’s look at what the State of the Art for such kind of applications for mobile devices and mobile networks is.

In 2009, a group of researchers from the North Western University (Evanston and Chicago, Illinois, U.S.A.) have started a project called *WindRider* [51]. *WindRider* is an application for mobile devices that performs active and passive Internet measurements. One of the goal of this measurement tool is to detect whether your mobile broadband provider is performing application-specific or service-specific differentiation. By June 2012, the implementation for Windows Mobile 5 operation system was completed. However, this version of mobile OS is outdated now and it is not supported anymore. *WindRider* implementations for Apple IOS and Android OS were announced, but none of them have been finished yet. Moreover, the website of this project has not been updated for the last two years, and the current status of this project is unknown.

One of the most popular applications for traffic shaping detection is called *Glasnost* [3]. Using this application, you can test if your ISP is throttling or blocking different types of application-layer protocol traffic, like P2P protocols (including BitTorrent, eMule), HTTP traffic, SSH transfer, Flash video and others. *Glasnost* is based on a client-server architecture, where the client connects to a *Glasnost*-server to download and run various tests. Each test measures the path between the client and the server by generating flows that carry application-level data which are constructed to detect traffic differentiation along the path.

However, *Glasnost* is not applicable for running on smartphone devices, because:

– the *Glasnost*-client component is implemented using Java Applet technology, but unfortunately Apple IOS devices do not support running Java applets; and Android OS offers only limited applet support, which makes it impossible to run the *Glasnost*-client on Android-based smartphones;

– In addition, the *Glasnost* tests were not designed to use them on mobile networks. As a result, a single test execution might consume more than 100 MB of data traffic, which is completely unacceptable for mobile Internet users.

Other existing tools (mentioned in section 2.1.5) have different client ports for different operational systems (MAC OS X, Linux, FreeBSD, Windows). But none them has an implementation that can be run on modern mobile platforms (Android-based or iOS-based). So, currently there is no working application for smartphone devices that can perform traffic shaping detection. The only workaround to make such measurements on mobile networks, is to use a smartphone device as an Internet bridge, and run one of the mentioned applications on the connected workstation (or laptop).

Chapter 3

Implementation

In this chapter we will provide information about a tool developed in the frame of the master thesis project, which can be used for Android smartphones and helps to detect the deep-packet inspection based traffic shaping on mobile networks. We will start from the application architecture overview and the basic idea of the technique that helps to detect the traffic shaping along the path. Then we will talk about the client's pre- and post-test request processing, the format of the protocol description files which are used to define the measurement tests, the measurement test lifecycle and finally we will discuss the algorithm of statistical analysis of obtained measurement results.

3.1 Traffic Shaping Detection Tool Overview

As we have discussed in subsection 2.1.5, ISPs can differentiate between flows of different types by examining the IP headers, the transport protocol headers or the packet payloads. During this project, we created an application for Android OS that helps to detect the traffic shaping based on the deep packet inspection techniques. In other words, we assume that IPSs perform deep packet inspection to determine whether a flow carries some "unwanted" application protocol data. Detection of the presence of traffic differentiation based on examining IP headers or transport protocol headers is not considered in this thesis. Another limitation of our measurement tool, is that all communication between the server and the client components is done using stream sockets (implemented on top of the TCP protocol). Therefore, the application can detect traffic shaping of only those application protocols, which use TCP on the underlying transport layer.

The application is based on a client-server architecture (see Figure 3.1). A client connects to a measurement server and retrieves the list of available application protocols to test. The client can select one of these protocols and download a corresponding application protocol description file from the server side. Each

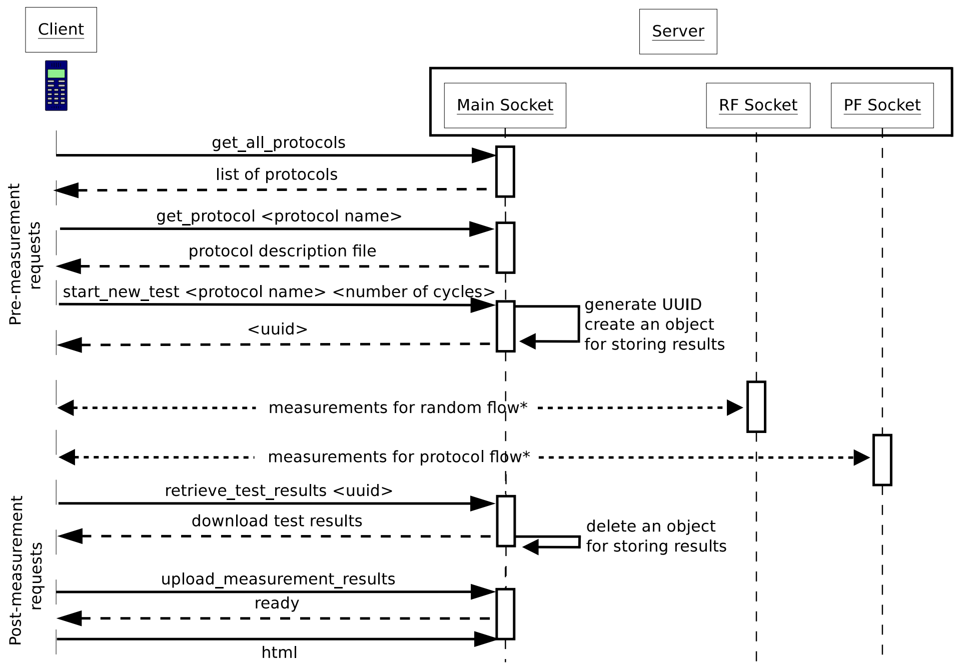

Client
Server
Main Socket
RF Socket
PF Socket
Pre-measurement requests
get_all_protocols
list of protocols
get_protocol <protocol name>
protocol description file
start_new_test <protocol name> <number of cycles>
<uuid>
generate UUID create an object for storing results
measurements for random flow*
measurements for protocol flow*
Post-measurement requests
retrieve_test_results <uuid>
download test results
delete an object for storing results
upload_measurement_results
ready
html

Figure 3.1: Traffic shaping detection tool lifecycle overview (*this sequence diagram does not present the lifecycle of measurements for protocol and random flows, refer to section 3.4 for more details).

description file provides a set of rules, which define the client's and the server's behavior during the measurement test (please, refer to section 3.2 for more details). Once the client downloads a particular protocol description file, it can start a measurement test that checks whether current ISP deploys some traffic differentiation policy for the selected application protocol or not. We do not want to collect any data about the measurements without client's permission, therefore all measurement test results are stored locally on the client's smartphone. Users can submit measurement results to the server side manually. The pre-measurement test and the post-measurement test client requests processing is discussed in Section 3.3. The measurement test execution details are presented in Section 3.4.

The core idea behind the traffic shaping detection measurement test is the emulation of a pair of flows that are identical except in one respect that should trigger traffic differentiation along the path. In the context of this paper, when we talk about the flow, we mean the sequence of packets that are exchanged between the server and the client sides in both directions within the same TCP connection. The performance of the flow implies the application goodput during the lifetime of this flow. According to the goodput definition, this is the number of useful information bits, delivered by the network to a certain destination, per unit of time [58]. We distinguish two components: downlink performance and uplink performance. The first component denotes the application goodput in download (from the server to the client) direction, and the second component denotes the application goodput in upload (from the client to the server) direction respectively. Measuring and comparing the performance of those two flows helps to determine whether content-based traffic shaping methods were applied or not. Since my application for traffic shaping detection injects custom packets into the network during the measurements it can be classified as an active measurement tool.

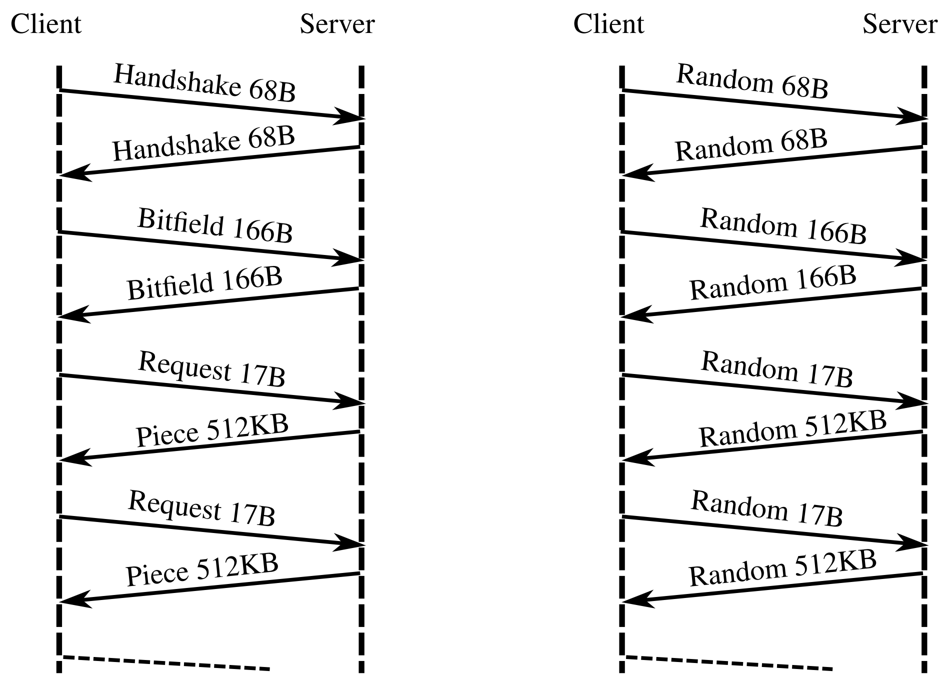
Let's consider an example of constructing a pair of flows that can help to reveal the presence of traffic shaping of a BitTorrent [59] application protocol along the path (see Figure 3.2). The left figure corresponds to the first flow. The client opens a TCP connection to the measurement server (the Figure 3.2 represents only the application layer protocol messages, hence the TCP handshake is not shown) and starts sending packets that implement the BitTorrent protocol. In this case, the payload of the packets carry BitTorrent protocol headers and content. The server in its turn responds with packets that conform to the BitTorrent specification.

The packet exchange on the right figure corresponds to the second flow. Now, the client opens another TCP connection and sends the same packets, but in this case the payload contains randomly generated data. Note that packets preserve their sizes as in the first flow.

The two flows traverse the same network path and have the same network-level characteristics. As a result an Internet Service Provider that differentiates BitTorrent traffic would impact only on the first flow, and keep the second flow's performance untouched. Thus, significant differences in those two flows' performances are likely to be caused by the traffic manipulation along the path. If the ISP completely blocks BitTorrent traffic, it also would be noticed because one of the participating sides (client, server or both of them) will receive a socket time-out. And the flow with random data will successfully finish the packets exchanging measurement cycle.

The presented traffic shaping detection technique is similar to that used in the Glasnost [3] project. The benefit of the proposed technique is that we are running an active measurement test. This implies that we totally control the measurement test lifecycle (we can repeat flows with different properties like payloads or port numbers). As a drawback, we need to generate and inject additional data into the network, which in case of mobile networks (EDGE, HSPA) is still relatively expensive. Additionally, the active measurement tests allow to make a conclusion about the presence of traffic differentiation along the path without involving other users into a measurement process. Unlike the projects based on the passive detection techniques (e.g., NANO [5]), which require making observations from many end-hosts.

DiffProbe [8] also uses an active probing method in order to detect whether an ISP is deploying forwarding mechanisms such as priority scheduling, variations of Weighted Fair Queuing (WFQ) or Weighted Random Early Detection (WRED) to discriminate against some of its customer flows. Like in our proposed method, DiffProbe also traverse the network with two separate flows: an Application flow and a Probing flow. The difference is that instead of measuring network bandwidth,



a) BitTorrent Flow b) Random Flow

Figure 3.2: A pair of flows used to detect BiTorrent traffic differentiation along the path.

DiffProbe compares delays and packet losses experienced by these two flows. In this way, DiffProbe complements the proposed traffic shaping detection method as it can detect differentiation that leads to a small increase in latency. However, as soon as active queue management techniques, used by ISP, affect on the application goodput, we can also detect this type of traffic differentiation.

3.2 Protocol Description Files Format

In this section we discuss the format of so-called protocol description files that are used by the server and the client in order to construct the flows that carry headers and payloads of a particular application protocol.

Only flows that carry packets which conform to an application protocol that we want to test will trigger the traffic shaping (if shaping policy for particular protocol exists). Therefore, a set of the rules that define protocol headers and payloads must be defined for every application protocol for which we would like to be able to run measurement tests.

First, we had an idea to replay network traffic for corresponding application protocols previously captured with Wireshark [60] and stored in *pcap* [61] files. However, during the development phase, we faced the problem of reading pcap files on Android smartphones. There is no working library available for parsing pcap formatted files for Android OS. The only solution found was a *Jnetpcap for Android* library [62], which is still in an early development phase and non-optimized. This open-source library is able to read pcap files, though its performance does not allow to use this library in a real application, because of the slow data parsing speed. For example parsing a small pcap capture file with only 6 records (each record less than 2 kB) on a HTC Desire S smartphone (CPU 1GHz, RAM 768 MB, Android OS version 2.3.5) took approximately 5 seconds. Fixing performance problems, as well as creating our own pcap-parser solution for Android OS, would have taken too much time. So, we had to abandon the idea of using pcap formatted files in our traffic shaping detection tool.

As a result, we decided to create a so-called protocol description file for every application protocol for which we would have liked to provide measurement tests. The proposed protocol description file format is very similar to the format used in the *Glasnost* [3] application. The reason why we did not use the *Glasnost* format and developed our new own format was that the measurement test lifecycles are different in these two projects. The *Glasnost* protocols description file contains commands and parameters that are redundant in our application, and vise versa some of command that we need are not presented there.

Each protocol description file defines the name of the application protocol, port numbers that should be used during the measurement tests, well-known port numbers that might be officially or unofficially assigned to a particular application protocol, and a sequence of commands that tells how to construct a payload which conforms to a protocol specification.

Let's look at the example of the protocol description file that corresponds to a HTTP protocol.

```
protocol HTTP
PFport 30008
RFport 31008

request string("GET /wiki/Jacobs_University_Bremen HTTP/
1.1") byte(13) byte(10) string("Host:  en.wikipedia.org")
byte(13) byte(10) string("User-Agent:  Mozilla/5.0 (Linux;
U; Android 2.3.5; en-de; HTC Desire S Build/GRJ90) AppleWebKit/
533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1")
byte(13) byte(10) string("Accept:  text/html") byte(13)
byte(10) string("Connection:  close")
byte(13) byte(10) byte(13) byte(10)

response string("HTTP/1.1 200 OK") byte(13) byte(10)
string("Server:  Apache") byte(13) byte(10) string("Content
-Language:  en") byte(13) byte(10) string("Content-type:
text/html; charset=utf-8") byte(13) byte(10) string("Content
-Length:  20") byte(13) byte(10) byte(13) byte(10)
string("12345678901234567890")
```

The first line defines an application protocol name which is visible to an end user. The protocol name might be an arbitrary string value placed right after a keyword *protocol* (separated by a space character). Note, that the application protocol name is a mandatory field and it must be unique for each protocol description file. In addition it must not contain space characters.

Next two following properties (*PFport* and *RFport*) define the TCP port numbers which are used to communicate with a server during the measurement tests. The destination port number defined by the *PFport* field is used by a client to establish a TCP connection for sending a flow that carries application protocol data (so-called ”protocol flow”). The *RFport* field defines the destination port number used to send the flow with randomly generated data (so-called ”random flow”). The requirements for *PFport* and *RFport* values are the following: these values are mandatory parameters and they must be unique for each protocol description file, and these port numbers should not be registered to any application protocol in order to avoid the possible traffic shaping from an ISP. The list of Internet socket port numbers used by different application protocols can be found here [63].

Finally, the protocol description file contains a set of *request* and *response* instructions that define how to build the application protocol headers and payloads. In the example above, the *request* command tells a sender (might be a client or server component depending in which direction we are measuring the path at the moment) to send an HTTP request to retrieve the page about the Jacobs University Bremen from Wikipedia. The responder side sends back an HTTP 200 OK response that contains 20 bytes of payload. The following functions can be used in a protocol description file to construct the application protocol header and payload:

– `string(argument)` function appends to a message buffer a sequence of bytes that encode a given string argument;

– `byte(value)` function appends to a message buffer given byte value (this argument must be in range from 0 to 255);

– `repbyte(value, N)` function (is not presented in the example above) appends given byte value to a message buffer exactly N times (N must be a positive integer value).

The protocol description file might contain more than one pair of *request* – *response* commands. The only constraint is that for every *request*, exactly one *response* must be defined immediately after the *request* definition. The commands defined in a protocol description file are separated by a newline character (empty lines are ignored by a parser).

Since the same protocol description file is used by both components, the server and the client sides can generate a flow that carries packets that implement some application protocol (a destination port number defined by *PFport* value is used). For detecting content-based differentiation, a new TCP connection is established (*RFport* number is used), and the server and the client generate a flow that carries randomly generated data (while the size of the messages is preserved). Thereby, only the protocol flow might be affected by the traffic shaping mechanism.

The protocol description files for different application protocols are stored in a server's file system. Each protocol description is parsed by a server at startup time. The client can download and parse the content of any description file, right before running a corresponding measurement test.

3.3 Client Requests Processing

In this section we describe the protocol used for communication between the client and the server. For processing pre-measurement and post-measurement requests from the different clients, the server component opens a TCP server socket (hereinafter referred to as ”Main socket”) on the port number which is known to all clients. The main socket is responsible for processing the client demands in order to prepare the client for measurement test execution, and for handling post-execution requests. The main socket is able to handle the following client requests:

– get a list of available application protocol;

– get a protocol description file of the specific application protocol;

– initiate a traffic shaping measurement test for the selected application protocol;

– retrieve a downlink measurement test results from the server side;

– submit a measurement test results to the server side;

Let's look at the client request processing in more details:

a) Retrieving a list of available application protocols to test and downloading the specific protocol description file.

All the protocol description files are stored in a server's file system. They are parsed by a server at startup time. We do not store these protocol description files on the client, since by changing even a single protocol description file on the server side, we would force end-users to update their client applications as well. The different versions of the same protocol description file on the server and the client sides would make it impossible to run the corresponding traffic shaping measurement test. Thereby, the client should be able to retrieve the list of supported application protocols, and download the corresponding protocol description

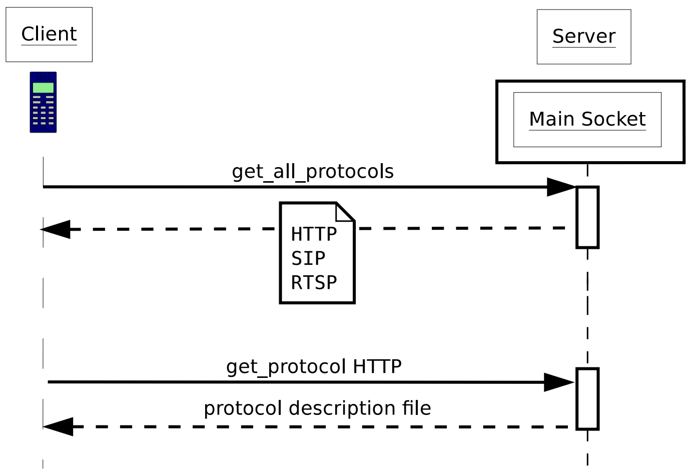


Figure 3.3: The client retrieves the list of available application protocols and downloads the HTTP protocol description file.

file right before the test execution from the server.

To learn which application protocols are available for running traffic shaping measurement tests, the client established a TCP connection with the server's main socket and sends a message that contains only **get_all_protocols\r\n** string value. Once this request is received by the server, it responds with a message that contains a list of available application protocol names separated by the "**\r\n**" delimiter (see Figure 3.3). The **end_of_message\r\n** string indicates the end of the response message.

Since the application protocol name is unique value, it can be used as an identifier when the client wants to download the specific protocol description file. For this purposes a client sends a message that contains the string value of the following format: **get_protocol <protocol name>\r\n**. Where the **<protocol name>** is a corresponding application protocol name. When the server receives that command it sends back to the client the content of the corresponding protocol description file (see Figure 3.3). Which is parsed by a client's application later. The **end_of_message\r\n** string indicates the end of protocol description file. If the server can not find the protocol description file that corresponds to the application protocol name provided in the client's request, it sends back a message with a **get_protocol_failed\r\n** string.

b) Initiate the traffic shaping measurement test for the selected application protocol.

The main socket is also used to accept the client requests when the user wants to start the new traffic shaping measurement test for a certain application protocol. In order to initiate new measurement test the client sends a message that contains

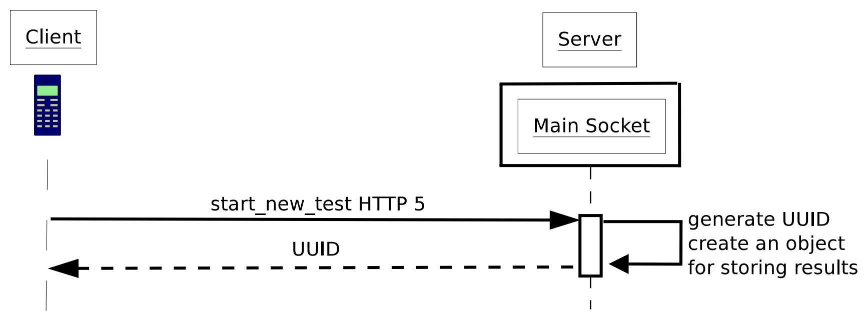


Figure 3.4: The client initiates a new measurement test for an HTTP application protocol with 5 measurement cycles.

the string of the following format: **start_new_test** <**protocol name**> <**number of cycles**>\r\n, where the <**protocol name**> is a corresponding application protocol name, and the <**number of cycles**> represents the total number of measurement tests to run in a row.

A single measurement test execution usually is not enough to make an unambiguous conclusion about the presence of traffic shaping along the path. So, it is highly recommended to repeat the same test several times to increase the quality of measurements. Once the server receives the command, it creates an object for storing measurement test results, generates and assigns a universally unique identifier (UUID) to that object, and sends back the string representation of the UUID value to the client application (see Figure 3.4).

Now, the client can establish new TCP connections with the measurement server (using the *PFport* and *RFport* destination port numbers specified in the corresponding protocol description file) in order to run random and protocol flows. The previously obtained UUID is used to identify the measurement test instance on the server side.

c) Retrieve the measurement test results from the server side.

According to the application's architecture, once the measurement test is done, the measurement test results for an upload direction are stored on the client, and the results for a download direction on the server. So, to collect and merge measurement results of the same test, the client can send a message of the following format: **retrieve_test_results** <**uuid**>\r\n, where the <**uuid**> is the string representation of previously obtained UUID of a measurement test. The server responds with the measurement test results for a download direction. The response message contains a plain text composed from the multiple text blocks of the following format:

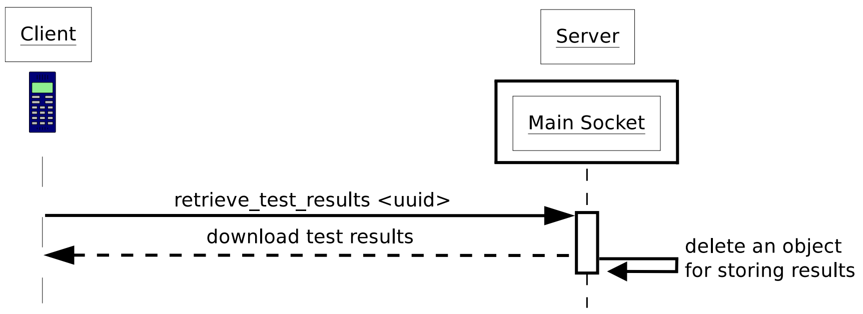


Figure 3.5: The client requests the downlink measurement test results.

```
Cycle <sequence number>
Protocol
<message size> <RTT>
...
<message size> <RTT>
Random
<message size> <RTT>
...
<message size> <RTT>
End Cycle
```

where the <**sequence number**> defines the number of a measurement cycle for which the results are presented, the <**message size**> value represents the number of bytes in a message that was used to traverse the path during the test (see Section 3.4 for details) and the <**RTT**> value is a corresponding measured round trip time (in milliseconds) of this message. The keywords **Protocol** and **Random** are used to distinguish the measurement results between the random and the protocol flows. The number of these blocks is equal to the number of measurement cycles in a test. The **end_of_message\r\n** string indicates the end of the response message.

If some error occures during the test, or server side can not find the results for this test, it replies with a message that contains the **test_results_not_found\r\n** string value to notify client that this operation failed. Once the data have been transmitted the server deletes the instance of the object that stores measurement test results for the corresponding measurement test UUID.

d) Submit measurement test results to the server.

If the user wants to share the measurement test results with us, he can push the report file to the measurement server. The measurement test results are stored in an HTML file. They present the measured link goodput values in the upload and the download directions for the random and the protocol flows in a table format.

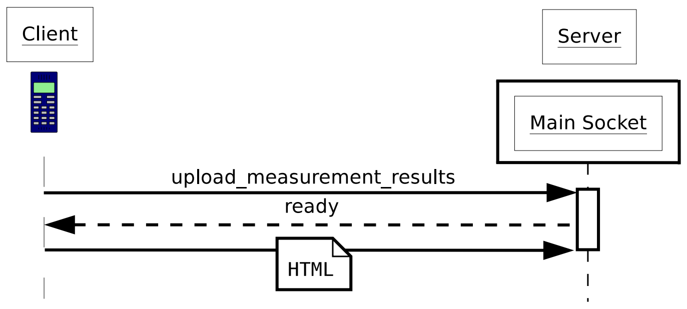


Figure 3.6: The client submits the measurement test results HTML file to the server.

The HTML file is the most convenient way to store the measurement test results on the client, since it can be viewed by any browser installed on an Android smartphone (in contrast, most Android browsers do not render XML files properly).

In order to push the report file, the client sends a message to the main socket that carries the **upload_measurement_results\r\n** string value, and waits for the response message with the **ready\r\n** string value. After that, the content of a HTML report file is transmitted to the server side, where it is stored on the server's file system. The **end_of_message\r\n** string indicates the end of an HTML file content. Once the file transmission is completed the client closes the TCP connection. If the "**ready\r\n**" message has not been received within 3 seconds (default timeout value), the client assumes that server is busy right now and it aborts the upload results request.

3.4 Measurement Test Lifecycle

The traffic shaping detection technique has been described in Section 3.1. In the current section, the implementation details of this technique are presented.

As it was mentioned above, to detect the traffic differentiation, we need to emulate a pair of flows that are identical except in one respect that should trigger traffic differentiation along the path. By measuring and comparing performance of these two flows, we can make a conclusion about the presence of a content-driven traffic manipulation from the Internet Service Provider's side. So the core task during the measurement test is to determine the application goodput in both (upload and download) directions for "protocol" and "random" flows.

The application goodput estimation is based on a simple idea. One of the participants, for example, the client side, measures the current local time (called "start-time") and sends a message to the server (see Figure 3.7). Both components know the size of this message, and once server receives the whole message, it immediately responds with a short `"OK"` notification to acknowledge the client that initial message has been delivered successfully. Now, the client measures the local time (called "end-time") when the `"OK"` notification is delivered. Since we know the

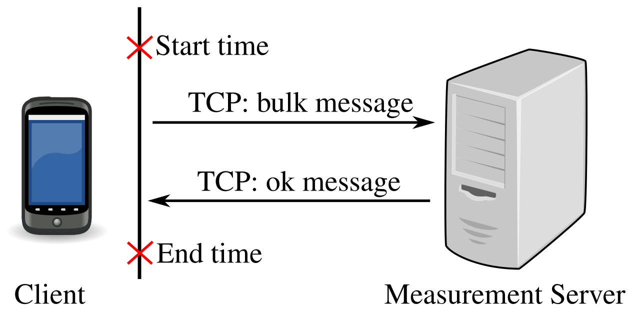


Figure 3.7: Scheme of uplink bandwidth estimation

number of bytes being transmitted, start and end measurement time values, we can estimate the goodput in upload direction using the following formula:

```
Goodput = Number of bytes/(start time - end time)
```

This calculated goodput value slightly differs from the actual one, because we are measuring the round trip time. However, we do not need to calculate the goodput very precisely. More important is to know the ratio of estimated performances for ”random” and ”protocol” flows.

The group of researchers in [55] showed that the near-nominal application throughput (and hence the goodput) in mobile networks is realizable for large payloads only. So, the size of the message used for probing the network (hereinafter reffed to as ”bulk message”) should be large enough to fully utilize the allocated bandwidth. This value is different for various mobile networks, and it depends on the network type and on the network configuration. In order to be able to measure the goodput on different mobile networks, we send a train of bulk messages. We start with a relatively small 2kB message and gradually increase the bulk message size if it is necessary. At each step the bulk message RTT value is measured. The decision about increasing the bulk message size or terminating the current goodput estimation round is made using the following rule:

```
if (bulk message $RTT \leq MAX\_RTT$) then
    increase the bulk message size and send it again
end
else
    stop current measurement round
end
```

In the traffic shaping detection tool configuration the MAX_RTT value of 2 seconds is used. This value was empirically determined during the series of tests on the mobile networks. As it turned out, the bulk message with a 2 seconds RTT value is enough to fully utilize the allocated network bandwidth regardless the network type. Increasing the bulk message size after that value does not make sense anymore, since the estimated goodput value remains the same.

In the current application configuration the bulk message size changes according the following scheme:

```
2kB -> 4kB -> 8kB -> 16kB -> 32kB -> 64kB -> 128kB ->
-> 256kB -> 512kB -> 1MB -> 2MB -> 4MB -> 8MB
```

In practice, the goodput estimation round usually terminates when the bulk message size reaches the 1MB or 2MB value (in case of HSPA mobile networks). The bulk message size of 4MB is used in rare cases when the mobile network bandwidth value is higher than 5 Mbps. The 8MB message size is never used in practice (modern HSPA mobile networks have a limit up to 7.2 Mbps) and it is reserved for future.

Using the proposed measurement technique the client side can estimate the application goodput on upload direction. And vice versa, to measure the goodput on download direction the server sends a train of bulk messages to the client.

As it was described in section 3.3, when a user wants to start a new test, first he establishes a new TCP connection with the main socket and sends the **start new test** <**protocol name**> <**number of cycles**> command. Once a server receives this command it creates an object to store measurement results, and generates the UUID for the measurement test session (see Algorithm 1). This UUID value later is used to identify the test instance on a server side while handing clients requests.

For every application protocol that is supported, two server TCP sockets are opened and bound to port numbers specified in a corresponding protocol description file (*PFport* and *RFport*). The first socket is used to accept client connections which correspond to ”protocol” flows, and the second server socket handles connections that correspond to ”random” flows during the measurement test. Both sockets are configured to infinitely listen to new connections made to these sockets and accept them. Once a new client connection has been accepted, the server component creates a new thread to handle this connection (see Algorithm 1). The accepted client socket is passed as an argument to that thread. Hence, server sockets are prevented from being blocked, and the server component is able to support processing multiple client connections.

A single measurement test execution usually is not enough to make an unambiguous conclusion about the presence of traffic shaping along the path. So, it is highly recommended to repeat the same test several times, to improve the quality of measurements. Each set of goodput measurements for a single flow in both directions is called a measurement cycle. In order to neutralize the impact of the previous measurements, the client establishes a new TCP connection with a corresponding server socket (for random or protocol flow) for each measurement cycle (see Algorithm 2). After a new connection has been established, the client and the server send to each other the set of ”request-response” messages (hereinafter reffed to as ”protocol messages”) defined in the corresponding protocol description file. Both sides know the size, the number of protocol messages and their order.

Once the server receives the ”protocol request” message it immediately sends back the corresponding ”protocol response” message. During sending the protocol messages, the server and the client calculate the number of received bytes in order to determine the end of a message. This is done to preserve a format of application protocol which is tested. Otherwise, if we append custom ”end of the message” characters there is a risk that an ISP would not match the application flow.

These protocol messages are used to trigger the traffic differentiation along the path. While the content of bulk messages is generated randomly for both (random and protocol) flows. There is no need to make the bulk messages conform to the tested application protocol. The deep packet inspection tools do not sniff every packet within a flow, because it would be a huge waste of clock cycles. In addition it would be useless for matching most packets, which are likely to consist of data without any application protocol ”identifiers” (for example the middle of some file for P2P traffic). The bulk messages are used only for goodput estimation. We need to inject the protocol messages at the beginning of a new TCP connection, since some deep packet inspection tools (e.g. L7-filter [64]) make a decision about the type of the flow by looking at the first several packets only. In the case when we run a random flow, the server and the client also inject protocol messages, but in this case, the protocol messages content is generated randomly, while the size of the messages defined by the protocol description file is preserved.

Once all protocol messages defined in a protocol description file have been sent, the client sends the information that helps the server to identify the measurement cycle (e.g., the string representation of the test UUID of the measurement test and the measurement cycle number separated by a space character. The **\r\n** characters indicate the end of the message). After the server received the UUID and the measurement cycle number it responds with a message that contains the **uuid_received\r\n** string.

After that the client and the server perform the goodput measurements for the upload and the download directions (see Algorithm 3) using algorithm described above. The protocol messages are injected every time right before the bulk message is about to be sent. This is done to ensure that the deep packet inspection tools, which might continue sniffing the packets, could still match the flow. When the measured RTT value becomes larger than MAX_RTT threshold, or the bulk message size reaches the maximum value, the client (or the server depends on which direction we are measuring the goodput) sends a **terminate_message\r\n** string to notify the server that the measurements in a particular direction cycle are done.

After the entire measurement test has been completed, the upload goodput results are stored on the client side, and the measured download goodput results – on the server side. To retrieve the measurement results from the server, the client sends a **retrieve_test_results <uuid>** command to the main socket.

Cycle	2 kB	4 kB	8 kB	16 kB	32kB	64 kB	128 kB	256 kB
1	115	269	306	320	325	333	357	-
2	115	293	306	320	333	323	320	-
3	134	292	321	321	325	288	314	-
4	108	179	268	292	309	287	305	-
5	161	292	292	285	247	295	308	-

Table 3.1: Download performance values (in kbps) for Random flow on the HSPA network of the Congstar mobile operator.

3.5 Measurement Results Data Analysis

This section contains the information about how to interpret the measurement test results, and describes the algorithm that helps to make a decision about the presence of traffic shaping along the path.

The measurement test results are presented to the end user in the form of tables, which contain the measured goodput values (see Table 3.1). There are four tables: one for each (protocol and random) flow in each (upload and download) direction. The goodput values corresponding to the biggest bulk message size within a measurement cycle represent the value closest to the actual goodput value (e.g., in a Table 3.1 the estimated goodput value for the measurement cycle number 1 is a 357 kpbs). By comparing the estimated goodput values for the protocol flow and the random flow in the same direction, we can make a decision about the presence of the traffic shaping along the path.

The ISPs can shape the application protocol performance not only by limiting the application goodput. They could also terminate the TCP connection by sending a TCP FIN or TCP RST packet to the client (or server), or drop the packets with a 100% drop rate. The traffic shaping detection tool is able to detect these types of traffic differentiations. In the first scenario, the detection tool would notice that the socket was unexpectedly closed during the test, and it would be shown in the output table as a `"connection_reset"` record in a corresponding measurement cycle row. In the second case, the traffic detection tool would receive the `SocketTimeoutException` during the test execution. And this also would be displayed in the output table as a `"timeout"` record (see Table 3.2). Thus, if the output tables for Protocol flows have many `"connection_reset"` or/and `"timeout"` records and the corresponding tables for Random flows do not have them (or have only few of them), then it is fairly likely that the provider does traffic shaping for an application protocol that we have been tested.

In order to help the user, the application automatically makes a decision about the presence of traffic shaping along the path using the statistical methods (implementation is done on the client component in the `de.jacobs.university.cnds.bonafide.utils.ResultAnalyzer` class). The *ResultAnalyzer* can return one out of the five possible decisions: we observe the traffic shaping, the traffic shaping most probably exist, the traffic shaping most probably do not exist, we do not observe the traffic shaping and we can not rely on provided data.

At the first step, we analyze the measurement results by the completeness criterion. For that purposes, we calculate so-called `fail_ratio` for both flows, which determine the ratio of the measurement cycles that exited with a `"connection_reset` or a `"timeout"` exception. If the ratio for a random flow is less than 20% and the ratio for a protocol flow is more than 70% we conclude that application protocol performance has been affected by a traffic shaping along the path, otherwise we continue the data analysis.

Cycle	**2 kB**	**4 kB**	**8 kB**
1	17	43	`timeout`
2	19	44	`timeout`
3	18	45	`timeout`
4	17	44	63

Table 3.2: Example of measurement test with `fail_ratio` of 75%

Next, we formulate the null hypothesis that the measured goodput values for the random and the protocol flows in the same direction are close to each other, so the traffic shaping along the path is not observed. To check this null hypothesis, the `ResultAnalyzer` uses the *Mann–Whitney U* statistical test [65]. According to that method, we combine the goodput values for the two flows into a single set, sort this set in ascending order and assign the rank to each goodput value according to its position in a sorted set. Then we calculate the sum of goodput ranks for each flow separately and determine the greater (T_x) value. The *Mann–Whitney U* value can be calculated using the following formula:

$$U = n_1 \cdot n_2 + \frac{n_x \cdot (n_x + 1)}{2} - T_x;$$

where the n_1, n_2 are numbers of measured goodput values for protocol flow and random flow respectively, and the n_x is a number of measured goodput values in a flow with a higher rank T_x.

Finally we compare the calculated U value with a corresponding $U_{critical}$ value from the predefined *Mann-Whitney* critical values table (with a level of confidence 95%) [66]. If the $U > U_{critical}$ then we can accept our null hypothesis, which means that there is no traffic shaping along the path.

Additionally, we calculate the confidence interval for both flows which can be defined by the following segment:

$$[T_{mean} - St(n) \cdot \frac{\sigma}{\sqrt{n}}; T_{mean} + St(n) \cdot \frac{\sigma}{\sqrt{n}}];$$

where T_{mean} is a mean goodput value for a corresponding flow, n is a number of measured goodput values for a flow, $St(n)$ is a corresponding Student's-t distribution coefficient for an one-sided critical regions with a confidence of 95%, and σ is a standard deviation:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(T_{mean}-T_i)^2}{n-1}}$$

where T_i is a measured goodput value during the i^{th} measurement cycle. All goodput values that are out of the confidence interval are thrown away, and confidence interval is re-calculated again (including the mean value). This process repeats until all of the remaining goodput values lays inside the confidence interval. After that step, the measurement results are cleaned from the distorted measurements. Thereby, we can compare the bounds of the corresponding confidence intervals, the mean goodput values and the maximum observed goodputs for both flows in order to make a decision about the presence of the traffic shaping along the path (see Algorithm 4).

These two statistical methods add to each other. The $U \leq U_{critical}$ does not mean that the tested application protocol has been shaped. For example, the *Mann-Whitney U* test would return the false positive decision about the presence of traffic shaping on the following sets of goodput values:

```
Protocol flow:  1000, 1001, 1002, 1003, 1004 (kbps)
Random flow:  1005, 1006, 1007, 1008, 1009 (kbps)
```

The maximum goodput value for the protocol flow is less than minimum goodput value for the random flow (i.e., $U = 0$). However, all goodput values stay on the same level, and the traffic shaping methods definitely were not applied in this case. The correct decision in this case can be made using the confidence intervals method.

And vice versa, the confidence intervals method would make a false positive decision in case of measurement results with a lot of distortions[1]:

```
Protocol flow:  25, 28, 32, 330, 338 (kbps)
Random flow:  27, 315, 30, 332, 325 (kbps)
```

The mean value and the confidence interval for the Random flow are 324 and [317;330] kbps. Which is greater than values for the Protocol flow: 28 and [23;32] kbps. However, the *Mann-Whitney U* value is 12 which gives us a correct decision: NO SHAPING.

Mobile network performance might not be very stable during the measurement test. The network performance can change significantly multiple times during a relatively short time frame. This is especially noticeably on low-speed networks like EDGE. Two factors have an impact on the network performance: the signal strength and the load of the base station which the mobile device is connected to. Unfortunately, we do not have an access to the information about the base-

[1]This is a real example from experiments on the Congstar mobile network for the RTSP protocol.

station load at the moment. But the client component tracks all mobile network state changes and signal strength changes during the test. Therefore, the lack of signal or network discretions indicates that the obtained measurement results are not reliable anymore.

Chapter 4

Experiments Design

In this chapter, we will focus on the design of experiments that we made using the developed traffic shaping detection tool. First, we will explain which application protocols we would like to test during the traffic shaping detection experiments on mobile networks, and clarify why exactly these protocols have been chosen. Next, we will describe the experimental setup used for application correctness verification. Finally, we will describe the design of experiments made on the mobile networks.

4.1 List of Available Protocol Description Files

Currently, the following protocol description files are available to download and to run measurement tests: *HTTP*, *FlashVideo (Youtube)*, *SIP*, *RTSP*, *BitTorrent* and *VoIP H323*. In our opinion, being able to run traffic shaping detection tests for these application protocols might be interesting from the smartphone users' perspective.

We can speculate that mobile network operators might want to manipulate on the VoIP applications traffic performance. VoIP applications for smartphone devices actually are competitors for mobile service operators. To reduce financial losses it is tempting to restrict the performance of traffic flows that carry VoIP data.

The *VoIP H323* [29] standard addresses call signaling and control, multimedia transport and control, and bandwidth control for point-to-point and multi-point conferences. It is widely deployed worldwide by service providers and enterprises for both voice and video services over IP networks [67].

The Session Initiation Protocol (SIP) is a signaling protocol for controlling voice and video streams over IP networks. This application protocol is designed to be independent from the Transport Layer protocols, therefore it can be used over the TCP [68]. There is a number of popular SIP clients for Android-powered smartphones that allow to make VoIP calls over the Internet (*SIPDroid*, *Linphone*, *Fritz!App*, *CsipSimple*, etc.).

Since these H323 and SIP protocols are widely used in VoIP applications we think it would be interesting to run a series of traffic measurement tests against them. Even despite the fact that packets that conform SIP protocol carry only low volume payloads, the mobile operators might want to completely block or limit the performance of SIP flows.

According to the Google Play statistics [69][70] the BitTorrent client applications are much more popular than other Peer-to-Peer (P2P) client applications (like eMule or Gnutella). It is highly unlikely that the end-users will use the P2P clients on the mobile networks, because mobile networks are more expensive comparatively to Wi-Fi networks. However, many mobile Internet providers (e.g. *NettoKOM*, *O2*, *Congstar*) claim [11][12][13] that they do not support P2P traffic on their networks. It is interesting to check whether they perform deep packet based traffic shaping for the BitTorrent application protocol.

The Real Time Streaming Protocol (RTSP) [71] is a network control protocol designed for use in entertainment and communications systems to control streaming media servers. The protocol is used for establishing and controlling media sessions between end points. Like HTTP, RTSP uses TCP to maintain an end-to-end connection [72]. Some popular applications (Winamp, Spotify, VLC media player) use the RTSP for controlling the audio streaming.

Finally, we created protocol description files for the HTTP and the Flashvideo protocols. In our opinion, it is interesting to run an experiment for these application protocols, since according to the Cisco Visual Networking Index (VNI) global mobile data traffic forecast [73] the mobile video traffic and the web traffic represent more than 90% overall mobile traffic.

4.2 Mesurement Tool Correctness Verification Experimental Setup

To ensure that the developed traffic shaping detection tool is actually able to detect the content-based traffic differentiation along the path, we ran a series of measurement tests on an experimental infrastructure (see Figure 4.1). In this experimental setup all generated traffic goes through the Linux machine, which plays the role of a router. The router is configured to perform deep packet inspection and throttling flows' performance according to predefined set of rules. Since we know how the performance of flows that carry different types of data should look like, we can analyze results obtained during the measurement tests, and verify whether traffic shaping detection actually works properly or not.

The OpenVPN [74] open source package that implements VPN techniques has been installed onto a Linux machine for creating a point-to-point client-server connection. In this scenario all traffic from the client to the server and vice versa goes through the router. Thereby the router can manipulate the flows' performance.

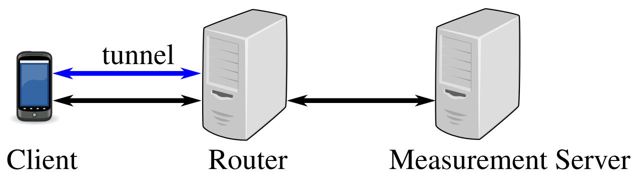


Figure 4.1: Experimental Setup Overview

To apply some traffic shaping rules on the router, we need to enable the router to classify the traffic. For this purpose L7-filter [64] [75] classifier software has been installed.

L7-filter is a software package providing a classifier for Linux's Netfilter subsystem, which can categorize flows based on the application layer data. Unlike most other classifiers, it does not just look at simple values such as port numbers. Instead, it does regular expression matching on the application layer data to determine what protocols are being used. L7-filter can be used in different scenarios like when you need to match any protocol that uses unpredictable ports (e.g., peer-to peer file sharing), to match traffic on non-standard ports (e.g., web traffic on port 1111), to distinguish between protocols which share a port (e.g., peer-to-peer file sharing that uses port 80).

There are two versions for this software. The first is implemented as a kernel module for Linux 2.4 and 2.6. The second experimental version runs as a user-space program and relies on Netfilter's user-space libraries for the classification process. In our experiments, we used the user-space version of L7-filter, which is still in the early stages of development, but is relatively easy to install compared to the kernel-version, and detects a larger number of protocols. All versions of L7-filter have been released under the GNU General Public License.

Both versions of L7-filter use regular expressions (though the user-space and kernel modules use different regular expression libraries) to identify the network protocol. The basic idea of L7-filter is to use regular expression matching of first several IP packets application layer data per flow to determine what protocol is being used. If the data matches regular expression for some traffic type, then the entire flow is marked as "matched" [76]. Running a regular expression matcher on every packet does not make any sense. Not only because it would be a huge waste of clock cycles, but it would be useless for matching most packets, which are likely to consist of data without any protocol "identifier" (the middle of some file for P2P traffic for example).

Another benefit of using L7-filter for our testing purposes is that it provides a large number of well tested regular expressions for different protocols, including those that might be interesting from the smartphone users perspective, like: flashvideo, SIP, RTSP, H323, different P2P protocols, HTTP.

In order to test how traffic shaping detection works for different traffic shaping configurations, a number of traffic differentiation rules for the BitTorrent protocol have been deployed on the router (see Table 4.1). The router has been configured

Destination Port Number	Maximum allowed bandwidth
45000	no limit
45001	1024 kbps
45002	512 kbps
45003	256 kbps
45004	128 kbps
45005	64 kbps
45006	drop packets (100% drop rate)
45007	drop packets (50% drop rate)

Table 4.1: Throttling policy for BitTorrent flows deployed on the router in experimental enviroment

to limit the BitTorrent flows performance on port numbers 45000-45005 and to drop BitTorrent flow packets on port numbers 45006 and 45007 in both (upload and download) directions. The performance of flows that carry other protocol data is not affected and stays the same on all destination port numbers.

The evaluation of measurement results is discussed in Section 5.1.

4.3 Experiments on Mobile Networks

Once the application correctness was verified, we ran a series of traffic shaping detection tests on mobile networks. The main goal of these measurements was to learn how and to which extend content-based traffic differentiation policies are deployed on mobile networks.

Usually, mobile providers do not own the radio spectrum or wireless network infrastructure over which they provide services to the end-users. These mobile providers are called mobile virtual network operators (MVNOs) [77]. An MVNO enters into a business agreement with a mobile network operator (MNO) to obtain an access to network services at wholesale rates, then sets retail prices independently. For example, by July 2012, in Germany there are 4 mobile network operators and more than 30 mobile virtual network operators [78].

In our traffic shaping detection experiments we had an access to four SIM-cards from the mobile providers listed in table 4.2. These mobile providers use three out of four mobile networks that are available in Germany. Thereby, we can compare how the traffic shaping policies for different mobile network operators differ from each other.

Furthermore, we choose two mobile providers (*ALDI Talk* and *NettoKOM*), which are based on the same mobile network infrastructure, so we can compare the traffic shaping configurations used across the same mobile network operator.

Of course, the number of operators that we selected is not enough to determine whether the traffic shaping configuration depends on the policies deployed by a

MVNO	**MNO**
ALDI Talk (MEDIONMobile) [79]	*E-Plus*
NettoKOM [11]	*E-Plus*
O_2 [12]	O_2
Congstar [13]	*Telekom* (former *T-Mobile*)

Table 4.2: List of mobile operators used in our traffic shaping detection experiments on mobile networks.

mobile network operator or by mobile provider (MVNO), though further measurements for more providers would be helpful.

In order to answer the questions formulated in Introduction 1 we performed various traffic shaping detection tests for six different application protocols. Three different aspects were taken into account while running measurement tests:

a) **Mobile network type**. Mobile operators provide access to the Internet over different mobile network types (e.g., GSM, EDGE, UMTS, HSPA) In order to check, whether the traffic shaping policies within the same mobile differ for various network types, we ran measurement tests over the EDGE [80] and HSPA [81] network types (one of the most popular and widely deployed 2G and 3G mobile network technologies respectively).

b) **The time of the day**. One of our hypotheses was that mobile providers can behave differently depending on the time of the day. For example, they can limit their network bandwidth during the peak hours to reduce the load on their network equipment. To verify whether they actually do that, we ran the measurement tests during the different times of the day.

c) **Mobile network speed**. Usually, mobile providers charge users monthly for using the Internet and they provide access to the Internet regardless the amount of data that users consume. However the amount of data that users consume has an impact on the network speed. Once a user exceeds the limit (defined by a tariff-plan that is used), mobile providers restrict the network bandwidth to some comparatively low-speed for that user. In order to check, whether mobile operators additionally perform content-based traffic shaping in that condition, we ran a series of measurements after the limit has been crossed (see Table D.1 for details about the tariff-plans).

The traffic shaping detection application has been published on the Google Play content-distribution service [82]. Thereby, we also collected measurement results which have been submitted by the users who used our application.

The evaluation of experiment results is discussed in Sections 5.2 – 5.4.

Chapter 5

Evaluation

In this chapter we discuss the evaluation and the measurement results collected during the set of traffic shaping detection experiments. First, we will verify the correctness of our application by analyzing the traffic shaping detection test results made on the experimental setup. Next, we will discuss the results collected during the traffic shaping detection experiments on mobile networks. In addition to our experiments, we will discuss the measurement results which have been submitted by the users who downloaded the traffic shaping detection application from the Google Play service.

5.1 Application Correctness Verification

The first test to verify the correctness of the developed traffic shaping detection tool was to check whether the measured goodput for BitTorrent flows and Random flows stayed the same on port number 45000, where there was no traffic shaping policy deployed. For this purpose, we executed a measurement test with five measurement cycles. As we can see from the results (see Table C.2), the goodput values for BitTorrent and Random flows for the download (as well as for the upload) direction stayed very close to each other. Which indicates that the BitTorrent flows have not been affected by traffic shaping along the path. In addition, we checked the L7-filter log file to ensure that all BitTorrent flows have been successfully matched.

Afterwards, we ran the measurement tests on port numbers 45001 – 45005. The measured goodput for Random flows stayed very close to the goodput values observed in the experiments where no traffic shaping was involved (see Table C.2). And if we compare these values with measured performance for Protocol flows, we can see that the *Mann–Whitney U* values are `"0"` for all measurements. In addition, the maximum goodput values for the Protocol flows are always less than the corresponding lower bounds of the confidence intervals for the Random flows.

Therefore, we can conclude that different traffic shaping schemes were definitely applied along the path. Furthermore, the measured goodput for Protocol flows on different port numbers matches the bandwidth limits configured on the router (see Table 4.1).

In addition two more destination port numbers have been configured to manipulate on the BitTorrent traffic. On the port 45006 all the packets that belong to BitTorrent flows were dropped. The traffic shaping detection tests for port number 45006 showed (see Table C.1) that for the BitTorrent flows the `SocketTimeout Exception` has been thrown on every measurement cycle. In this way we can successfully detect traffic shaping that drops all the packets that belong to an "unwanted" protocol flow. Note that we have no measurement results for the download direction in this case. This happened because the download and the upload measurement tests are executed within the same TCP connection. According to the measurement test lifecycle (see Section 3.4), the client and the server inject the protocol messages before sending the test UUID. The L7-filter tool matched the flow after protocol messages had been sent, and the router started dropping the packages. As a result the **retrieve_test_results <uuid>** command failed later, since the test UUID had never been delivered to the server.

On port 45007 the packets of the BitTorrent flows were dropped with a constant drop rate of 50%. Measurement tests on port number 45007 showed (see Table C.1) that goodput for BitTorrent flows was approximately 2.5 times less than for Random flows. Therefore, we can successfully detect the traffic shaping based on the packet dropping with a constant rate techniques. However, we can not distinguish this type of traffic differentiation from the traffic shaping based on the limiting the network bandwidth (like on port numbers 45001 – 45005).

The results obtained from the measurements on the experimental setup helped us to verify that the developed traffic shaping detection application can successfully detect the following types of the traffic manipulations along the path:

– *Packet dropping* (dropping packets that carry data of some traffic type using a fixed or variable drop rate);

– *Packet blocking* (blocking packets or by injecting a connection termination message);

– *Deprioritizing* (assigning differentiated flows to lower priority queues in order to limit the goodput of certain classes).

5.2 Measurements on HSPA Mobile Networks

First, we ran measurement tests for *ALDITalk* and *NettoKOM* mobile operators which share the same *E-Plus* HSPA mobile network. Four experiments were completed for these two mobile providers: the daytime measurements (approximately from 1PM to 2PM for *ALDITalk*, from 3PM to 4PM for *NettoKOM*) and the nighttime measurements (from 4AM to 5AM for *ALDITalk*, from 12AM to 1AM for *NettoKOM*).

The measurement results with calculated *Mann-Whitney U* values, mean values and confidence intervals for the *ALDITalk* daytime experiments are presented in Tables E.1 and E.2; for the nighttime experiments in Tables E.3 and E.4; for the *NettoKOM* daytime experiments in Tables E.5 and E.6; for the nighttime experiments in Tables E.7 and E.8.

The statistical analysis showed that these two mobile operators do not apply content-based traffic shaping to the tested application protocols. As we can see from the measurement result tables, the $U > U_{critical}$ for all measurements, and the confidence intervals for corresponding Random and Protocol flows stay close to each other.

In addition, Figures 5.1 and 5.2 provide a comparison of measured goodput values (in kbps) for *ALDITalk's* and *NettoKOM's* HSPA mobile networks during different time of the day. These figures represent the ”minimum-maximum” goodput intervals and calculated confidence intervals for protocol (PF) and random (RF) flows for tested application protocols (we ran five measurement cycles within each measurement test). The interesting observation is that the confidence intervals' boundaries within the same direction for both operators stay approximately on the same level: ∼[1000;1150] kbps for the download direction and ∼[1200;1270] kbps for the upload direction. Also, the average goodput in the upload direction is usually 100-150 kbps higher than goodput in the download direction for both operators. Based on these two facts, we can assume that the *ALDITalk* and *NettoKOM* mobile operators use similar configurations of the *E-PLUS's* HSPA mobile network.

For some application protocols we can observe that calculated confidence intervals for protocol flow and random flow do not intersect. For example, during nighttime experiments for HTTP protocol on *NettoKOM's* HSPA mobile network, the confidence interval for random flow is ”higher” than corresponding confidence interval for protocol flow (see Figures 5.1 and 5.2). The mean goodput value for protocol flow is 14% less than the mean goodput value for random flow in the download direction, and 10% less in the upload direction. However, the *Mann-Whitney* values are $U = 8$ and $U = 11$ for the download and the upload direction respectively (while the $U_{critical} = 2$ in both cases). Therefore, we can conclude that the *NettoKOM* operator does not apply traffic shaping policy to an HTTP protocol. We believe that in case of traffic shaping, the mean goodput value for protocol flow will be significantly lower than the mean goodput value for random flow.

We can also observe the difference between protocol and random flows' confidence intervals in experiments on *NettoKOM's* HSPA mobile network for Flash Video protocol in the upload direction during nighttime (see Figure 5.2). However, in this case the mean goodput value for protocol flow is higher than the mean goodput value for random flow. Therefore, we can conclude that no traffic shaping policy has been applied to a FlashVideo protocol.

Next, we ran traffic shaping detection tests on the *Congstar's* HSPA mobile network. The measurements took place during the evening time (from 7PM to 9PM). The obtained measurement results are presented in Tables E.9 and E.10.

The first thing we can notice is that the measured goodput of SIP and random flows significantly differ from each other. The *Mann-Whitney U* values are 0 for both directions. The confidence intervals for the Random flows are [310;315] kbps for the download direction and [331;332] kbps for the download direction. While the confidence intervals for the SIP flows are [15;16] and [5] kbps respectively. Thereby, we can conclude that the *Congstar* operator does traffic shaping for the SIP application protocol. The measurement results for other tested protocols do not reveal the presence of traffic shaping along the path.

The second observation is that the measured goodput stays very stable for all measurements. And the goodput upper limit has a surprisingly small value (the maximum observed goodput is 360 kbps only), while the HSPA networks usually provide much faster connections (up to 7.2 Mbps).

Later, we repeated the traffic shaping detection test for the SIP protocol on *Congstar's* mobile network once again, but only during the morning hours at 9AM (see Table E.11 for results). As we can see, the mean goodput in the download direction for all application protocols increased by more than 16 times compared to previous measurement results. Moreover, we no longer observe the shaping of SIP protocol (the *Mann-Whitney U* values are 5 and 12 for the download and the upload directions respectively). From this we can assume that the *Congstar* provider does time-based traffic differentiation of the SIP protocol, and it limits the network performance in the download direction depending on the time of the day.

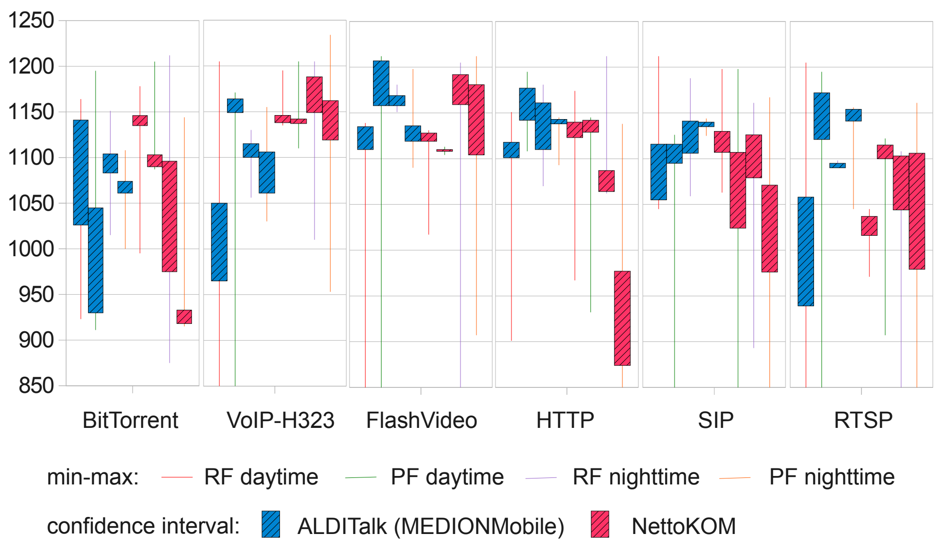


Figure 5.1: Comparison of goodput values (in kbps) for *ALDITalk's* and *NettoKOM's* HSPA mobile networks for random flows (RF) and protocol flows (PF) in the download direction during different time of the day.

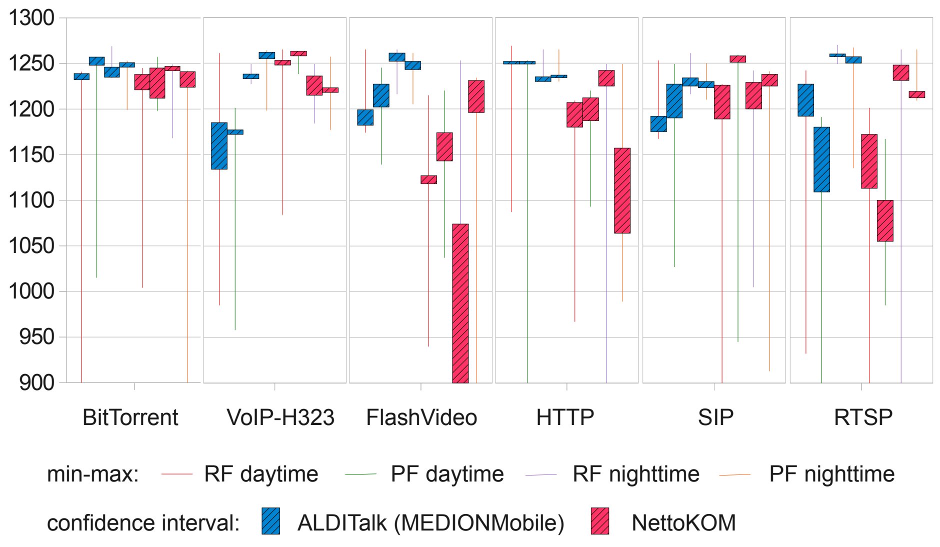


Figure 5.2: Comparison of goodput values (in kbps) for *ALDITalk's* and *NettoKOM's* HSPA mobile networks for random flows (RF) and protocol flows (PF) in the upload direction during different time of the day.

5.3 Measurements on EDGE Mobile Networks

Four traffic shaping detection experiments on EDGE networks were completed during the master thesis project: two experiments for the *NettoKOM* provider at the daytime (9AM – 10AM) and the nighttime (12AM – 1AM) and two experiments for the *ALDITalk* provider at the daytime (10AM and 1PM) and the nighttime (1AM).

The measurement results for *ALDITalk* EDGE network for the daytime are presented in Tables E.12 and E.13, for the nighttime in Tables E.14 and E.15. The daytime measurement results for *NettoKOM* EDGE network are presented in Tables E.16 and E.17, the nighttime results are in Tables E.18 and E.19.

The obtained measurement results show that both mobile operators do not apply traffic shaping policies for the tested application protocols on their EDGE networks. For all measurement, the *Mann-Whitney U* value is greater than the corresponding $U_{critical}$. And only for the HTTP protocol test in upload direction on *ALDITalk* network during the nighttime the $U < U_{critical}$. However, since the corresponding confidence intervals intersect we can make a decision that there is no traffic shaping observed.

Figures 5.3 and 5.4 provide a comparison of measured goodput values (in kbps) for *ALDITalk's* and *NettoKOM's* EDGE mobile networks during different time of the day. These figures represent the "minimum-maximum" goodput intervals and calculated confidence intervals for protocol (PF) and random (RF) flows for tested application protocols (we ran five measurement cycles within each measurement test). As expected, this mobile network type is characterized by a comparatively slow and unstable network performance. The confidence intervals for the download direction within the same network can significantly differ from each other depending on the time of the day (e.g., in *NettoKOM* network measured download goodput usually is 1.4 times higher than during the daytime). During the nighttime the EDGE networks tend to be more stable, especially in the upload direction.

In addition we can see that the *ALDITalk* operator provides the commensurable goodput for the upload and the download directions. While the download goodput for *NettoKOM* network is usually 4-5 times higher than the upload goodput. Thereby, we can conclude that these mobile operators use their own network configurations on top of the *E-Plus* mobile network.

For some application protocols we can observe that calculated confidence intervals for protocol flow and random flow do not intersect. For example, during nighttime experiments in the download direction for BitTorrent protocol on *ALDITalk's* EDGE mobile network, the confidence interval for random flow is "higher" than corresponding confidence interval for protocol flow (see Figure 5.3). The mean goodput value for protocol flow is 9% less than the mean goodput value for random flow. However, the *Mann-Whitney* value $U = 7$ which is greater than the $U_{critical} = 2$. Therefore, we can conclude that the *ALDITalk* operator does not apply traffic shaping policy to a BitTorrent protocol.

We can also observe the difference between protocol and random flows' confidence intervals in experiments on *ALDITalk's* EDGE mobile network for RTSP protocol in the download direction during nighttime (see Figure 5.3). However, in this case the mean goodput value for protocol flow is higher than the mean goodput value for random flow. Therefore, we can conclude that no traffic shaping policy has been applied to a RTSP protocol.

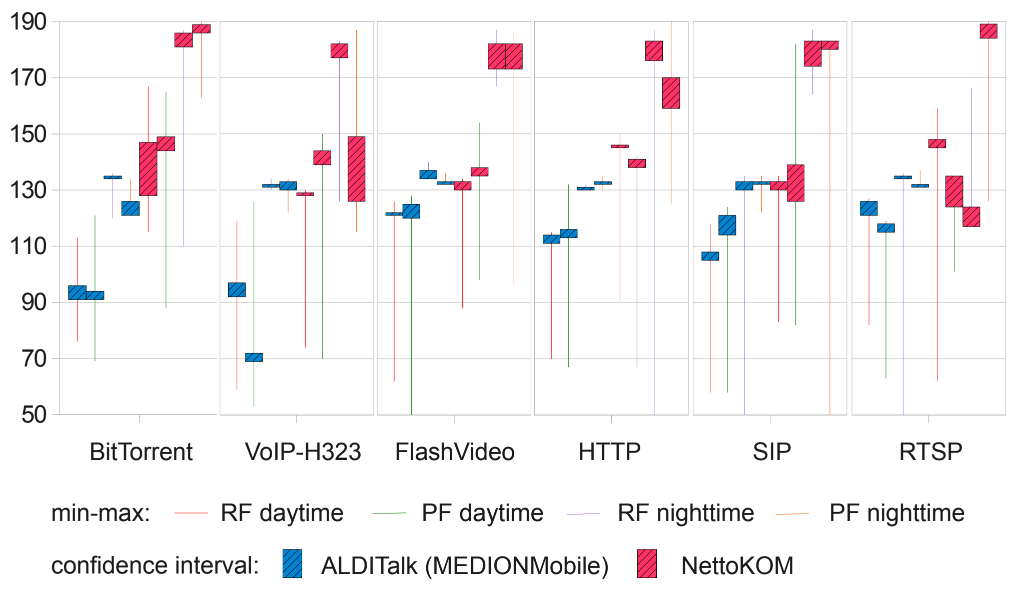


Figure 5.3: Comparison of goodput values (in kbps) for *ALDITalk's* and *NettoKOM's* EDGE mobile networks for random flows (RF) and protocol flows (PF) in the download direction during different time of the day.

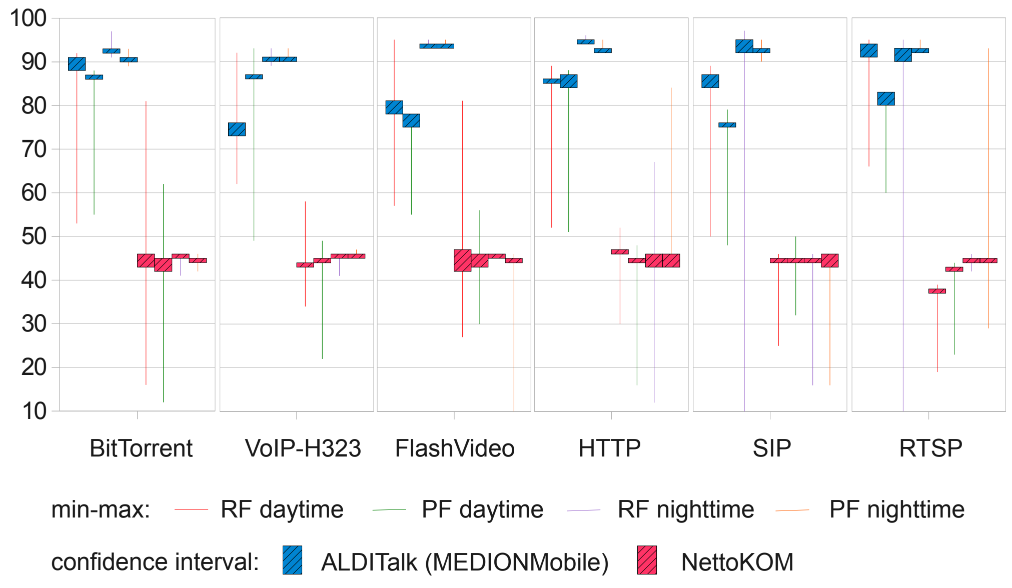


Figure 5.4: Comparison of goodput values (in kbps) for *ALDITalk's* and *NettoKOM's* EDGE mobile networks for random flows (RF) and protocol flows (PF) in the upload direction during different time of the day.

5.4 Measurements on Bandwidth-limited Mobile Networks

Usually, the amount of data that users consume have an impact on the mobile network speed. Once a user exceeds the limit (defined by a tariff-plan that is used), mobile providers restrict the network bandwidth to some comparatively low-speed for that user. In order to check, whether mobile operators additionally perform some content-based traffic shaping in this case, we ran a series of measurement tests on the HSPA mobile networks, after the limit has been crossed, for the following mobile operators: *NettoKOM*, *ALDITalk* and O_2 (see Table D.1 for details about the tariff-plans). All measurements took place during daytime: for *NettoKOM* at 10AM, for *ALDITalk* at 11AM, for O_2 at 2PM.

As we can see from the measurement results (see Tables E.22 and E.23 for the *ALDITalk* provider, Tables E.20 and E.21 for the *NettoKOM* provider, Tables E.24 and E.25 for the O2 provider), the calculated confidence intervals for the download and the upload directions stay close to each other for all Protocol flows and Random flows within the same mobile network. In addition, for almost all measurements the *Mann-Whitney U* value is greater than $U_{critical}$. Therefore, we can conclude that none of three tested operators does apply any traffic shaping policies to the tested application protocols in case of the bandwidth-limited mobile networks. In fact, the measured maximum goodput values for all mobile operators conform to the corresponding limits defined by the tariff-plans.

We can notice that the average goodput in the upload direction for the *NettoKOM* mobile operator is 2 times less than the average uplink goodput for the *ALDITalk* mobile operator. Which proves once again that *NettoKOM* and *ALDITalk* providers use different configurations of the same network.

5.5 Other Measurements

In addition, we collected the measurement results which have been submitted by the users who downloaded our application from the Google Play content distribution service.

A set of measurements was completed on the MTS BY (Belarus) HSPA mobile network. The measurement results are presented in Tables E.26 and E.27. Note, that traffic shaping detection tests have taken place during different times of the day. By comparing the calculated *Mann-Whitney U* values with corresponding $U_{critical}$ values and the confidence intervals for the Random and the Protocol flows, we can make a conclusion that this mobile operator does not apply any traffic shaping policy to the tested application protocols.

In addition, two more measurement test reports have been submitted. The measurement test results for the BitTorrent application protocol on the *LUXGSM* (Luxemburg) UMTS mobile network during the daytime are presented in Table E.28. The measurement test results for the HTTP application protocol on the *AT&T*

(USA) UMTS mobile network during the daytime are presented in Table E.29. In both cases these two mobile operators do not perform traffic shaping of the tested application protocols.

5.6 Measurement Test Performance

The main drawback of the presented traffic shaping detection mechanism is a need of injecting bulk messages into the tested mobile network in order to determine precisely the channel bandwidth. The exact number of bytes required to obtain reliable measurement results depends on the mobile network type and the actual network performance. Based on the measurement results from experiments on mobile networks, we compose Table 5.1, which represents the approximate amount of data consumed by the traffic shaping detection tool while it tests a mobile network in one (upload or download) direction for one flow depending on the measured goodput.

Measured goodput	**Amount of data**
$<$ 32 kpbs	16 kB
32 – 64 kbps	32 kB
64 – 128 kbps	64 kB
128 – 256 kbps	128 kB
256 – 512 kbps	256 kB
512 – 1024 kbps	512 kB
1 – 2 Mbps	1 MB
2 – 4 Mbps	2 MB
$>$ 4 Mbps	4 MB

Table 5.1: Approximate amount of data required to test a mobile network in one direction for one flow depending on the measured network performance

Hence we can estimate the total number of bytes in a single measurement test using the following rule:

$$B_{total} = N \cdot (B_{PFD} + B_{PFU} + B_{RFD} + B_{RFU})$$

where N is a number of measurement cycles, B_{PFD} and B_{RFD} are number of bytes required to test a network in the download direction for protocol flow and random flow respectively, B_{PFU} and B_{RFU} – for the upload direction.

For example, to run a test with a 5 measurement cycles on a mobile network with a speed limit up to 1200 kbps in both directions (quite common configuration in HSPA mobile networks), we need to consume approximately 20 MB of data. In a very fast networks this value can even grow up to 60 – 80 MB. Which is too much from a mobile user's perspective, because usually mobile operators reduce the channel bandwidth after the user exceeds some data limit. Reducing the amount of data consumed during the measurement cycle has a negative impact on the quality of measurement results. However, the user can define the number of measurement cycles within a measurement test from the client application's GUI.

Chapter 6

Conclusions

In the scope of the Master Thesis project, a measurement tool for detecting the presence of content-based traffic shaping mechanisms in mobile networks has been developed. This active measurement tool is based on a client-server architecture. The server component is a daemon running on a Linux server, which is reachable in the Internet network for other measurement participants. The client is an Android-application, which can be installed on any Android OS-based mobile device. When the client side establishes a connection with a measurement server, it can download and execute a number of tests that help to reveal the following deep-packet inspection based traffic manipulation methods along the path:

– *Packet dropping* (dropping packets that carry data of some traffic type using a fixed or variable drop rate);

– *Packet blocking* (blocking packets or by injecting a connection termination message);

– *Deprioritizing* (assigning flows to lower priority queues in order to limit the goodput of certain classes).

We have made a series of measurement experiments for different mobile network operators. The results have shown that for most of the them there is no content-based traffic shaping for the tested application protocol types. Surprisingly, most mobile providers do not block SIP and VoIP traffic, even despite the fact they claim that VoIP is not supported in their networks [11] [12]. The only evidence of traffic shaping that has been found refers to the provider Congstar, which limits the upload and download performance of SIP flows during the evening hours. This makes it impossible to make SIP-calls on this mobile network. However, it seems to be a time dependent traffic shaping policy, since during other experiments on this mobile network that have taken place during the morning hours the manipulation on the SIP flows has not been noticed.

Also, our experiments have shown that the tested mobile operators do not apply traffic shaping policies in case when they have already limited the network performance (after users crossed the data limit).

During developing and testing the traffic shaping detector tool, we faced a problem that is specific for active Internet measurements in mobile networks. Mobile network performance might not be very stable during the measurement test. For example, the network bandwidth can change significantly multiple times during a relatively short time frame. This is especially noticeably on low-speed networks like EDGE. The network performance depends on two factors: the signal strength and the load of the base station which the mobile device is connected to. Hence, during the analysis of the measurement results, we should keep in mind that significant changes in a flows' performances might be caused not only by traffic differentiation along the path.

To obtain reliable results, the client and the server need to send messages with a large payload. A single traffic shaping measurement test with five measurement cycles might consume up to 80 MB of data depending on the network type. This might repel mobile Internet users from using our application. However, reducing the amount of data consumed during the test has a negative impact on the quality of measurement results.

Bibliography

[1] Wikipedia. The Free Encyclopedia: Smartphone. Available at http://en.wikipedia.org/wiki/Smartphone.

[2] About.com - Part of The New York Times Company: What Makes a Smartphone Smart? Available at http://cellphones.about.com/od/smartphonebasics/a/what_is_smart.htm.

[3] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, Krishna P. Gummadi, Ratul Mahajan, and Stefan Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[4] Marcel Dischinger, Alan Mislove, Andreas Haeberlen, and Krishna P. Gummadi. Detecting BitTorrent Blocking. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC'08)*, Vouliagmeni, Greece, October 2008.

[5] Mukarram Bin Tariq, Murtaza Motiwala, Nick Feamster, and Mostafa Ammar. Detecting network neutrality violations with causal inference. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 289–300, New York, NY, USA, 2009. ACM.

[6] Ying Zhang, Zhuoqing Morley Mao, and Ming Zhang. Detecting traffic differentiation in backbone ISPs with NetPolice. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 103–115, New York, NY, USA, 2009. ACM.

[7] Partha Kanuparthy and Constantine Dovrolis. ShaperProbe: end-to-end detection of ISP traffic shaping using active methods. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 473–482, New York, NY, USA, 2011. ACM.

[8] Partha Kanuparthy and Constantine Dovrolis. DiffProbe: Detecting ISP Service Discrimination. In *INFOCOM*, pages 1649–1657. IEEE, 2010.

[9] Android Operating System. Available at http://android.com.

[10] Google Play: a digital-distribution multimedia-content service from Google Inc. Available at https://play.google.com/store.

[11] NettoKOM: Internet-Flat 1 GB. Available at http://www.nettokom.de/tarife/internet-flat-1gb.html.

[12] O2 Blue S – Der Tarif für Ihr Smartphone. Available at http://www.o2online.de/tarife/smartphone-tarife/o2-blue-s/.

[13] Congstar Surf Flat 500. Available at http://www.congstar.de/surf-flat-500/.

[14] Markus Peuhkuri. Internet traffic measurements aims, methodology, and discoveries. Master's thesis, Helsinki University of Technology, 2002.

[15] N. Duffield. Sampling for Passive Internet Measurement: A Review. *Statistical Science*, (19(3)), 2004.

[16] S. Deering and J. Mogul. *Path MTU Discovery. RFC 1191*, November 1990. Available at http://tools.ietf.org/html/rfc1191.

[17] C. Dovrolis, R. Prasad, M. Murray, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, (6):27–35, Apr 2003.

[18] S. Shalunov, B. Teitelbaum, A. Karp, J. Boote, and M. Zekauskas. *A One-way Active Measurement Protocol (OWAMP). RFC 4656*, September 2006. Available at http://tools.ietf.org/html/rfc4656.

[19] K. Hedayat, R. Krzanowski, A. Morton, K. Yum, and J. Babiarz. *A Two-Way Active Measurement Protocol (TWAMP). RFC 5357*, October 2008. Available at http://tools.ietf.org/html/rfc5357.

[20] CAIDA: The Cooperative Association for Internet Data Analysis. Available at http://www.caida.org/.

[21] CAIDA: The skitter tool. Available at http://www.caida.org/tools/measurement/skitter.

[22] CAIDA: Archipelago Measurement Infrastructure. Available at http://www.caida.org/projects/ark/.

[23] NetConfigs: A launch pad for network and Internet management related resources. Available at http://www.netconfigs.com/.

[24] Mohit Lad, Dan Massey, and Lixia Zhang. Link-Rank: A Graphical Tool for Capturing BGP Routing Dynamics. *Network Operations and Management Symposium (NOMS)*, April 2004.

[25] BGPlay @ Route Views. Available at http://bgplay.routeviews.org/.

[26] CAIDA: iffinder tool. Available at http://www.caida.org/tools/measurement/iffinder/.

[27] Mubashar Mushtaq and Toufik Ahmed. Adaptive Packet Video Streaming Over P2P Networks Using Active Measurements. In *Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 423–428, Washington, DC, USA, 2006. IEEE Computer Society.

[28] Reza Rejaie and Antonio Ortega. PALS: peer-to-peer adaptive layered streaming. In *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '03, pages 153–161, New York, NY, USA, 2003. ACM.

[29] Prasad Calyam, Weiping M, Mukundan Sridharan, Arif Khan, and Paul Schopis. H.323 Beacon: An H.323 application related end-to-end performance troubleshooting tool. In *ACM SIGCOMM NetTs*, 2004.

[30] Qian Zhang, Wenwu Zhu, and Ya-Qin Zhang. Channel-adaptive Resource Allocation for Scalable Video Transmission over 3G Wireless Network. *IEEE Trans. Circuits Syst. Video Techn.*, 14(8):1049–1063, 2004.

[31] Magnus Lundevall, Jonas Eriksson, Frida Eng, Birgitta Olin, Niclas Wiberg, and Stefan Wänstedt. Streaming Applications Over HSDPA in Mixed Service Scenarios. In *VTC,2004*.

[32] Tarek Saadawi. Plenary lecture 3: multicast active probing measurement technique for multimedia networks. In *Proceedings of the 5th European conference on European computing conference*, ECC'11, pages 20–20, Stevens Point, Wisconsin, USA, 2011. World Scientific and Engineering Academy and Society (WSEAS).

[33] David B. Makofske and Kevin C. Almeroth. Real-Time Multicast Tree Visualization and Monitoring. *Software–Practice and Experience*, 30:1047–1065, 2000.

[34] Mlisten: MBONE Collection Tool. Available at http://www.cc.gatech.edu/computing/Telecomm/projects/mbone/.

[35] RTPmon Tool. Available at http://www.cs.columbia.edu/~hgs/rtp/rtpmon.html.

[36] Internet Topology Discovery Using mrinfo. Available at http://inl.info.ucl.ac.be/content/mrinfo.

[37] CAIDA: Mantra - Monitoring Multicast on a Global Scale. Available at http://www.caida.org/tools/measurement/mantra/.

[38] Mtrace tool. Available at ftp://ftp.parc.xerox.com/pub/net-research/ipmulti/.

[39] D. Thaler and Adams A. Mrtree. Merit Network, Inc. and University of Michigan. Available at http://www.merit.edu/net-research/mbone/mrtree_man.html.

[40] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Services. RFC 2475*, December 1998. Available at http://tools.ietf.org/pdf/rfc2475.pdf.

[41] DslReports: Comcast is using Sandvine to manage P2P conectios. Available at http://www.dslreports.com/forum/r18323368-Comcast-is-using-Sandvine-to-manage-P2P-Connections.

[42] VELOCIX: New Generation Content Delivery Network. Available at http://www.velocix.com.

[43] Ying Zhang, Z. Morley, and Mao Ming Zhang. Ascertaining the Reality of Network Neutrality Violation in Backbone ISPs. In *In Proc. 7th ACM Workshop on Hot Topics in Networks (Hotnets-VII*, 2008.

[44] Neubot: The Network Neutrality Bot. Available at http://www.neubot.org/.

[45] Constantine Dovrolis, P. Krishna Gummadi, Aleksandar Kuzmanovic, and Sascha D. Meinrath. Measurement lab: overview and an invitation to the research community. *Computer Communication Review*, 40(3):53–56, 2010.

[46] SamKnows - Accurate broadband performance information for consumers, governments and ISPs. Available at http://www.samknows.com/broadband/index.php.

[47] RIPE Atlas Project. Available at http://atlas.ripe.net/.

[48] Apple unveils iPad 2, on sale in Australia on March 25. Available at http://www.startupsmart.com.au/technology/2011-03-03/apple-unveils-ipad-2-on-sale-in-australia-on-march-25.html.

[49] How many Android phones have been activated? Available at http://www.asymco.com/2011/12/21/how-many-android-phones-have-been-activated/.

[50] Network Diagnostic Tool (NDT). Available at http://www.internet2.edu/performance/ndt/.

[51] Windrider - A Mobile Network Neutrality Monitoring System. Available at http://www.cs.northwestern.edu/~ict992/mobile.htm.

[52] Fing - the Internet measurement tool. Available at http://www.over-look.com/site/.

[53] MobiPerf Official Website. Available at http://mobiperf.com.

[54] Google Play: Traffic Monitor application. Available at https://play.google.com/store/apps/details?id=com.radioopt.widget.

[55] J. Prokkola, P.H.J. Perala, M. Hanski, and E. Piri. 3G/HSPA Performance in Live Networks from the End User Perspective. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1 – 6, june 2009.

[56] K. Pentikousis, M. Palola, M. Jurvansuu, and P. Perala. Active goodput measurements from a public 3G/UMTS network. *Communications Letters, IEEE*, 9(9):802 – 804, sep 2005.

[57] P. Benko, G. Malicsko, and A. Veres. A large-scale, passive analysis of end-to-end TCP performance over GPRS. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1882 –1892 vol.3, march 2004.

[58] Wikipedia. The Free Encyclopedia: Goodput. Available at http://en.wikipedia.org/wiki/Goodput.

[59] The BitTorrent Protocol Specification, Version 11031. Available at http://bittorrent.org/beps/bep_0003.html.

[60] Wireshark: the world's foremost network protocol analyzer. Available at http://www.wireshark.org/.

[61] Wireshark Development: Libpcap File Format. Available at http://wiki.wireshark.org/Development/LibpcapFileFormat/.

[62] Jnetpcap library for an Android OS. Available at http://jnetpcap.com/userguide/android.

[63] IANA RFC 6335: Service Name and Transport Protocol Port Number Registry. Available at https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt.

[64] Application layer packet classier for Linux (L7-filter). Available at http://l7-filter.clearfoundation.com/.

[65] H. B. Mann and D. R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[66] Critical Values for the Mann-Whitney U-Test. Available at http://www.saburchill.com/IBbiology/downloads/002.pdf.

[67] Wikipedia. The Free Encyclopedia: H.323. Available at http://en.wikipedia.org/wiki/H.323.

[68] Wikipedia. The Free Encyclopedia: Session Initiation Protocol. Available at http://en.wikipedia.org/wiki/Session_Initiation_Protocol.

[69] Google Play: search query for the BitTorrent client applications. Available at https://play.google.com/store/search?q=bittorrent+client&c=apps.

[70] Google Play: search query for the Gnutella client applications. Available at https://play.google.com/store/search?q=Gnutella&c=apps.

[71] H. Schulzrinne, A. Rao, and R. Lanphier. *Real Time Streaming Protocol (RTSP). RFC 2326*, April 1998. Available at http://www.ietf.org/rfc/rfc2326.txt.

[72] Wikipedia. The Free Encyclopedia: Real Time Streaming Protocol. Available at http://en.wikipedia.org/wiki/RTSP.

[73] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 20112016. Available at http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html.

[74] OpenVPN Community Software. Available at http://openvpn.net/index.php/open-source.html.

[75] D. Guo, L. N. Bhuyan, and B. Liu. An Efficient Parallelized L7-Filter Design for Multicore Servers. *Networking, IEEE/ACM Transactions on*, PP(99):1, 2011.

[76] Technical Details of L7-filter. Available at http://l7-filter.sourceforge.net/technicaldetails.

[77] Wikipedia. The Free Encyclopedia: Mobile virtual network operator. Available at http://en.wikipedia.org/wiki/MVNO.

[78] Wikipedia. The Free Encyclopedia: List of mobile network operators of Europe. Available at http://en.wikipedia.org/wiki/List_of_mobile_network_operators_of_Europe#Germany.

[79] ALDI Talk: Mobiles Internet mit ALDI. Available at http://www.alditalk.de/web/internet-prepaid-tarif/flatrate_m/.

[80] A. Furuskar, S. Mazur, F. Muller, and H. Olofsson. EDGE: enhanced data rates for GSM and TDMA/136 evolution. *Personal Communications, IEEE*, 6(3):56 –66, jun 1999.

[81] S.I. Shah. UMTS: High Speed Packet Access (HSPA) Technology. In *Networking and Communications Conference, 2008. INCC 2008. IEEE International*, page 2, may 2008.

[82] Google Play: BonaFide Provider. Available at https://play.google.com/store/apps/details?id=de.jacobs.university.cnds.bonafide.

[83] Apache Maven Project. Available at http://maven.apache.org/.

[84] Android Developers: ADT Plugin for Eclipse. Available at http://developer.android.com/sdk/eclipse-adt.html.

[85] Android Developers: Building and Running from the Command Line. Available at http://developer.android.com/guide/developing/building/building-cmdline.html.

Appendix A

Source Code

The source code of the developed traffic shaping detection tool is publicly available under the BSD 2-Clause License and can be found at the CNDS website: `cnds.eecs.jacobs-university.de/software`

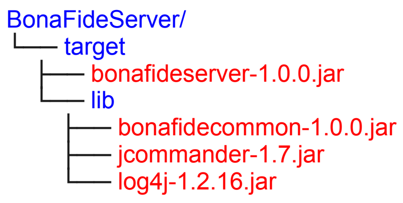
The implementation of a traffic shaping detection tool has three components:

– the server side implementation (”BonaFideServer”);

– the client side implementation (”BonaFideClient”);

– the common classes used by the server and the client components (”BonaFide-Common”);

In order to build the ”BonaFideServer” component the Apache Maven [83] build automation tool can be used. The project build process is triggered by running the `mvn install` command from the source code's root folder. The ”BonaFide-Common” and ”BonaFideServer” components are build in this case. The folder called ”*target*” will be created inside the ”BonaFideServer” project's folder that contains the executable jar file and required dependencies (see Figure A.1).



```
BonaFideServer/
└── target
    ├── bonafideserver-1.0.0.jar
    └── lib
        ├── bonafidecommon-1.0.0.jar
        ├── jcommander-1.7.jar
        └── log4j-1.2.16.jar
```

Figure A.1: BonaFideServer component's build structure

The ”BonaFideServer” project uses two additional dependency libraries:

– **log4j** is used for logging and printing traces during the program runtime;

– **jcommander** – a small java library used for parsing initial program input parameters;

The server component can be started using the following command, where **-p**, **-l** and **-s** are program input parameters defined in Table A.1:

```
java -jar bonafideserver-1.0.0.jar -p 4000 -l list.txt
-s incoming
```

The set of pre-defined protocol description files is also provided with the source code (see *protocols* folder). Modify the *list* file in order to specify the full path to protocol description files, and specify the path to list file as **-l** parameter when you start the ”BonaFideServer” component.

Note, for starting the server component as a daemon through a SSH session we need to tell the program to ignore the HUP (hangup) signal, enabling the command to keep running after the user who issues the command has logged out. For these purposes we can run ”BonaFideServer” service using the `nohup` POSIX command. Or, under the Debian systems, it is possible to use the following command to daemonise a process:

```
/sbin/start-stop-daemon
```

Parameter	**Description**	**Mandatory**
-p	port number for running Main Socket	no[1]
-l	path to a file which maintains a list of protocols to load	yes
-s	path to a folder for storing submitted results	yes
-v	defines logging level	no[2]

Table A.1: BonaFideServer component’s input parameters

To build the ”BonaFideClient” component it is recommended to use the Eclipse IDE with pre-installed ADT plugin [84], which was specially designed to help in Android applications development process. However, you can also build and install the ”BonaFide Provider” application manually using the command line (please refer to the ”Building and running from the command line” [85] tutorial for details).

The client component implementation is an application for the Android OS that can be installed on any Android-powered mobile device. The final release is named ”BonaFide Provider” and it can be directly installed from the Google Play digital-distribution multimedia-content service using the following URL:

`https://play.google.com/store/apps/details?id=de.jacobs.university.cnds.bonafide`

[1]Port number 4000 is a default value

[2]Possible logging level values sorted by priority: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE. INFO is a default value.

Figure A.2: BonaFide Provider application QRcode

or using the generated QRcode (see Figure A.2):

During the installation process the ”BonaFide Provider” application asks for the following permissions:

– **android.permission.INTERNET**: allows an application to open network sockets;

– **android.permission.ACCESS_NETWORK_STATE**: allows an application to access information about networks;

– **android.permission.WRITE_EXTERNAL_STORAGE**: allows an application to write to external storage (SD card);

– **android.permission.READ_PHONE_STATE**: allows read only access to phone state (required to obtain the information about the mobile service operator).

Appendix B

Pseudocode

```
Only for client: {
    establish a new TCP connection with the main socket
    send start_new_test <protocol_name> <number_of_cycles>
    receive UUID
    close a TCP connection
    measure_flow() /* for protocol flow                           */
    measure_flow() /* for random flow                             */
}
Only for server: {
For Random and Protocol sockets:
while true do
    accept new TCP connection
    create new Thread(measure_flow())
end
}
```

Algorithm 1: Running new measurement test pseudocode.

```
if (client) then
    for cycle_number := 1 to number_of_cycles do
        establish a new TCP connection
        inject protocol messages
        send UUID and cycle_number
        /* measure in upload direction                          */
        run_measurement_cycle(true, true)
        /* measure in download direction                        */
        run_measurement_cycle(false, true)
        cycle_number++
        close a TCP connection
    end
end
if (server) then
    inject protocol messages
    receive UUID and cycle_number
    /* measure in upload direction                              */
    run_measurement_cycle(true, false)
    /* measure in download direction                            */
    run_measurement_cycle(false, false)
end
```

Algorithm 2: Function `measure_flow()` pseudocode.

```
input:
boolean measure_uplink; true – current cycle measures uplink performance,
false – downlink performance
boolean client: true – function called from the client, false – from the server
initialization: set bulk_message_size value to minimum value
while true do
    if (measure_uplink and client) or (!measure_uplink and !client) then
        inject protocol messages
        send bulk_message
        receive "OK" notification
        calculate RTT
        if (RTT $\leq$ MAX_RTT and bulk_message_size < MAX_SIZE) then
            increase bulk_message_size
        else
            send terminate_message
            break
        end
    else
        inject protocol messages
        receive bulk_message
        if (bulk_message is terminate_message) then
            break
        else
            send "OK" notification
        end
    end
end
```

Algorithm 3: Function `run_measurement_cycle()` pseudocode.

```
input :
$U$: calculated Mann – Whitney value
$U_{critical}$: critical Mann – Whitney value
$P_{mean}$: mean goodput value for Protocol flow
$R_{mean}$: mean goodput value for Random flow
$P_{max}$: max goodput value for Protocol flow
$\sigma_P$: sigma value for Protocol flow (level of confidence 95%)
$\sigma_R$: sigma value for Random flow (level of confidence 95%)
output:
decision about the presence of traffic shaping
if $(U > U_{critical})$ then
    return NO SHAPING
else
    if $(P_{mean} \geq R_{mean})$ or $((P_{mean} + \sigma_P) \geq (R_{mean} - \sigma_R))$ then
        return NO SHAPING
    end
    if $(P_{mean} > 0.8 * R_{mean})$ or $(P_{max} \geq (R_{mean} - \sigma_R))$ then
        return MOST PROBABLY NOT SHAPING
    end
    if $(0.6 * R_{mean} < P_{mean} \leq 0.8 * R_{mean})$ then
        return MOST PROBABLY SHAPING
    end
    if $(P_{mean} \leq 0.6 * R_{mean})$ then
        return SHAPING
    end
end
```

Algorithm 4: Decision about the presence of traffic shaping pseudocode.

Appendix C

Verification Test Results

Port	**Cycle**	**Download**		**Upload**		U_d	U_u	$D_{intervals}$	$U_{intervals}$
		RF	**PF**	**RF**	**PF**				
45006	1	no data	no data	1540	`timeout`	–	–	–	–
	2			1538	`timeout`				
	3			**1542**	`timeout`				
	4			1541	`timeout`				
	5			1538	`timeout`				
45007	1	6883	2805	1540	618	0	0	Random: [6885;6888] Protocol: [2808;2809]	Random: [618;621] Protocol: [1539;1542]
	2	**6889**	2809	1538	**623**				
	3	6892	2801	1541	622				
	4	6880	**2810**	1543	620				
	5	6885	2809	**1544**	616				

Table C.1: Measured goodput values (in kbps) for BitTorrent and Random flows on destination ports 45006 and 45007[1]

[1]U_d – Mann-Whitney U value for the download direction; U_u – Mann-Whitney U value for the upload direction; $D_{intervals}$ – 95% confidence intervals for the download direction; $U_{intervals}$ – 95% confidence intervals for the upload direction;

Port	**Cycle**	**Download**		**Upload**		U_d	U_u	$D_{intervals}$	$U_{intervals}$
		RF	**PF**	**RF**	**PF**				
	1	7043	7051	1544	1552				
	2	**7049**	7049	1543	**1560**			Random:	Random:
45000	3	7035	**7054**	1540	1554	2	0	[7043;7046]	[1540;1543]
	4	7046	7052	1541	1548			Protocol:	Protocol:
	5	7040	**7054**	**1545**	1557			[7053;7054]	[1552;1555]
	1	6981	1012	1539	1016				
	2	6988	**1015**	**1541**	1016			Random:	Random:
45001	3	**6992**	1012	1538	1017	0	0	[6989;6992]	[1537;1538]
	4	6990	1013	1537	**1019**			Protocol:	Protocol:
	5	6885	1012	1535	1016			[1011;1012]	[1015;1016]
	1	6095	**481**	1536	483				
	2	7003	**481**	1539	**484**			Random:	Random:
45002	3	7002	**481**	1539	483	0	0	[6997;7010]	[1537;1538]
	4	6098	**481**	1538	**484**			Protocol:	Protocol:
	5	**7005**	**481**	**1540**	483			[481]	[482;483]
	1	**7008**	**212**	1540	210				
	2	7006	**212**	**1541**	210			Random:	Random:
45003	3	7008	**212**	**1541**	**211**	0	0	[7007;7008]	[1540;1541]
	4	7009	**212**	1540	210			Protocol:	Protocol:
	5	7004	**212**	**1541**	210			[212;212]	[209;210]
	1	**7026**	**101**	1537	**103**				
	2	7023	**101**	**1538**	**103**			Random:	Random:
45004	3	7019	**101**	**1538**	**103**	0	0	[7022;7023]	[1537, 1538]
	4	7022	**101**	**1538**	**103**			Protocol:	Protocol:
	5	7023	**101**	1536	**103**			[101]	[103]
	1	7018	58	1540	**57**				
	2	7023	58	1542	**57**			Random:	Random:
45005	3	**7025**	58	1543	56	0	0	[7019;7022]	[1541;1542]
	4	7018	**59**	**1545**	**57**			Protocol:	Protocol:
	5	7015	58	1541	**57**			[57;58]	[56;57]

Table C.2: Measured goodput values (in kbps) for BitTorrent and Random flows on destination ports 45000 – 45005

Appendix D

Mobile Operators Details

Mobile Provider, tarif-plan name	**Maximum Bandwidth**	**Data Limit**	**Reduced Bandwidth**
ALDI Talk (MEDIONMobile), Internet-Flatrate M 1GB [79]	7.2 Mbps	500MB	56 kbps
NettoKOM, Internet-Flat 1GB [11]	7.2 Mbps	1 GB	56 kbps
O_2, *Blue S* [12]	7.2 Mbps	300 MB	64 kbps
Congstar, Surf Flat 500 [13]	7.2 Mbps	500 MB	64 kbps download 16 kbps upload

Table D.1: Details of the tariff plans of mobile networks used during the traffic shaping detection experiments

Appendix E

Experiment Results

Here and after, in the measurement results tables the following abbreviations are used:

RF – Random flow;

PF – Protocol flow;

U – Mann-Whitney U value;

U_c – Mann-Whitney U critical value;

R_m – mean goodput for the Random flow[1];

P_m – mean goodput for the Protocol flow;

$R_{interval}$ – 95% confidence interval for the Random flow;

$P_{interval}$ – 95% confidence interval for the Protocol flow;

[1]Boundaries of the confidence intervals and the mean values have been rounded to integer values.

Protocol	Cycle	Download		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	923	970	8	2	988	[1026;1141]	1084	[930;1045]
	2	1102	**1195**						
	3	940	1135						
	4	797	911						
	5	**1164**	1149						
VoIP-H323	1	**1205**	**1171**	10	2	1008	[965;1050]	1157	[1149;1164]
	2	1093	1059						
	3	962	1154						
	4	804	419						
	5	971	1148						
FlashVideo	1	1132	1135	10	2	1123	[1110;1135]	1183	[1158;1207]
	2	1098	1202						
	3	757	**1212**						
	4	**1139**	969						
	5	1059	704						
HTTP	1	1117	1108	3	2	1110	[1101;1118]	1160	[1142;1177]
	2	1104	**1195**						
	3	**1151**	1171						
	4	915	1126						
	5	901	1184						
SIP	1	1056	**1126**	10	2	1086	[1055;1116]	1106	[1095;1116]
	2	**1212**	1090						
	3	1117	1123						
	4	1045	577						
	5	1198	1087						
RTSP	1	1117	534	11	2	999	[939;1058]	1147	[1121;1172]
	2	958	285						
	3	923	**1195**						
	4	**1205**	1135						
	5	843	1111						

Table E.1: The results of measurements made on the *ALDITalk's* HSPA mobile network in the download direction during the daytime (1PM – 2PM).

Protocol	Cycle	Upload		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	756	1015	8	2	1236	[1232;1239]	1253	[1248;1257]
	2	**1241**	1245						
	3	1231	**1257**						
	4	1238	**1257**						
	5	1209	1205						
VoIP-H323	1	1208	1171	12	2	1160	[1134;1185]	1175	[1172;1177]
	2	**1261**	1180						
	3	985	958						
	4	1154	**1201**						
	5	1120	1174						
FlashVideo	1	**1265**	1205	12	2	1191	[1182;1199]	1215	[1202;1227]
	2	1205	1239						
	3	1174	1201						
	4	1208	**1245**						
	5	1178	1139						
HTTP	1	1250	1245	10	2	1255	[1249;1252]	1251	[1249;1252]
	2	1261	**1253**						
	3	1241	1249						
	4	1087	813						
	5	**1269**	**1253**						
SIP	1	**1253**	**1249**	11	2	1184	[1175;1192]	1209	[1190;1227]
	2	1167	1160						
	3	1191	1027						
	4	1154	1191						
	5	1195	1227						
RTSP	1	802	1076	9	2	1210	[1192;1227]	1145	[1109;1180]
	2	1209	1043						
	3	932	737						
	4	**1242**	1168						
	5	1181	**1191**						

Table E.2: The results of measurements made on the *ALDITalk's* HSPA mobile network in the upload direction during the daytime (1PM – 2PM).

Protocol	Cycle	Download		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
	1	1084	1000						
	2	1104	1070						
BitTorrent	3	**1151**	1078	10	2	1094	[1083;1104]	1068	[1061;1074]
	4	1020	1056						
	5	1015	**1108**						
	1	1117	1129						
	2	**1130**	**1155**						
VoIP-H323	3	1093	1030	10	2	1108	[1100;1115]	1084	[1061;1106]
	4	1056	1056						
	5	1114	1067						
	1	1174	1142						
	2	1154	1114						
FlashVideo	3	1151	**1198**	5	2	1164	[1158;1169]	1128	[1119;1136]
	4	**1181**	1090						
	5	1164	1129						
	1	1178	1138						
	2	1070	1093						
HTTP	3	1089	1123	11	2	1136	[1110;1161]	1141	[1138;1143]
	4	**1184**	**1145**						
	5	1142	1142						
	1	1107	1135						
	2	1062	1136						
SIP	3	**1188**	**1144**	8	2	1124	[1106;1141]	1138	[1135;1140]
	4	1141	1143						
	5	1059	1125						
	1	1120	1135						
	2	1150	**1156**						
RTSP	3	**1098**	1045	8	2	1093	[1090;1095]	1148	[1141;1154]
	4	1093	1154						
	5	1090	1108						

Table E.3: The results of measurements made on the *ALDITalk's* HSPA mobile network in the download direction during the nighttime (4AM – 5AM).

Protocol	**Cycle**	**Upload**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	1253	1246	11	2	1241	[1235;1246]	1249	[1246;1251]
	2	1234	1237						
	3	1238	1249						
	4	**1269**	**1253**						
	5	1220	1199						
VoIP-H323	1	1234	1198	10	2	1236	[1233;1238]	1259	[1255;1262]
	2	1242	1260						
	3	**1249**	**1264**						
	4	1227	1220						
	5	1234	1253						
FlashVideo	1	1261	1246	8	2	1257	[1252;1261]	1248	[1243;1252]
	2	1216	1242						
	3	1249	**1261**						
	4	1253	1205						
	5	**1265**	1257						
HTTP	1	**1265**	1245	12	2	1233	[1230;1235]	1236	[1234;1237]
	2	1230	1230						
	3	1235	**1265**						
	4	1243	1238						
	5	1237	1232						
SIP	1	1216	1226	7	2	1230	[1225;1234]	1227	[1223;1230]
	2	1235	1222						
	3	1226	1234						
	4	**1261**	**1250**						
	5	1257	1210						
RTSP	1	1261	1135	7	2	1259	[1257;1260]	1254	[1250;1257]
	2	**1270**	**1267**						
	3	1249	1242						
	4	1261	1249						
	5	1261	1253						

Table E.4: The results of measurements made on the *ALDITalk's* HSPA mobile network in the upload direction during the nighttime (4AM – 5AM).

Protocol	Cycle	Download		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	1132	1108	9	2	1141	[1135;1146]	1097	[1090;1103]
	2	1145	1132						
	3	1148	1087						
	4	**1178**	**1205**						
	5	995	1096						
VoIP-H323	1	**1195**	1110	7	2	1140	[1135;1146]	1196	[1137;1142]
	2	1139	**1205**						
	3	1145	1195						
	4	1138	1184						
	5	1164	1198						
FlashVideo	1	1126	1009	6	2	1124	[1119;1128]	1108	[1105;1110]
	2	1017	**1113**						
	3	1102	1009						
	4	**1131**	1108						
	5	1117	1104						
HTTP	1	1134	932	9	2	1132	[1123;1140]	1136	[1129;1142]
	2	1145	1084						
	3	1117	1141						
	4	**1174**	1123						
	5	967	**1145**						
SIP	1	**1198**	1025	6	2	1119	[1107;1130]	1066	[1024;1107]
	2	1120	816						
	3	1099	782						
	4	1063	**1198**						
	5	1138	1108						
RTSP	1	443	907	8	2	1027	[1016;1037]	1108	[1100;1115]
	2	971	**1122**						
	3	1008	1098						
	4	1029	1106						
	5	**1045**	930						

Table E.5: The results of measurements made on the *NettoKOM's* HSPA mobile network in the download direction during the daytime (3PM – 4PM).

Protocol	**Cycle**	**Upload**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	1004	1238	7	2	1230	[1221;1238]	1229	[1212;1245]
	2	1019	1174						
	3	**1245**	1198						
	4	1227	1253						
	5	1220	**1257**						
VoIP-H323	1	1249	1257	8	2	1261	[1248;1253]	1251	[1258;1263]
	2	**1265**	1238						
	3	1084	1249						
	4	1261	1249						
	5	1257	**1261**						
FlashVideo	1	**1215**	1037	8	2	1123	[1118;1127]	1159	[1143;1174]
	2	1126	1167						
	3	940	**1220**						
	4	1129	1181						
	5	1116	1129						
HTTP	1	1110	1093	10	2	1194	[1180;1207]	1200	[1187;1212]
	2	1206	1212						
	3	1009	1188						
	4	**1209**	1090						
	5	967	**1220**						
SIP	1	**1227**	957	8	2	1208	[1189;1226]	1255	[1251;1258]
	2	1171	**1259**						
	3	717	945						
	4	1067	1257						
	5	**1227**	1249						
RTSP	1	**1201**	**1167**	12	2	1143	[1113;1172]	1078	[1055;1100]
	2	1123	1059						
	3	1105	985						
	4	446	1123						
	5	864	1054						

Table E.6: The results of measurements made on the *NettoKOM's* HSPA mobile network in the upload direction during the daytime (3PM – 4PM).

Protocol	**Cycle**	**Download**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	969	940	9	2	1036	[975;1096]	926	[918;933]
	2	1157	915						
	3	983	985						
	4	875	923						
	5	**1212**	**1144**						
VoIP-H323	1	1151	1181	12	2	1169	[1149;1188]	1141	[1119;1162]
	2	1012	**1234**						
	3	1010	953						
	4	1188	1110						
	5	**1205**	1132						
FlashVideo	1	1177	**1212**	10	2	1176	[1159;1192]	1143	[1104;1181]
	2	1148	907						
	3	555	1181						
	4	840	944						
	5	**1205**	1105						
HTTP	1	**1212**	988	8	2	1076	[1064;1087]	926	[874;977]
	2	1099	825						
	3	1062	967						
	4	1067	**1138**						
	5	350	466						
SIP	1	1081	1114	12	2	1103	[1079;1126]	1024	[976;1071]
	2	**1161**	953						
	3	923	1007						
	4	1126	736						
	5	893	**1167**						
RTSP	1	1099	1073	8	2	1074	[1044;1103]	1043	[979;1106]
	2	456	843						
	3	1015	923						
	4	**1108**	**1161**						
	5	713	1135						

Table E.7: The results of measurements made on the *NettoKOM's* HSPA mobile network in the download direction during the nighttime (12AM – 1AM).

Protocol	**Cycle**	**Upload**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	1245	589	8	2	1245	[1242;1247]	1233	[1224;1241]
	2	**1249**	1216						
	3	1216	**1242**						
	4	1168	1171						
	5	1241	**1242**						
VoIP-H323	1	1208	1177	11	2	1226	[1215;1236]	1221	[1218;1223]
	2	**1249**	1223						
	3	1184	**1257**						
	4	1230	1219						
	5	1242	1223						
FlashVideo	1	676	858	10	2	963	[851;1074]	1214	[1196;1231]
	2	**1253**	681						
	3	1174	1181						
	4	923	1227						
	5	793	**1234**						
HTTP	1	**1249**	1129	11	2	1234	[1225;1242]	1111	[1064;1157]
	2	1220	**1249**						
	3	1004	989						
	4	1234	1181						
	5	883	1024						
SIP	1	1184	913	10	2	1215	[1200;1229]	1232	[1225;1238]
	2	**1242**	1234						
	3	1238	**1241**						
	4	1005	955						
	5	1198	1223						
RTSP	1	1234	1223	8	2	1240	[1231;1248]	1216	[1212;1219]
	2	1231	1220						
	3	1257	**1265**						
	4	**1265**	1213						
	5	874	1209						

Table E.8: The results of measurements made on the *NettoKOM's* HSPA mobile network in the upload direction during the nighttime (12AM – 1AM).

Protocol	Cycle	Download		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	317	**357**	12	2	317	[316;317]	318	[310;325]
	2	**348**	328						
	3	318	322						
	4	319	305						
	5	317	290						
H323	1	299	283	12	2	303	[297;326]	318	[299;306]
	2	296	297						
	3	274	304						
	4	**350**	**340**						
	5	341	308						
FlashVideo	1	306	314	11	2	314	[304;315]	310	[307;320]
	2	320	**337**						
	3	**335**	325						
	4	304	304						
	5	273	292						
HTTP	1	303	355	11	2	305	[302;307]	317	[313;320]
	2	333	**324**						
	3	**328**	312						
	4	310	316						
	5	303	300						
SIP	1	311	**17**	0	2	313	[310;315]	16	[15;16]
	2	**344**	16						
	3	324	13						
	4	318	**17**						
	5	311	15						
RTSP	1	**319**	313	10	2	319	[318;319]	322	[316;327]
	2	**319**	321						
	3	**319**	332						
	4	**319**	**360**						
	5	316	308						

Table E.9: The results of measurements made on the *Congstar's* HSPA mobile network in the download direction during the evening (7PM).

Protocol	**Cycle**	**Upload**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	32	333	12	2	332	[328;335]	332	[331;332]
	2	**336**	329						
	3	333	**336**						
	4	335	331						
	5	326	332						
H323	1	**338**	**335**	6	2	324	[320;327]	333	[332;333]
	2	321	332						
	3	316	326						
	4	321	**335**						
	5	331	332						
FlashVideo	1	334	320	10	2	332	[330;333]	327	[324;329]
	2	330	331						
	3	332	324						
	4	318	**340**						
	5	**337**	328						
HTTP	1	**333**	334	8	2	332	[331;332]	331	[328;333]
	2	**333**	334						
	3	320	**337**						
	4	331	322						
	5	328	327						
SIP	1	329	**5**	0	2	332	[331;332]	5	[5]
	2	**332**	**5**						
	3	320	**5**						
	4	**332**	**5**						
	5	328	**5**						
RTSP	1	27	25	12	2	324	[317;330]	28	[23;32]
	2	315	28						
	3	30	32						
	4	**332**	330						
	5	325	**338**						

Table E.10: The results of measurements made on the *Congstar's* HSPA mobile network in the upload direction during the evening (7PM).

Direction	**Cycle**	**RF**	**PF**	U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
Download	1	5073	5471	5	2	5569	[5544;5593]	4927	[4646;5207]
	2	5481	4540						
	3	5545	**5650**						
	4	**5689**	4770						
	5	5593	4369						
Upload	1	356	355	12	2	354	[353;354]	355	[354;355]
	2	355	357						
	3	340	**341**						
	4	**352**	355						
	5	354	337						

Table E.11: SIP protocol measurement test results on Congstar's HSPA mobile network during the morning hours (9AM).

Protocol	Cycle	Download		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	76	**121**	12	1	94	[91;96]	93	[91;94]
	2	95	93						
	3	**113**	95						
	4	timeout	69						
	5	93	93						
VoIP-H323	1	94	71	7	1	95	[92;97]	71	[69;72]
	2	reset	53						
	3	96	**126**						
	4	**119**	72						
	5	59	70						
FlashVideo	1	123	102	8	2	123	[121;122]	122	[120;125]
	2	123	49						
	3	**126**	122						
	4	122	**128**						
	5	62	120						
HTTP	1	**115**	67	7	2	113	[111;114]	115	[113;116]
	2	112	117						
	3	70	115						
	4	113	**132**						
	5	114	115						
SIP	1	**118**	**124**	11	2	107	[105;108]	118	[114;121]
	2	110	58						
	3	58	114						
	4	107	118						
	5	105	85						
RTSP	1	64	101	8	2	124	[121;126]	117	[115;118]
	2	120	63						
	3	82	116						
	4	**127**	116						
	5	125	**119**						

Table E.12: The results of measurements made on the *ALDITalk's* EDGE mobile network in the download direction during the daytime (10AM and 1PM).

Protocol	**Cycle**	**Upload**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	**92**	**88**	5	1	90	[88;91]	87	[86;87]
	2	90	86						
	3	reset	55						
	4	89	**88**						
	5	53	87						
VoIP-H323	1	**92**	87	7	1	75	[73;76]	87	[86;87]
	2	timeout	89						
	3	62	87						
	4	74	49						
	5	76	**93**						
FlashVideo	1	**95**	76	3	2	80	[78;81]	77	[75;78]
	2	57	76						
	3	79	**78**						
	4	81	54						
	5	82	55						
HTTP	1	52	**88**	3	2	86	[85;86]	86	[84;87]
	2	87	85						
	3	**89**	77						
	4	87	51						
	5	86	85						
SIP	1	**89**	48	8	2	86	[84;87]	76	[75;76]
	2	86	76						
	3	84	76						
	4	50	78						
	5	70	**79**						
RTSP	1	66	60	3	2	93	[91;94]	82	[80;83]
	2	**95**	82						
	3	91	52						
	4	88	**83**						
	5	94	81						

Table E.13: The results of measurements made on the *ALDITalk's* EDGE mobile network in the upload direction during the daytime (10AM and 1PM).

Protocol	Cycle	Download		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	135	120	7	2	135	[134;135]	124	[121;126]
	2	135	129						
	3	**136**	**134**						
	4	122	123						
	5	120	121						
VoIP-H323	1	133	132	6	2	132	[131;132]	132	[130;133]
	2	130	**134**						
	3	132	123						
	4	**134**	122						
	5	132	132						
FlashVideo	1	139	133	4	2	136	[134;137]	133	[132;133]
	2	**140**	135						
	3	134	**136**						
	4	135	133						
	5	135	133						
HTTP	1	**132**	130	5	2	131	[130;131]	133	[132;133]
	2	131	133						
	3	131	**135**						
	4	131	134						
	5	**132**	133						
SIP	1	12	133	9	2	132	[130;133]	133	[132;133]
	2	130	122						
	3	**135**	133						
	4	133	134						
	5	120	**135**						
RTSP	1	123	133	12	2	135	[134;135]	132	[131;132]
	2	134	**137**						
	3	**136**	132						
	4	24	132						
	5	135	133						

Table E.14: The results of measurements made on the *ALDITalk's* EDGE mobile network in the download direction during the nighttime (1AM).

Protocol	Cycle	Upload		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	93	90	7	2	93	[92;93]	91	[90;91]
	2	**97**	**93**						
	3	91	89						
	4	93	91						
	5	93	92						
VoIP-H323	1	89	**93**	10	2	91	[90;91]	91	[90;91]
	2	91	92						
	3	**93**	91						
	4	91	91						
	5	92	91						
FlashVideo	1	**95**	94	8	2	94	[93;94]	94	[93;94]
	2	94	94						
	3	94	93						
	4	94	**95**						
	5	94	**95**						
HTTP	1	95	93	1	2	95	[94;95]	93	[92;93]
	2	94	93						
	3	**96**	94						
	4	95	93						
	5	95	**95**						
SIP	1	5	90	11	2	94	[92;95]	93	[92;93]
	2	94	94						
	3	**97**	**95**						
	4	92	94						
	5	94	93						
RTSP	1	92	93	7	2	92	[90;93]	93	[92;93]
	2	91	92						
	3	**95**	**95**						
	4	7	94						
	5	92	**95**						

Table E.15: The results of measurements made on the *ALDITalk's* EDGE mobile network in the upload direction during the nighttime (1AM).

Protocol	Cycle	Download		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	115	150	12	2	138	[128;147]	147	[144;149]
	2	152	143						
	3	142	**165**						
	4	121	149						
	5	**167**	88						
VoIP-H323	1	**130**	143	5	2	129	[128;129]	142	[139;144]
	2	120	70						
	3	**130**	139						
	4	129	146						
	5	74	**150**						
FlashVideo	1	**134**	140	4	2	132	[130;133]	137	[135;138]
	2	133	135						
	3	**134**	**154**						
	4	130	98						
	5	88	136						
HTTP	1	**150**	67	4	1	147	[145;148]	140	[138;141]
	2	146	139						
	3	reset	141						
	4	145	**142**						
	5	91	131						
SIP	1	**135**	142	9	2	132	[130;133]	133	[126;139]
	2	122	**182**						
	3	83	138						
	4	130	120						
	5	131	82						
RTSP	1	**159**	**128**	5	2	147	[145;148]	125	[124;125]
	2	145	125						
	3	62	101						
	4	148	122						
	5	150	125						

Table E.16: The results of measurements made on the *NettoKOM's* EDGE mobile network in the download direction during the daytime (9AM –10AM).

Protocol	**Cycle**	**Upload**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	**81**	**62**	10	2	45	[43;46]	44	[42;45]
	2	46	47						
	3	45	45						
	4	16	12						
	5	46	42						
VoIP-H323	1	34	**49**	11	2	44	[43;44]	45	[44;45]
	2	**58**	45						
	3	44	45						
	4	44	46						
	5	45	22						
FlashVideo	1	**81**	47	11	2	45	[42;47]	45	[43;46]
	2	80	44						
	3	46	44						
	4	27	30						
	5	45	**56**						
HTTP	1	**52**	16	11	1	47	[44;49]	46	[45;46]
	2	46	46						
	3	timeout	46						
	4	44	**48**						
	5	30	45						
SIP	1	45	45	12	2	45	[44;45]	45	[44;45]
	2	44	32						
	3	**46**	46						
	4	25	45						
	5	46	**50**						
RTSP	1	**39**	23	4	2	38	[37;38]	43	[42;43]
	2	37	42						
	3	19	**44**						
	4	38	43						
	5	31	**44**						

Table E.17: The results of measurements made on the *NettoKOM's* EDGE mobile network in the upload direction during the daytime (9AM –10AM).

Protocol	Cycle	Download		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	**187**	**190**	8	2	184	[181;186]	188	[186;189]
	2	186	163						
	3	147	148						
	4	180	189						
	5	110	186						
VoIP-H323	1	126	137	9	2	180	[177;182]	138	[126;149]
	2	181	120						
	3	**183**	115						
	4	122	**187**						
	5	176	158						
FlashVideo	1	183	96	8	2	178	[173;182]	178	[173;182]
	2	183	178						
	3	170	125						
	4	**187**	171						
	5	167	**186**						
HTTP	1	26	170	9	2	180	[176;183]	165	[159;170]
	2	178	125						
	3	186	**190**						
	4	**187**	118						
	5	172	161						
SIP	1	164	11	9	2	179	[174;183]	182	[180;183]
	2	**187**	170						
	3	185	**183**						
	4	170	180						
	5	182	**183**						
RTSP	1	128	183	6	2	121	[117;124]	187	[184;189]
	2	**166**	126						
	3	119	187						
	4	151	127						
	5	117	**191**						

Table E.18: The results of measurements made on the *NettoKOM's* EDGE mobile network in the download direction during the nighttime (12AM – 1AM).

Protocol	Cycle	Upload		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	41	**46**	5	2	46	[45;46]	45	[44;45]
	2	**46**	**46**						
	3	**46**	42						
	4	**46**	42						
	5	**46**	**46**						
VoIP-H323	1	41	46	12	2	46	[45;46]	46	[45;46]
	2	**46**	**47**						
	3	**46**	45						
	4	**46**	46						
	5	**46**	**47**						
FlashVideo	1	45	**46**	2	2	46	[45;46]	45	[44;45]
	2	**46**	44						
	3	**46**	5						
	4	**46**	45						
	5	45	45						
HTTP	1	**67**	**94**	11	2	45	[43;46]	45	[43;46]
	2	45	46						
	3	46	46						
	4	46	47						
	5	12	44						
SIP	1	16	32	6	2	45	[44;45]	45	[43;46]
	2	**46**	44						
	3	44	16						
	4	**46**	**46**						
	5	**46**	**46**						
RTSP	1	43	46	12	2	45	[44;45]	45	[44;45]
	2	45	46						
	3	**46**	**93**						
	4	42	29						
	5	**46**	44						

Table E.19: The results of measurements made on the *NettoKOM's* EDGE mobile network in the upload direction during the nighttime (12AM – 1AM).

Protocol	Cycle	Download		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	52	**54**	6	2	52	[51;52]	53	[52;53]
	2	**53**	51						
	3	52	**54**						
	4	50	**54**						
	5	**53**	53						
VoIP-H323	1	52	**54**	6	2	51	[50;51]	53	[52;53]
	2	52	50						
	3	51	**54**						
	4	49	53						
	5	**53**	53						
FlashVideo	1	54	**53**	4	2	54	[53;54]	52	[51;52]
	2	54	51						
	3	51	**53**						
	4	**56**	**53**						
	5	55	52						
HTTP	1	52	52	11	2	53	[52;53]	53	[52;53]
	2	**55**	**55**						
	3	54	54						
	4	54	54						
	5	51	53						
SIP	1	**54**	53	11	2	54	[53;54]	54	[53;54]
	2	**54**	55						
	3	53	54						
	4	**54**	**56**						
	5	53	54						
RTSP	1	54	52	0	2	53	[52;53]	53	[52;53]
	2	**55**	**53**						
	3	53	**53**						
	4	53	**53**						
	5	53	**53**						

Table E.20: The results of measurements made on the *NettoKOM's* HSPA mobile network in the download direction during the daytime (10AM) after the user crossed the data limit.

Protocol	**Cycle**	**Upload**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	23	21	7	2	25	[23;26]	24	[23;24]
	2	22	23						
	3	**28**	**25**						
	4	27	**25**						
	5	27	24						
VoIP-H323	1	25	**25**	4	2	25	[24;25]	25	[24;25]
	2	25	24						
	3	**27**	**25**						
	4	25	**25**						
	5	24	**25**						
FlashVideo	1	**27**	23	0	2	27	[26;27]	24	[23;24]
	2	26	24						
	3	26	24						
	4	**27**	25						
	5	**27**	**26**						
HTTP	1	21	22	6	2	25	[24;25]	22	[21;22]
	2	25	22						
	3	26	**25**						
	4	24	21						
	5	**28**	**25**						
SIP	1	23	**27**	6	2	25	[22;23]	22	[24;25]
	2	19	25						
	3	**25**	26						
	4	23	20						
	5	24	25						
RTSP	1	21	25	6	2	23	[22;23]	26	[25;26]
	2	23	**26**						
	3	23	**26**						
	4	**27**	**26**						
	5	25	**26**						

Table E.21: The results of measurements made on the *NettoKOM's* HSPA mobile network in the upload direction during the daytime (10AM) after the user crossed the data limit.

Protocol	**Cycle**	**Download**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	54	**55**	4	2	55	[54;55]	53	[52;53]
	2	**56**	52						
	3	55	53						
	4	54	**55**						
	5	55	53						
VoIP-H323	1	53	**54**	9	2	53	[52;53]	53	[52;53]
	2	52	53						
	3	53	53						
	4	**55**	**54**						
	5	**55**	53						
HTTP	1	**55**	55	9	2	55	[54;55]	54	[53;54]
	2	**55**	**56**						
	3	54	55						
	4	54	53						
	5	**55**	54						
SIP	1	55	52	0	2	56	[55;56]	52	[51;52]
	2	**56**	53						
	3	55	**55**						
	4	**56**	53						
	5	**56**	52						
RTSP	1	53	54	8	2	55	[53;54]	54	[53;54]
	2	**55**	**56**						
	3	**55**	54						
	4	54	54						
	5	54	52						

Table E.22: The results of measurements made on the *ALDITalk's* HSPA mobile network in the download direction during the daytime (11AM) after the user crossed the data limit.

Protocol	Cycle	Upload		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	54	51	5	2	54	[53;54]	53	[52;53]
	2	**55**	**55**						
	3	**55**	54						
	4	53	54						
	5	54	53						
VoIP-H323	1	52	54	5	2	53	[52;53]	54	[53;54]
	2	53	**55**						
	3	**55**	**55**						
	4	53	54						
	5	53	54						
HTTP	1	52	**55**	8	2	53	[52;53]	53	[52;53]
	2	**55**	53						
	3	**55**	52						
	4	54	53						
	5	53	**55**						
SIP	1	**55**	**54**	4	2	54	[53;54]	54	[53;54]
	2	53	**54**						
	3	**55**	53						
	4	54	**54**						
	5	54	**54**						
RTSP	1	54	**55**	10	2	55	[55]	54	[53;54]
	2	**55**	**55**						
	3	54	**55**						
	4	54	**55**						
	5	**55**	**55**						

Table E.23: The results of measurements made on the *ALDITalk's* HSPA mobile network in the upload direction during the daytime (11AM) after the user crossed the data limit.

Protocol	**Cycle**	**Download**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	59	**62**	11	2	59	[58;59]	59	[58;59]
	2	**61**	60						
	3	59	59						
	4	58	58						
	5	59	59						
VoIP-H323	1	56	**63**	9	2	60	[59;60]	60	[59;60]
	2	60	60						
	3	**62**	58						
	4	60	60						
	5	60	60						
HTTP	1	59	61	11	2	59	[58;59]	59	[58;59]
	2	58	**63**						
	3	59	59						
	4	60	58						
	5	**62**	59						
SIP	1	59	58	10	2	61	[59;62]	59	[58;59]
	2	57	60						
	3	62	59						
	4	63	**63**						
	5	**63**	60						
RTSP	1	**63**	57	5	2	60	[59;60]	58	[57;58]
	2	60	58						
	3	60	58						
	4	58	**63**						
	5	61	59						

Table E.24: The results of measurements made on the O_2*'s* HSPA mobile network in the download direction during the daytime (2PM) after the user crossed the data limit.

Protocol	**Cycle**	**Upload**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
		RF	**PF**						
BitTorrent	1	**29**	**25**	7	2	24	[23;24]	24	[23;24]
	2	24	**25**						
	3	23	24						
	4	25	24						
	5	25	24						
VoIP-H323	1	**26**	**26**	3	2	26	[25;26]	26	[25;26]
	2	**26**	**26**						
	3	**26**	25						
	4	25	25						
	5	**26**	**26**						
HTTP	1	22	**25**	8	2	24	[23;24]	25	[24;25]
	2	25	23						
	3	24	24						
	4	25	**25**						
	5	**26**	**25**						
SIP	1	**25**	**25**	3	2	25	[24;25]	24	[23;24]
	2	24	24						
	3	**25**	24						
	4	24	24						
	5	**26**	22						
RTSP	1	24	**27**	6	2	25	[24;25]	26	[25;26]
	2	24	25						
	3	**25**	26						
	4	**25**	26						
	5	**25**	26						

Table E.25: The results of measurements made on the O_2*'s* HSPA mobile network in the upload direction during the daytime (2PM) after the user crossed the data limit.

Protocol	**Cycle**	**Download**		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$	**Time**
		RF	**PF**							
BitTorrent	1	760	752	9	2	745	[737;752]	772	[750;793]	8 PM
	2	730	978							
	3	739	**987**							
	4	754	793							
	5	**953**	652							
VoIP-H323	1	610	**989**	5	2	684	[682;685]	703	[698;707]	8 PM
	2	**1050**	698							
	3	685	710							
	4	683	702							
	5	685	797							
FlashVideo	1	5744	**6133**	7	2	5797	[5792;5801]	6095	[6087;6102]	3 AM
	2	5802	6045							
	3	**6089**	6089							
	4	6047	5748							
	5	5793	6102							
HTTP	1	5833	6028	11	2	5841	[5832;5849]	6043	[6033;6052]	3 AM
	2	5857	6046							
	3	**6078**	**6057**							
	4	5833	1015							
	5	23	5703							
SIP	1	1420	1647	10	2	1509	[1453;1564]	1594	[1544;1643]	3 PM
	2	1408	**2028**							
	3	**2023**	1497							
	4	1612	1640							
	5	1596	1408							
RTSP	1	1532	1647	6	2	1497	[1487;1506]	1495	[1476;1513]	3 PM
	2	1934	**1438**							
	3	**2088**	1550							
	4	1514	1480							
	5	1487	1474							

Table E.26: The results of measurements made on the *MTS's (Belarus)* HSPA mobile network in the download direction.

Protocol	Cycle	Upload		U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$	Time
		RF	**PF**							
BitTorrent	1	**2001**	1403	12	2	1979	[1965;1992]	1898	[1890;1905]	8 PM
	2	5	**2002**							
	3	1548	1756							
	4	1978	1893							
	5	1958	1903							
VoIP-H323	1	1789	2019	12	2	1691	[1565;1816]	1940	[1869;2010]	8 PM
	2	**2104**	1800							
	3	1442	2002							
	4	1842	**2319**							
	5	1331	1420							
FlashVideo	1	**1932**	**1904**	11	2	1839	[1815;1862]	1865	[1844;1885]	3 AM
	2	1801	1834							
	3	1883	1859							
	4	1402	29							
	5	1833	1410							
HTTP	1	**2169**	**1822**	3	2	1903	[1876;1929]	1782	[1756;1807]	3 AM
	2	1877	1789							
	3	1956	1735							
	4	1878	629							
	5	1566	1432							
SIP	1	2137	1755	11	2	2172	[2136;2207]	2236	[2123;2348]	3 PM
	2	1911	2012							
	3	1944	4							
	4	**2352**	2334							
	5	2207	**2363**							
RTSP	1	**1995**	**2400**	3	2	1929	[1902;1955]	1999	[1980;2017]	3 PM
	2	1922	1972							
	3	1888	2035							
	4	1977	1990							
	5	1777	2093							

Table E.27: The results of measurements made on the *MTS's (Belarus)* HSPA mobile network in the upload direction.

Direction	**Cycle**	**RF**	**PF**	U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
Download	1	1846	966	12	2	1846	[1843;1848]	1788	[1725;1850]
	2	**1850**	1217						
	3	1842	**1852**						
	4	1810	1849						
	5	842	1664						
Upload	1	1846	966	10	2	855	[843;874]	859	[849;860]
	2	**1850**	1217						
	3	1842	**1852**						
	4	1810	1849						
	5	842	1664						

Table E.28: The results of the BitTorrent protocol measurements made on the *LUXGSM's (Luxemburg)* HSPA mobile network during the daytime (12PM).

Direction	**Cycle**	**RF**	**PF**	U	U_c	R_m	$R_{interval}$	P_m	$P_{internal}$
Download	1	**335**	341	6	2	309	[302;315]	326	[318;333]
	2	317	294						
	3	316	**349**						
	4	280	319						
	5	296	320						
Upload	1	955	**1023**	11	2	309	[843;874]	326	[849;860]
	2	1003	1001						
	3	**1017**	1009						
	4	856	767						
	5	985	975						

Table E.29: The results of the HTTP protocol measurements made on the *AT&T's (USA)* HSPA mobile network during the daytime (3PM).