Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

Efficient and Accurate Hop-by-Hop Capacity Estimation

Bakar Andguladze

Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

Efficient and Accurate Hop-by-Hop Capacity Estimation

Effiziente und Genaue Hop-by-Hop Kapazitätsabschätzungen

Author:	Bakar Andguladze
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Simon Bauer, M. Sc.
	Benedikt Jaeger, M. Sc.
Date:	November 15, 2021

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, November 15, 2021

Location, Date

Signature

Abstract

This thesis proposes a hop-by-hop capacity estimation methodology with active probing.

We use the passive capacity measurement tool PPrate, which is based packet-pair technique and is used for end-to-end capacity estimations.

We have developed a traffic generation tool with adjustable Time-To-Live values, which enables us to generate traffic to each router on the network path one by one and take advantage of the ICMP messages received back at the host.

We further process the received time-exceeded ICMP messages, derive inter-arrival times from this data and estimate capacities of each hop with PPrate using these inter-arrival times.

Subsequently we evaluate the methodology in empty networks without any flow interference, which shows good results in most of the cases.

Then we conduct experiments in a busy path congested with cross-traffic.

Our technique, along with desired results, has shown some significant flaws too.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This master thesis presents the capacity estimation method for hop-by-hop measurements. Our goal is to create and test a method of calculating the capacity (i.e. maximum transmission rate) in a given network path and locating the narrow link. We will test the developed tool in regard to various metrics, such as packet size, intrusion rate, cross-traffic and flow-interference and finally, analyze the test results in order to conclude whether our approach is actually capable of delivering accurate results efficiently.

1.1 Motivation

As the Internet is becoming an increasingly essential part of our day-to-day life, it is even more important for Internet providers to enhance the quality of networks in order to create a better user experience. One of the means to achieve this goal is to have a better picture of the network they aim to improve. Therefore, we are going to create a tool that measures the capacity of the path between two hosts. The intended field of application is, for instance, traffic analysis and network monitoring, which provides the ground for enhancing network performance.

There are quite a few capacity estimation methodologies and tools available, that will be discussed in chapters below. However, the current State-of-the-Art methods have some significant flaws and limitations regarding our measurement goals, such as

- High intrusion, which can lead to network overload,
- Dependence on ongoing traffic, which can be unpredictable at times,
- Inability to locate the narrow link on the path.

Therefore, our goal is to develop a new solution that tries to minimize the influence of these limitations in regard to our measurement objectives. We will try to implement the least possible intrusion without compromising the accuracy. Also our tool will be able to find the first narrow link of the path by measuring the capacity of each hop in the network.

This new solution will be tested and evaluated in comparison to the results of existing capacity measurement tools, i.e. PPrate implementation by Patryk Brzoza[x], but in contrast to Brzoza's passive approach our tool will be based on an active measurement methodology. Moreover Brzoza's measurement tool was designed to estimate end-to-end capacity, while this thesis is concerned about measuring the capacity of each hop in the network and finding the narrow link, as hop-by-hop measurements provide a better picture of a network and enable to take a closer look at potential issues.

1.2 Research Questions

This thesis is supposed to answer the following research questions:

- **How to measure network capacity hop-by-hop?**
  In order to measure the path capacity hop-by-hop, we need to estimate capacities to each router on the path until the destination host is reached.
- **How to optimize the trade-off between accuracy and intrusiveness regarding large-scale measurements?**
  Certain level of intrusion into the network will be necessary for the measurements. However, there is an important factor to consider: the research shows that too high intrusion could disrupt the traffic in the network and too low could lead to unreliable results. Therefore an optimal trade-off has to be found: What will be the optimal amount of packets to send to each router to get correct results?
- **How robust is the proposed solution regarding the handling of flow-interference, such as cross-traffic?**
  Real networks are usually quite complex and different challenges might arise when we are trying to measure the path capacity. We need to find out whether our solution is feasible when it faces cross-traffic and analyze its effects on the final results.
- **Are we able to locate the capacity bottlenecks of a network?**
  We are interested to find the location of the weakest link in the given network. This can be achieved by finding the capacities to each hop. The weakest link will

1.3 Outline

The remainder of this thesis is structured as follows:
Chapter 2 defines the necessary terminology for understanding this thesis. Namely, capacity, TCP, ICMP, Raw sockets, etc.
Chapter 3 describes the related work: what has been done in regard to capacity estimations and some drawbacks and limitations to the existing approaches.
Chapter 4 describes the approach and the tool that we have developed to implement it.
Chapter 5 reviews the test setup and test environment in which the experiments are conducted.
Chapter 6 evaluates our approach based on several parameters. It reviews the different factors that might affect the measurements and to what extent.
Finally, chapter 7 concludes our thesis and subsequently discusses the future work - what can be done afterwards to further extend our methodology.

Chapter 2

Background

The following chapter provides the basic high-level background knowledge necessary for understanding the subsequent chapters and also defines the important terminology that is relevant for our work.

2.1 OSI Model and Network Layers

In order for the devices from all over the world to properly communicate and exchange data there is a necessity for communication models. The official standardized version of such models is the Open Systems Interconnection(OSI) model, developed by International Organization for Standardization[1]. It consists of 7 layers each of them having a specific purpose. However OSI model is only conceptual and it differs from the architectures that are actually used by networks. One of such architectures is Internet Protocol Suite, also known as TCP/IP. The figure 2.1 visualizes the layers of both of these models.

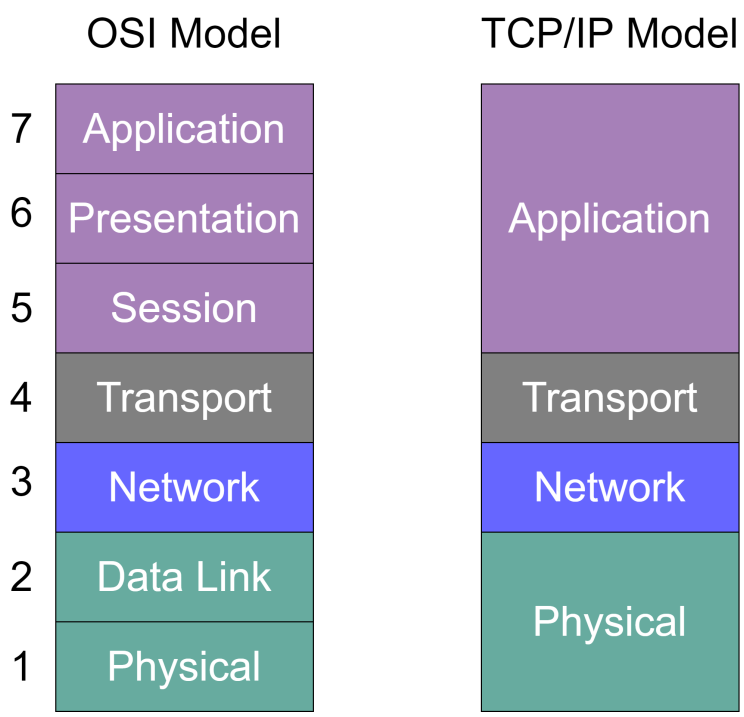

Figure 2.1: Comparison of OSI and TCP/IP Models

We will shortly describe each OSI layer as defined in "Networking - the Complete Reference"[2].

- The Physical and Data Link layers are responsible for the physical communications between devices, i.e. transporting data bits from one machine to another.
- The Network layer functions as a carrier of messages from the source system to the final destination. It provides end-to-end addressing, routing and error checking.
- The Transport layer takes care of packet segmentation and reassembling once they reach the destination. Port numbers are specified at this layer.
- The Session, Application and Presentation layers are used for user interaction through software applications.

Sending data in the Internet is operated from top down through these layers and receiving goes bottom up.

The scope of this thesis are the transport and the network layers and the protocols that we are focusing on are Internet Protocol(IP) and Internet Control Message Protocol(ICMP) from the Network layer and Transmission Control Protocol and User Datagram Protocol from the Transport layer.

We will further refer to the Transport layer as layer 4 (L4) and the Network layer as layer 3 (L3).

2.2 Network Layer Protocols

2.2.1 Internet Protocol (IP)

For the elaboration about L3 and L4 protocols we will use the definitions by Internet Engineering Task Force (IETF)[3].

As mentioned above, the Layer 3 is responsible for end-to-end addressing and routing. The most common L3 protocol is IP. The IP protocol transports the data packets from one host to another and in order to accomplish this goal it encapsulates TCP or UDP packets provided by the Transport layer. The figure 2.2[4] represents the structure of IP packet headers.

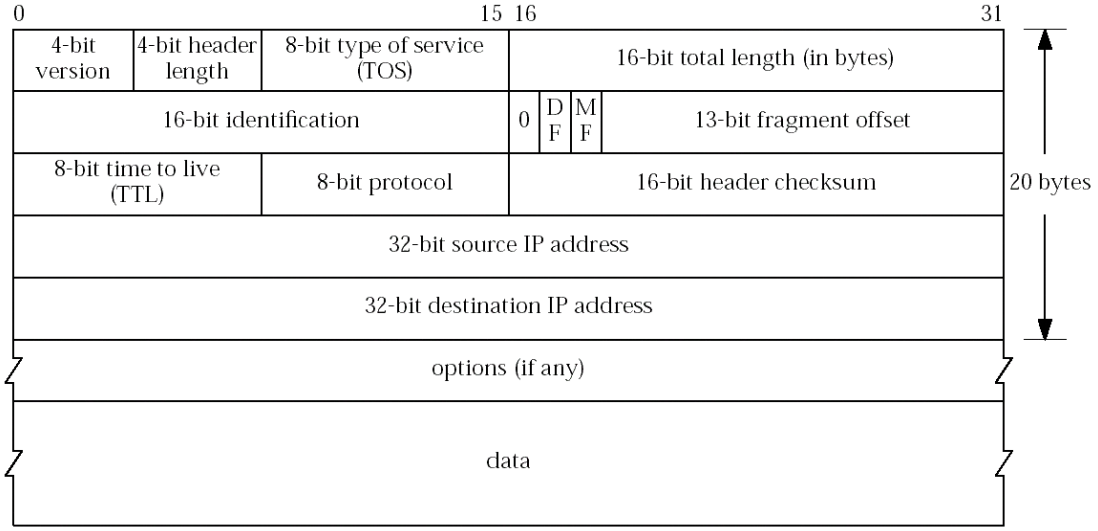

Figure 2.2: Structure of IP header

As it can be seen, there is an 8-bit time to live (TTL) field in the IP header. It is implemented to control the maximum time that a packet is allowed to stay in a network. This is necessary in order to avoid the looping of a packet between hosts. Each IP packet is assigned a certain TTL value and it decrements after each router that a packet passes through. If a packet is unable to reach the final destination before ttl equals zero it will be discarded and an error message will be returned to the sender.
This thesis is built upon taking advantage of this functionality to use it for accomplishing our goal.

2.2.2 Internet Control Message Protocol (ICMP)

Another Network layer protocol that is crucial for our work is icmp. It is used for delivering control messages back to the sender whenever a problem arises in a network[5]. As there exits a multitude of types of problems in the Internet ICMP packet header contains a Type field to classify the issues for the sender. The table 2.1 provides the incomplete list of ICMP message types.

Message type	Description
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect
8	Echo Request
11	Time Exceeded
12	Parameter Problem
13	Timestamp Request
14	Timestamp Reply
15	Info Request
16	Info Reply
17	Address Mask Request
18	Address Mask Reply

Table 2.1: ICMP message types [6]

Our object of interest are type 11 time exceeded ICMP messages. This is the type that provides feedback when the TTL field of an IP address reaches the value of 0.
We will describe how we use this feature in more detail in the Implementation section.

2.3 Transport Layer Protocols

2.3.1 Transmission Control Protocol (TCP)

Transmission Control Protocol serves as a highly reliable data transfer protocol which guarantees that all the packets sent from the source host will reach the destination. It is used in services where the reliability is a requirement, e.g. sending emails or transferring files.
This reliability is provided by the connection establishment process called three-way handshake. During this process the source host sends a packet with a `SYN` synchronization flag and the destination host responds with a packet with an `ACK` acknowledgement

flag. In turn the first host also returns a packet with an `ACK` flag. Once the connection is established the source can send more packets to the end host.

To further ensure the reliability, the sender host requires an `ACK` signal for every packet it sends. For this it assigns each packet a sequence number and expects a corresponding acknowledgement for each such packet. If the `ACK` signals are not received, packets with respective sequence numbers are sent again[7].

2.3.2 User Datagram Protocol (UDP)

Another famous Transport layer protocol is the User Datagram Protocol. Unlike TCP, which is known for its reliability, UDP is less reliable. But on the other hand it is aiming to transfer data from one host to another with minimal protocol mechanism[8]
UDP is used in situations where the certain rate of packet loss can be tolerated, such as video streaming.

2.4 Terminology

Certain terminology in our field of research can be used with different meanings, therefore we first need to state that this thesis will be using the definitions provided by Prasad et al in their paper "Bandwidth estimation: metrics, measurement, techniques and tools"[9], as it appears to be more widespread and accepted. Moreover, those are the definitions also used by other researchers at the chair, therefore, it is more practical to use the common language.

Prasad et al[9] introduce the following three metrics: capacity, bandwidth and bulk transfer capacity(BTC) and capacity is the primary focus of our work. Moreover, they distinguish between segments and hops. The former being the link at the data link layer (L2) and the latter - the links at the IP layer (L3).

Furthermore, they define segments as physical links between devices or shared access local area networks, while hops as sequences of one or more segments connected through Layer 2 devices. Additionally they define end-to-end path from a source host to a destination host as a sequence of hops.

Capacity

Prasad et al. define the capacity C_i of a hop i as a maximum possible amount of bits that can be transferred at that hop per second[9]. If a path consists of several hops, the capacity of the whole path equals that of the link with the lowest capacity:

$$C_{min} = \min_{i=1,..,n} C_i$$

C_{min} is referred as a narrow link. Prasad et al.[9] specify that they avoid the term "bottleneck" as it is interchangeably used for both minimal capacity and minimal available bandwidth, however we will use "bottleneck" to denote narrow link in this thesis.

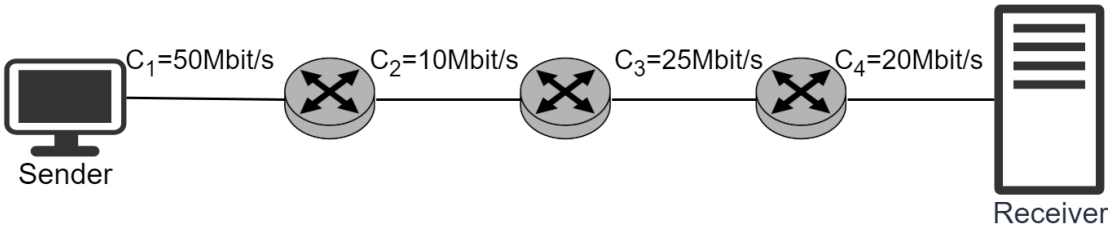

Figure 2.3: Sample network path and link capacities

The figure 2.3 presents a sample network path with four hops and the second hop C_2 as a bottleneck. The end-to-end capacity of this path will equal that of C_2, i.e. 10 Mbit/s

Available Bandwidth

When working with network capacities it is also very important to understand the concept of available bandwidth of a link. Unlike capacity it is not a constant value and it refers to the unused, unconsumed capacity of a link at a certain moment of time[9].

2.5 Traffic Generation

Throughout our work it will be required to artificially generate traffic between hosts in a network and subsequently capture and analyze it. In this section we will briefly tackle several tools and methodologies that will be later used to achieve our goals.

Raw Sockets

Network socket is the software used for data traffic between the hosts. A TCP packet consists of an IP header, a TCP header and a chunk of data it is supposed to deliver to another host[10].
The default type of sockets has TCP and IP headers automatically added, however in certain cases, including our research, it is necessary to configure the headers manually with an application. In such cases raw sockets have to be used.
In this thesis we use raw sockets for traffic generation as it provides the ability to

manipulate the `time to live` field of IP header, which will show to be a crucial part for our research.

iPerf

Another powerful tool used in this thesis is iPerf[11]. It is mainly used for the maximum achievable bandwidth estimation in IP networks, however it incorporates multitude of other features.
We will be using this tool for additional packet generation to emulate the cross-traffic in our test environment.

Chapter 3

Related Work

3.1 Capacity Estimation

There have been various researches conducted on capacity and bandwidth estimation and there are different approaches and tools in existence. In this section we will briefly categorize capacity estimation tools and discuss their differences, advantages and disadvantages.

3.1.1 Types of Capacity Estimation Tools

In his paper "Through the Diversity of Bandwidth-Related Metrics, Estimation Techniques and Tools: An Overview" Abut[12] defines several characteristics of capacity estimation tools.

First, he classifies them into *active* and *passive* ones. Active tools conduct their estimations by injecting certain amount of test packets in the target network. In contrast, passive tools rely on observation of the traffic in real networks and analyzing it without interfering in any way.
It is also important that active tools can be further subdivided based on the amount of traffic load they cause in networks they are trying to measure. Active tools are called intrusive if impose high pressure on a network and non-intrusive otherwise.

Furthermore, capacity estimation tools can deliver end-to-end or hop-by-hop results. End-to-end estimation tools are only able to tell the capacity of the whole path and are only able to locate the narrow link if it is located at either end of a path.
Hop-by-hop estimation tools, on the other hand, are able to estimate capacities of each hop in a path[12]

Lastly, capacity measurements between two hosts can be conducted from either one end or both ends, which further classifies tools into single-ended and both-ended types[12]

3.2 Existing Approaches

3.2.1 Variable Packet Size

Variable Packet Size (VPS) probing is one of simple active hop-by-hop capacity estimation methods. The concept of this methodology is to send Time-To-Live exceeding packets to each hop on the path up until the destination and measure the Round Trip Time (RTT) values of the ICMP time exceeded packets received from those hops[9].

Such approach however is vulnerable to queuing delays at intermediary hosts that can occur independently from the sender. According to Prasad et al.[9] in order to handle this issue, VPS sends multiple probe packets of a fixed size to each router on the path and gather with the expectation that at least one will survive delays.

As a result, the minimum of these RTTs should be measured. Prasad et al[9] provide the equation for this measurement:

$$T_i(L) = \alpha + \sum_{k=1}^{i} \frac{L}{C_k}$$

where T_i is the minimum RTT, L is a packet size, α denotes delays up to i-th hop and C_k - capacity of the k-th link.

3.2.2 Packet Pair Dispersion

Another very important capacity estimation method is Packet Pair/Train Dispersion. In contrast to Variable Packet Size it is an end-to-end technique. The main idea of PP is to send a same-sized packet pair to the target host. Once they encounter the narrow link the time difference between them after they pass through will be

$$\Delta t = \frac{s}{C_{bottleneck}}$$

where s denotes the packet size and C - capacity of the bottleneck link. The time difference is also called 'dispersion'. Moreover, for valid results packet pairs must follow same paths and need to be served by First in, First out principle[12]

Packet pair dispersion method can be modified by sending packet trains instead of just two packets in order to increase the accuracy, however the research of Dovrolis et al.

reject this hypothesis by stating that it leads to asymptotic dispersion rate, which has a value between the capacity and available bandwidth[13].

3.2.3 PPrate

One of the most prominent examples of packet pair dispersion based tools is PPrate, proposed by En-Najjary and Urvoy-Keller[14]. It is a passive end-to-end capacity estimation algorithm which takes inter-arrival times of packets as an input along with a packet size and has high accuracy rates.
However, it is only able to provide end-to-end estimation results and does not give us

In the next chapters we will use the PPrate implementation of Brzoza[15] to find out whether this tool can be used for hop-by-hop estimations by passing ICMP packet inter-arrival times to it and evaluate the results.

Chapter 4

Implementation

The goal of this thesis is to combine the advantages of passive and active capacity estimation tools and possibly eliminate their flaws in an attempt to get as accurate estimation results as possible.

We have set the requirements that should be met in order to consider our goal achieved: Firstly, the tool should minimize the intrusion into the network we are trying to measure. Secondly, the tool must deliver the accurate results despite the obstacles and challenges it might have to face during measurements, such as flow interference.

4.1 Approach

The basic idea of the proposed methodology is to send multiple TCP packets from the source host to the destination host and measure the capacity of the path to each router that the generated packets pass.
Every packet sent from the source has the `time to live` value adjusted in a way that it expires when the packet reaches the targeted router.
As a consequence each router generates ICMP time exceeded messages and sends them back to the original sender. The captured inter-arrival times are then passed to the PPrate algorithm and it calculates the capacity from the Sender host to each router respectively.

The first router with the smallest capacity will be the end of the bottleneck link. To be more specific, our method does not guarantee to estimate the exact capacity of each link, rather the capacity of the path from the source to each router. For example, in the figure 4.1 where a sample topology is depicted, the first link is the narrow one. Our

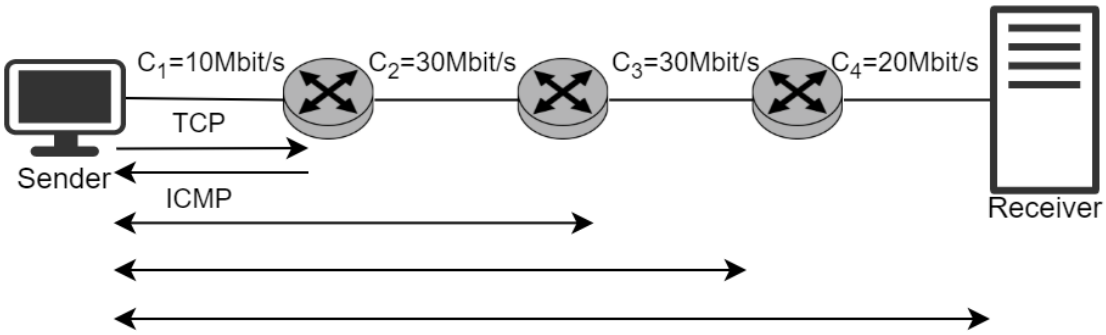

Figure 4.1: The basic idea of the capacity estimation method

method will return the capacity of 10 for each hop instead of the original 30, 30 and 20 respectively. This also serves as an indication on the location of the narrow link. Moreover, if there are multiple narrow links on the path, our tool in only able to locate the first one.

In order to calculate the capacity of the last link, we incorporate Brzoza's[15] framework into our program, which estimates capacity end-to-end. This is necessary in order to find out whether the last hop is the narrowest link or not.

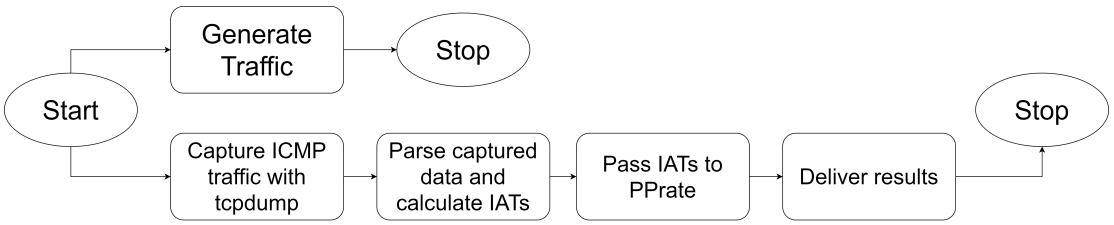

Figure 4.2: Flow of the capacity estimation tool

The basic flow of our approach as shown in the figure 4.2 is executed as follows:

- The tool generates TCP packets and sends them through the path directed to the destination host via raw sockets. Each packet has a time-to-live value adjusted the way that they expire at the targeted router so that ICMP messages are generated and sent back to the source.
- In parallel `tcpdump`[16] command is listening and capturing all the ICMP packets that return to the sender host.
- We aim a configurable amount of packets at each router and once the traffic generator covers all the hops on the path, it stops.

- After capturing all the ICMP time exceeded packets the program parses them and calculates inter-arrival times separately for each hop.
- These inter-arrival times are then passed to the PPrate algorithm which calculates the capacities from the host to each router on the path.

As this thesis builds upon the work by P. Brzoza[15] and uses his PPrate implementation in Python, as well as other important parts of his work, we also find it reasonable and more importantly practical to also implement our work in Python, which in turn is a very powerful tool to work with data analysis. Additionally, we implement the traffic generation part in `C` programming language as it provides the ability to manually build data packets via raw sockets.

4.2 Traffic Generation

In this section we will briefly explain how the traffic generation works in our tool.

We use raw sockets because they enable to manually build a packet and therefore also configure an IP header of the packet.

Listing 4.1: Creation of a packet[10]

```
1  // Creating a packet
2  char packet[4096] , source_ip[32] , *data , *pseudogram;
3
4  //zero out the packet buffer
5  memset (packet, 0, 4096);
6
7  //IP header
8  struct iphdr *iph = (struct iphdr *) packet;
9
10 //TCP header
11 struct tcphdr *tcph = (struct tcphdr *) (packet + sizeof (struct ip));
12 struct sockaddr_in sin;
13 struct pseudo_header psh;
14
15 //Data part
16 data = packet + sizeof(struct iphdr) + sizeof(struct tcphdr);
```

As mentioned in section 2.2.1, the IP header contains a field `Time to live (TTL)`, which has to be controlled by the sender so that it can target each router on the path. Therefore, we first assign the value of 1 to the `iph->ttl` field in order to get the ICMP responses from the first router.

```
1  iph->ttl = 1;
```

Finally, the tool generates the traffic with a `while` loop. Depending on the desired packet train length `iph->ttl` is being incremented after the specific amount `n`, which

denotes the train length and then targets the next router on the path until it reaches the final host.
The traffic is terminated once the `iph->ttl` surpasses the number of hops on the path.

Listing 4.2: Generating traffic towards the destination host

```
int i = 0;
while (1)
{
    //Send the packet
    if (sendto(s, packet, iph->tot_len, 0, (struct sockaddr *) &sin, sizeof(sin))
         < 0)
    {
        perror("sendto failed");
    }
    //Data sent successfully
    else
    {
        printf("%d \t", i+1);
        printf ("Packet Sent. Length: %d \n", iph->tot_len);
    }

    // ttl should be incremented after every n packets
    i++;
    if(i == n) // number of packets necessary for measuring
    {
        printf("hop #%d done\n\n", iph->ttl);
        iph->ttl++;
        i = 0;
    }

    if(iph->ttl == routers+2) // total number of routers + 2
    {
        break;
    }
}
```

4.3 Capturing the Traffic and Estimating Capacities

The whole process described in the section 4.2 is executed in parallel with the `tcpdump` command which serves as an ICMP listener and captures every ICMP packet generated by the routers. For that we need to adjust `tcpdump` so that it can filter every other type of traffic with the following script:

```
tcpdump -n icmp -w results/icmp.pcap
```

Additionally, we execute another `tcpdump` to capture the TCP traffic separately for end-to-end capacity estimation:

```
tcpdump -n tcp -w results/tcp.pcap
```

The captured data is subsequently converted to `.csv` format with the following `tshark` command:

```
tshark -r results/icmp.pcap -T fields -E header=y -E separator=, -E quote=d -E
    occurrence=f -e frame.time_epoch -e ip.src -e ip.dst -e ip.len > results/icmp
    .csv
```

Finally we extract the arrival times of the ICMP packets grouped by routers that have generated them, calculate the inter-arrival times and pass the lists of IATs to PPrate:

Listing 4.3: Calculating capacities hop-by-hop with PPrate[15]

```
def calculate_capacities():
    pcap_to_csv()
    filepath = dir_path + "/results/icmp.csv"
    streams = {}

    df = read_from_csv(filepath)
    packet_size = get_packet_size()
    group_by_routers(df, streams)
    results = []
    for key in sorted(streams):
        streams[key][0] = calculate_iats(streams[key][0])
        cap = pp.find_capacity(packet_size, streams[key][0]) # PPrate
        cap = bit_to_mbit(cap) # PPrate returns the results in bits.
        streams[key][2] = cap
        results.append(cap)
    return streams
```

In the next section we will describe the experiment results that we have conducted using the proposed approach, evaluate the methodology based on accuracy and decide whether it is applicable in real life or not.

Chapter 5

Evaluation

This chapter presents the results of the evaluation of our approach. The experiments have been conducted based on several test parameters and their combinations and we will be analyzing the baseline parameter sets as well as the key combinations.

The capacity estimation tools can face different challenges in real networks. We will try to imitate some of these challenges in our custom-built emulated network and find out what kind of impact can they have on measurement accuracy.
Two of the biggest obstacles our framework has to face are the cross-traffic and the ICMP rate limiting, which we will discuss shortly in respective sections.

Overall, we will evaluate the tool based on the path length, the capacity range, the packet size, the packet train length. For a deeper insight we will also apply the ICMP rate limiting and also check the influence of the packet loss on accuracy. The tests are conducted on an empty network as well as with the significant amount of cross-traffic load.

The following are the parameters which we use for the evaluation of the tool. They are manually configured by us for each test suite in order to get a better picture about the quality of our approach and shows us whether it is applicable in real life:

- **Path Length** - the number of routers between the source host and the destination host. (i.e. number of hops minus one)
  The default value: 8 routers.
- **Capacity Range** - The range of numbers from which a capacity for each link will be generated
  The default value: [10, 100).

- **Packet Size** - The size of each packet that will be sent from the source host to the destination. It represents the total size of the packet including TCP and IP headers(each weighing 20 bytes).
  The default value: 1500 bytes (i.e. size of MSS + TCP header + IP header = 1460 + 20 + 20)
- **Packet Train Length** - The amount of packets that will be targeted at each hop
  The default value: 300 packets.
- **Packet Loss** - The emulated packet loss on each router.
  The default value: 0.
- **ICMP Rate Limit** - The artificial ICMP rate limit that will be assigned to each router.
  The default value: 0.
- **Cross Traffic** - The coefficient of the cross traffic load. The Optimal value is recommended to be assigned from 0 to 1.0, the value of 0 meaning an empty network without any cross traffic.

To ensure the reliability on the the test results, we conduct test runs with each test configuration 20 times.

5.1 Test Environment and Setup

For the testing purposes we decided to build a virtual network using the network emulator - Mininet[17]. It provides the necessary tools to create an artificial network and has a very handy Python API that enables us to build custom topologies.

The figure 5.1 represents the template of the network that we have used in our experiments. It consists of the source and the destination hosts, the top and the bottom hosts and the routers that connect them with each other.

As a reader can see from the Figure 5.1, the main hosts (sender and receiver) are connected with a sequence of `n` routers, each consisting of four interfaces and each having its own subnetwork. The number of routers and, therefore, the number of top and bottom hosts is dynamic and configurable. The routes from and to all hosts are statically configured.

The measurements are conducted based on the main path, which is the one that consists of the routers (1...n) and connects the sender to the receiver. The top and the bottom hosts are used for generating cross-traffic in later experiments. The cross-traffic inter-

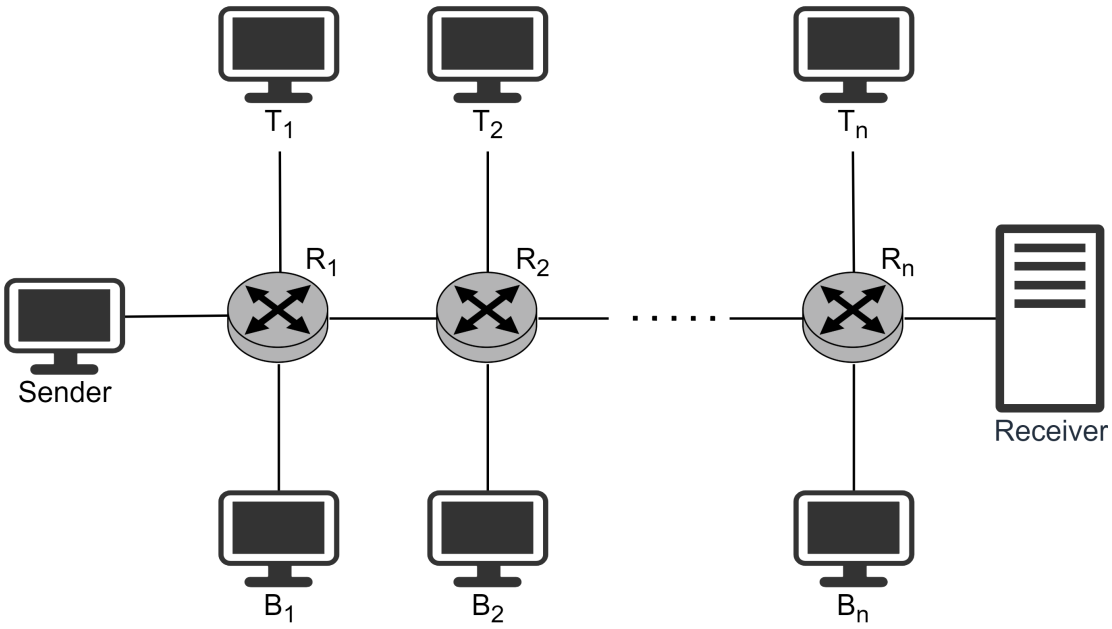

Figure 5.1: The template network topology

feres with the main flow of packets and can have a significant influence on the accuracy of capacity estimations.

The measuring node is the "Sender", as using our tool only makes sense if the measuring node is the one that generates the traffic. All the necessary traffic is captured at this node with the `tcpdump` command and subsequently analyzed by our program that delivers the final result.

Finally, it is also important to mention that Mininet has certain limitations, namely, it is limited with the computational power of the machine it runs on. Therefore, we have decided to execute the estimation experiments on the testbed of the chair.

Disclaimer: As the routers are all configured the same way in terms of packet loss and ICMP rate limiting, the results from all hops are analyzed together. The tests have shown no difference between the accuracy rates of different indexes of hops, e.g. the accuracy rate of the capacity estimations from the source to R_1 and to R_n is practically the same in equal circumstances.

5.2 Evaluation in Empty Network

We decided to divide our evaluation into two parts: the section 5.2 describes the measurement results on an empty network and the section 5.3 analyzes the effect of the cross

traffic based on the experiment outcomes on an overloaded network where the flow of our generated traffic is interfered by the cross-traffic.

To have a broad picture of the initial results, we first conducted baseline measurements with the default configuration of the parameters, namely, the path length of 8 routers and the packet size of 1500 bytes in a capacity range of [10, 100), which resulted in accurate estimations.
The graph in the figure 5.2 represents the comparison between the real and estimated capacities and as one can see, the accuracy is high.
Moreover, the figure 5.3 and the table 5.1 represent the error rate of the baseline measurement and statistical details about error rates respectively.

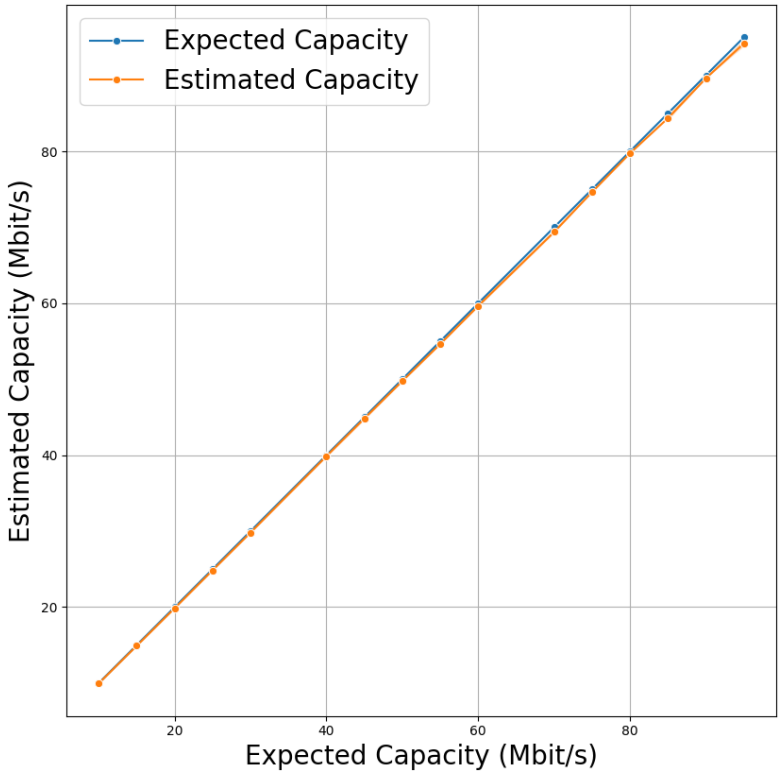

Figure 5.2: The baseline measurement results

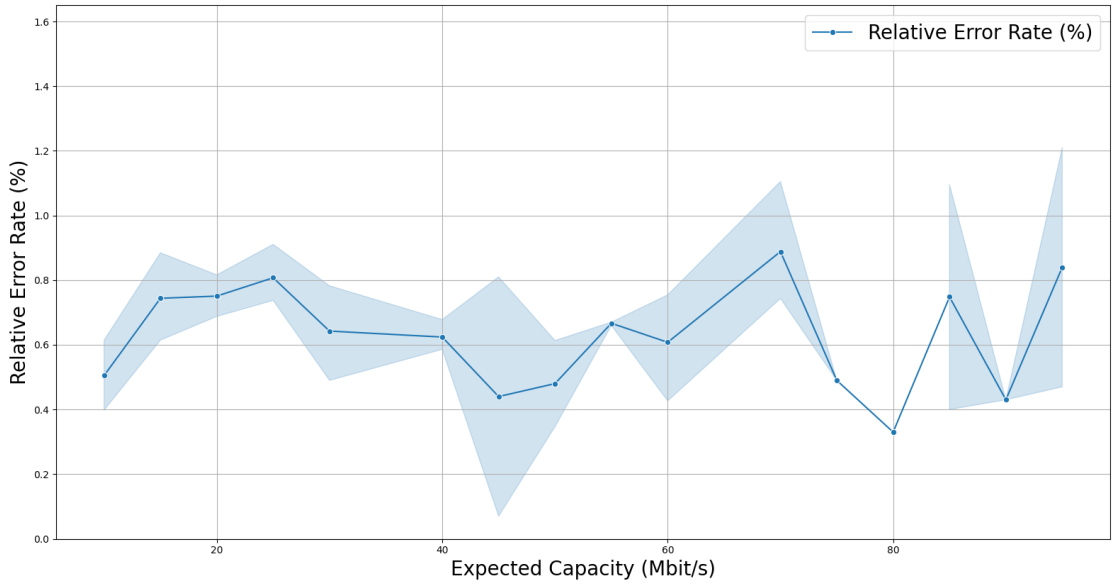

Figure 5.3: The baseline measurement error rate

Average	0.67%
Standard Deviation	0.27%
Min Error	0.07%
Max Error	1.55%

Table 5.1: Relative error statistics of the baseline measurement

The baseline evaluation shows the the good potential of our approach, however it was conducted with the simplest and likely some of the most promising configuration possible, which is not likely to be the case in real networks.
In the next sections we will dissect each parameter through subsequent experiments and check their impact on the accuracy in order to evaluate how robust and useful our capacity estimation methodology can be in real life.

5.2.1 Path Length

The first parameter to be analyzed is the path length from the source host to the sink, i.e. number of routers on the path. In order to analyze the impact that a path length can have on the measurement accuracy, we have to execute the test runs on different values of the given parameter.

As the default TTL value in Linux for TCP and ICMP is 64[18], we decided to conduct measurements on arbitrary numbers of routers up to 63, namely 1, 3, 8, 20, 32 and

63. These numbers, although being chosen randomly apart from 1 and 63, paint a good picture of how our estimation tool reacts on networks with different sizes. Based on the results we can assume that the path length does not have an impact on the accuracy of our capacity estimation framework. Figures 5.4 and 5.5 depict the differences between actual and estimated capacity values on networks with path length of 3 and 63 respectively. (We chose to present only these two graphs, as there are practically no relevant differences between the results of different path lengths)

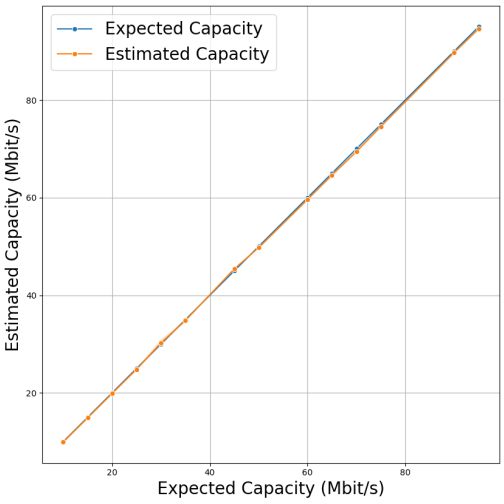

Figure 5.4: Path length: 3 Routers

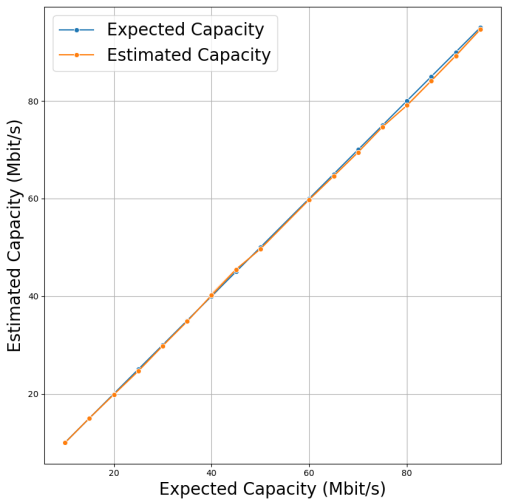

Figure 5.5: Path length: 63 Routers

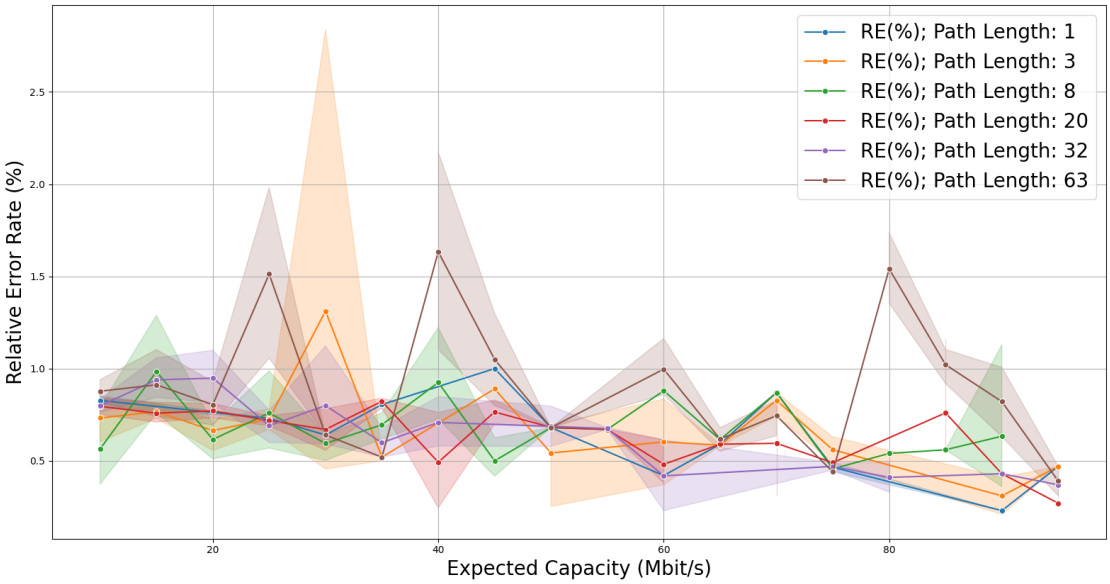

Figure 5.6: Error rates on different path lengths

The table 5.2 depicts the comparison of relative error rates in more details and as one can observe, the differences are practically irrelevant.

Table 5.2: Relative error stats of paths with different lengths.

Path Length	Average	Standard Deviation	Min Error	Max Error
1	0.65%	0.21%	0.23%	1%
3	0.72%	0.5%	0.11%	4.3%
8	0.74%	0.47%	0.01%	2.83%
20	0.75%	0.24%	0.01%	2.87%
32	0.83%	0.48%	0.0001%	4.09%
63	1.23%	2.53%	0.01%	42.57%

Conclusion

Judging by the results of the experiments focusing on the path length conducted in an empty network we can conclude that the number of hops does not affect the accuracy of the estimation tool. At this point it can be safely assumed that when the cross-traffic is out of the picture, our framework can estimate capacity accurately no matter the amount of hosts on the path. Although there is still a small likelihood of major inaccuracies in measurements, such as ≈43% relative error, it does not affect the overall picture, as it can be seen in the graphs.

After testing the path length we will further use 8 routers for the subsequent estimations as a default value.

5.2.2 Packet Size

Our next target parameter is the size of each packet in our generated traffic. As the packet size along with the amount of packets that have to be injected into the network can be the cause of the network overload, we have to find an optimal packet size that most importantly does not cost us the accuracy and also the load imposed on a target network remains minimal. In other words, the goal is to find the least packet size value that will deliver the accurate results.

We launched the tests with the packets of Maximum Transmision Unit (MTU) size, i.e. 1500 bytes and with the packets as small as 100 bytes. As depicted before, the former delivers highly accurate results, the latter, however, has shown an average relative error rate of ≈30%, thus we decided to narrow down between the two values and find out at which packet size does relative error rate remain consistently low, i.e. what is the minimal packet size that matches the accuracy of 1500 bytes.

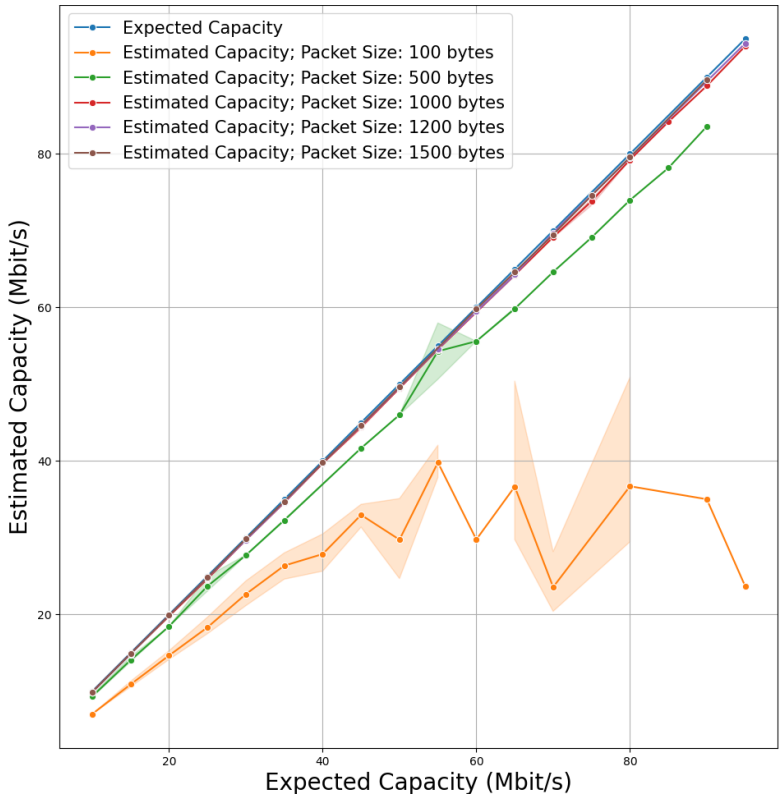

Figure 5.7: Comparison of the measurement results with different packet sizes

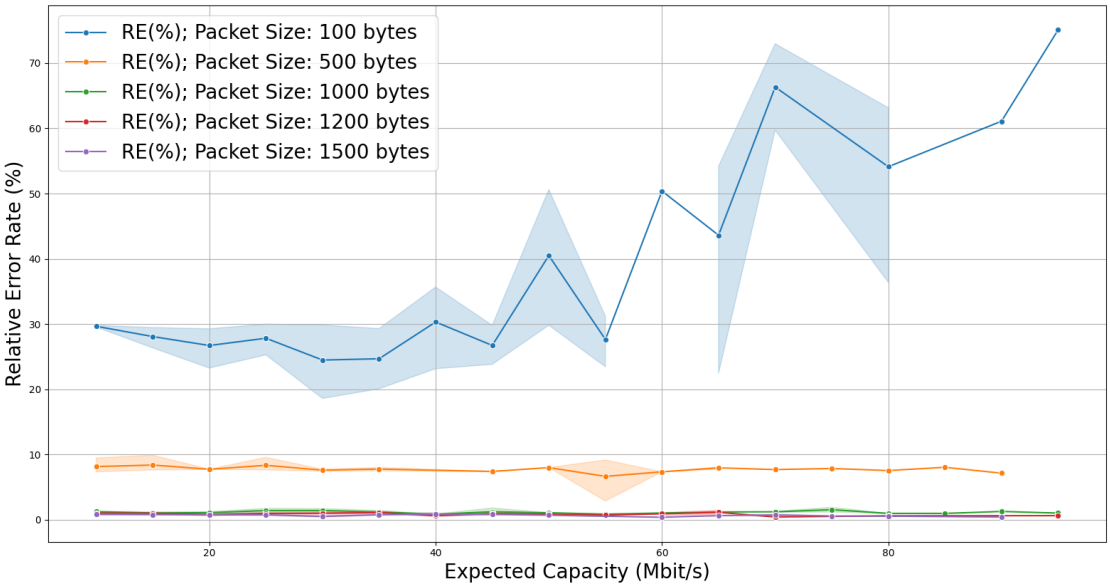

Figure 5.8: Error rates by packet size

Subsequently we tested packet sizes of 500, 1000 and 1200 bytes which showed drastic improvements in estimation accuracy. The results of these experiments are depicted in the figure 5.7 the respective error rates in the figure 5.8.

Depending on a researcher's threshold of the tolerable relative error rate the minimal packet size can vary. Thus we launched several more test runs with packet sizes between 500 and 1000 bytes in order to narrow down even more and to provide more detailed research.
The table 5.3 shows the results of all the tests on packet size.

Table 5.3: Relative error rate statistics for packet size

Packet Size	Average	Standard Deviation	Min Error	Max Error
100	30.23%	10.97%	6.29%	75.24%
500	7.96%	3.11%	1.12%	41.57%
600	2.11%	0.66%	0.1%	4.85%
700	1.8%	0.64%	0.01%	5.92%
800	1.53%	0.47%	0.14%	4.6%
900	1.3%	0.54%	0.07%	5.42%
1000	1.15%	0.48%	0.0%	3.57%
1200	0.93%	0.25%	0.1%	1.72%
1500	0.77%	0.32%	0.13%	2.98%

Conclusion

The estimations show that the impact of the packet size on the accuracy is huge and the packet size of 1500 bytes delivers the most accurate results. The accuracy declines along with the decreasing packet size. However, if we take, for example, less than 2% as an average relative error rate that can be tolerated, we can decrease packet size down to 750 bytes (the half of the original 1500 bytes) and still get highly accurate results.
We should expect, though, that this number might not be sufficient in networks where the cross traffic is the case. This will be discussed in another section.

5.2.3 Train Length

In the original paper of PPrate[14] En-Najjary and Urvoy-Keller state that the second version of the PPrate algorithm has good estimation results with as many as 300 IAT samples. Our previous estimations have also shown that the train length of 300 packets (299 IATs if packet loss is 0) can potentially be enough for accurate results with less than 1% error rate.
But in this thesis we are aiming for the minimal intrusion. Therefore we are interested in decreasing the packet train lengths without compromising accuracy and finding out

the least number of packets that will be provide the estimations as accurate as we have achieved with 300 packets.

The version of the Python implementation of PPrate by Brzoza[15] that we are using for our estimations does not have a constraint on the amount of IAT samples. Thus we experimented on as small numbers of packets as possible.

We conducted tests with different train lengths and the outcome was unexpected: The smallest train length that gave us accurate estimations was only 7 packets, meaning only 6 IATs were sufficient for PPrate to estimate capacity accurately. The algorithm was unable to calculate capacities with less than 6 IATs.

Nevertheless there is one more factor to consider. In our experiments conducted in Mininet we could observe that the packet trains with less than 50 packets had almost no packet loss and increasing the train length often resulted in higher rate of loss without explicitly configuring packet loss. E.g. after 20 experiments with the trains of 7 packets the loss was 0.

Table 5.4: Relative error statistics for packet train length

Train Length	Average	Standard Deviation	Min Error	Max Error
7	1.42%	1.61%	0.01%	10.09%
10	1.25%	1.3%	0.03%	10.78%
20	1.1%	1.0%	0.05%	5.84%
50	0.81%	0.47%	0.01%	4.12%
100	0.78%	0.41%	0.03%	2.95%
250	0.74%	0.3%	0.07%	2.76%
500	0.53%	0.38%	0.05%	2.94%
1000	0.49%	0.43%	0.04%	3.39%
5000	0.42%	0.36%	0.02%	2.72%

The table 5.4 shows shows the statistics of relative error rate depending on the train length. It suggests that with the increase of the train length average error rate decreases, but the difference in estimation results between 7 packet train and 5000 packet train is only 1%.

Conclusion

The test results are showing that PPrate is able to provide high accuracy with a very small number of packets in a packet train when we are running tests on an empty network. Although more importantly it is notable that longer packet trains provide more accurate results.
These experiments, however, are not enough to assess that we can get the similar results

in real networks where the cross-traffic is also an issue. This factor will be tested separately in section 5.3.

5.2.4 Optimal Intrusion

Based on our experiments on empty network we came to a conclusion that the packet size is more crucial parameter to the capacity estimation rather than packet train length. It is necessary that the packet size equals at least 700 bytes in order to keep the relative error rate reasonable, for example, below 2% on average.
When it comes to the train length, however, the situation is more flexible here. Meaning, PPrate can deliver accurate results in empty network with as few as 10 packets. We combined these values and conducted the tests with 10 packet trains and 700 bytes packet size. It resulted in 3.36% average relative error rate, therefore we increased both parameters several more times until the average error dropped below 2%.
The table 5.5 shows the results of those experiments.

Table 5.5: Relative error statistics for optimal intrusion

Packet Size	Train Length	Average	Standard Deviation	Min Error	Max Error
700	10	3.36%	5.33%	0.04%	48.69%
700	50	2.19%	1.43%	0.08%	10.93%
800	10	6.7%	6.4%	0.01%	32.91%
800	30	1.81%	1.03%	0.11%	6.72%
1000	30	1.32%	0.44%	0.06%	3.75%

Assuming that real networks are not usually empty, we can hypothesize that in practice higher intrusion will be necessary for relatively precise estimations. We will further research this topic in section 5.3

5.2.5 Capacity Range

One of the most important parameters is the capacity range of the links that we conduct our measurements on. So far we have seen that our tool can handle the links with capacities between 10 and 100 Mbits. The real networks can contain hops with higher capacities though, therefore it is necessary to research that aspect as well.

We tested the tool on several different networks with varying capacity ranges, such as [10, 100), as well as [100, 400), [400, 600) and [600, 1000) and as it appears, our methodology has a significant limitation when measuring the networks that contain links with relatively high capacities. Figures 5.9 and 5.10 show that the inaccuracy starts increasing with higher capacities.

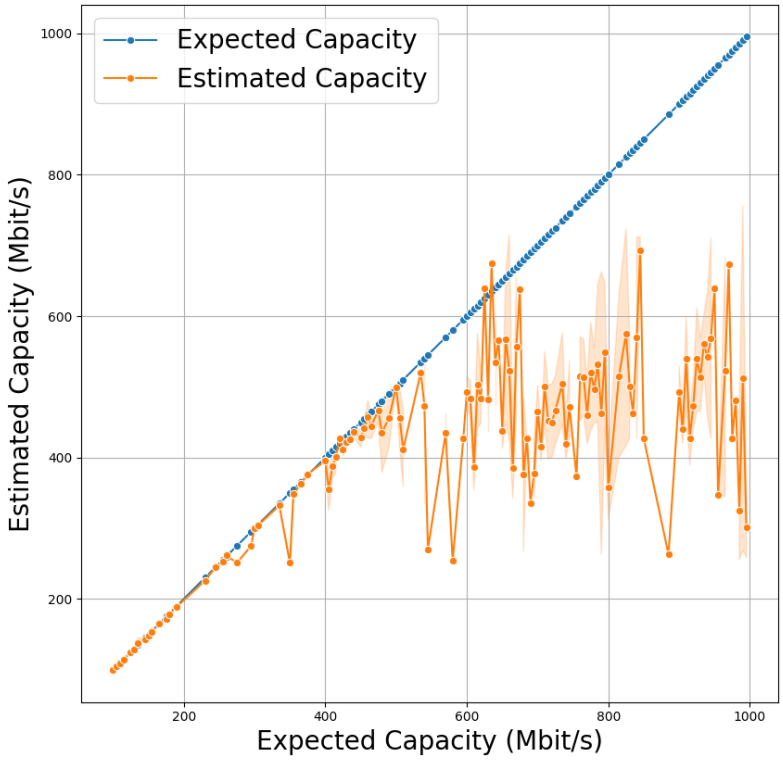

Figure 5.9: The impact of the capacity range on accuracy

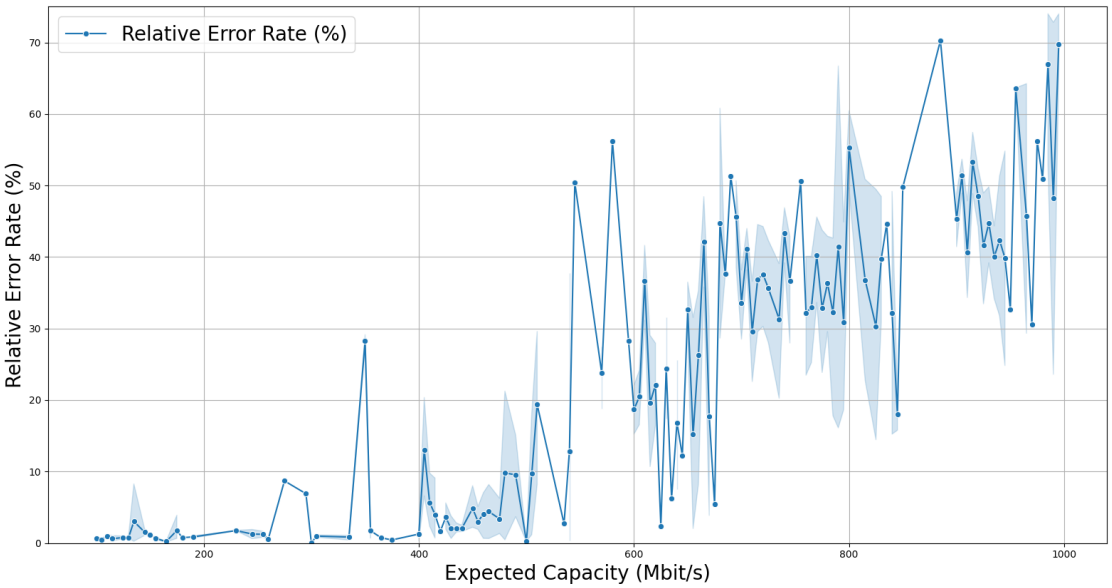

Figure 5.10: Error rates by capacity range

Conclusion

In conclusion, we can deduce that the accuracy of the tool greatly depends on the capacity range. We can argue that on an empty network the accuracy is good for the paths that have the capacity value up to ≈250 Mbits. However the pattern emerges that capacities above 250 Mbits are not underestimated below 250. We can assume that the estimation results below 250 are accurate. We will further observe this pattern with cross traffic in section 5.3

5.2.6 Packet Loss

It is a very commonly occurred event in everyday internet traffic when data packets fail to reach their destination. This event is referred as packet loss and there can be different reasons that cause it. According to A. S. Gillis[19] it can be caused by overloaded network, software and/or hardware problems, etc.

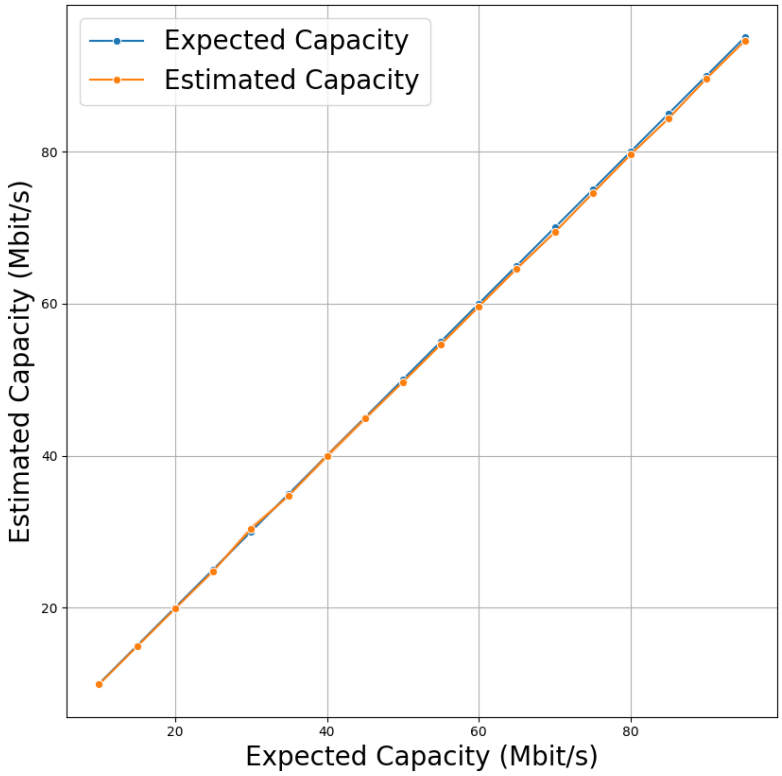

Figure 5.11: The impact of over 15% packet loss on capacity estimation

In order to make sure that our approach can cope with it and deliver the accurate estimations despite the lack of lost packets we tested our tool against this issue as well. We chose the data samples where the total packet loss in each test run was above 15%.

These samples had on average 35% total packet loss. Despite such a high loss the Figure 5.11 shows that the estimation accuracy was not compromised.

According to the test results so far, we can assume that the random packet loss does not affect the accuracy to extent that affects accuracy, because most of the time it happens to the larger packet trains and the amounts of the ICMP packets that reach the source host is high enough for PPrate to estimate capacities precisely enough.

Although there is a special case of packet loss that represents a serious challenge to our estimation methodology, that is ICMP rate limiting. In contrast with the end-to-end estimation, during which the measurements are based on inter-arrival-times of TCP ack signals, ICMP messages are prone to a huge amount of packet loss due to the ICMP rate limits of internet routers.
We will further discuss this issue in the next section.

5.2.7 ICMP Rate Limiting

One of the biggest obstacles active ICMP-based measurement tools can face is ICMP rate limiting as it causes a huge rate of packet loss and also the packets that reach destination might not be suitable for capacity measurements. According to the Linux man page[20] it works in the following way: there are two parameters that have to be configured for ICMP rate control: `icmp_ratemask` and `icmp_ratelimit`. The first parameter consists of 19 bits each determining which types of ICMP should be affected by the rate limit (each bit corresponds to a specific ICMP type) and the second one sets the limit of the ICMP responses. `icmp_ratelimit` indicates the minimum time interval between ICMP responses and it is measured in milliseconds. The value range is [1, 1000][21] and in Linux it has a default value of 1000.

All of our previous measurements were conducted with disabled `icmp_ratemask`. We have, however, also tested the effects of the rate limiting on our tool and tried to find out whether this can be a defining factor in the applicability of our methodology.

As the Figure 5.12 and the table 5.6 show the results are very unpredictable and unreliable.

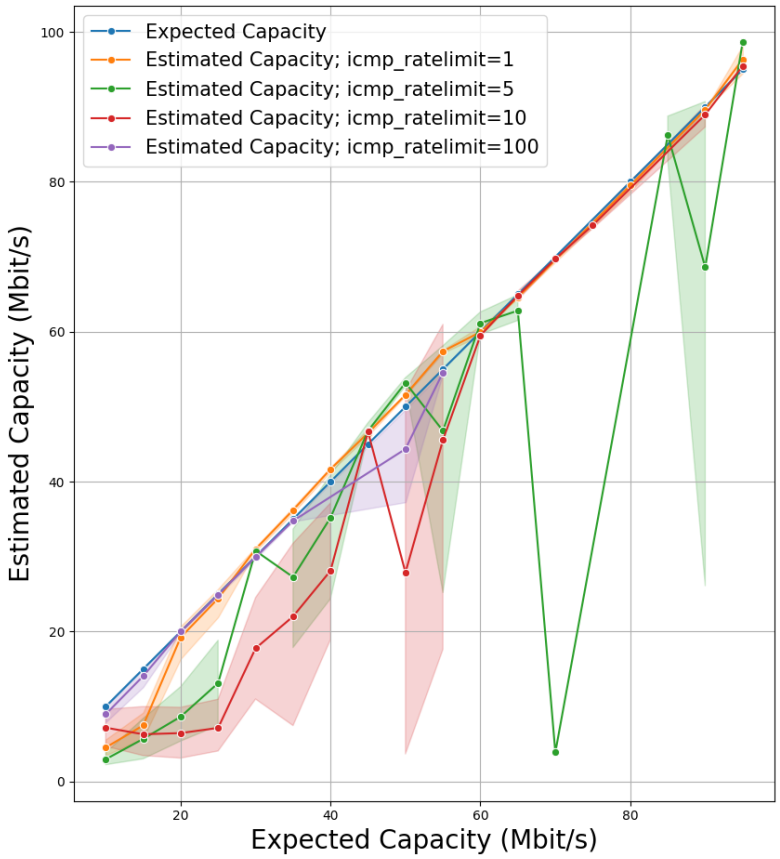

Figure 5.12: Effects of ICMP rate limiting on the accuracy

Table 5.6: Relative error statistics for the ICMP rate limiting

Train Length	Average	Standard Deviation	Min Error	Max Error
1	22.19%	31.59%	0.02%	84.94%
5	50.45%	41.24%	0.1%	94.45%
10	48.24%	44.35%	0.13%	96.09%
100	5.79%	19.79%	0.06%	96.28%

Nevertheless we have made one observation, that resulted in some improvements in terms of accuracy: the increase in train length provided more accurate results, as despite the huge packet loss rate, the number of 'surviving' packets was high enough to drastically improve the accuracy rate.

The Figure 5.13 presents the results of the estimation with the train length of 5000 packets and `icmp_ratelimit` of 1000 milliseconds.

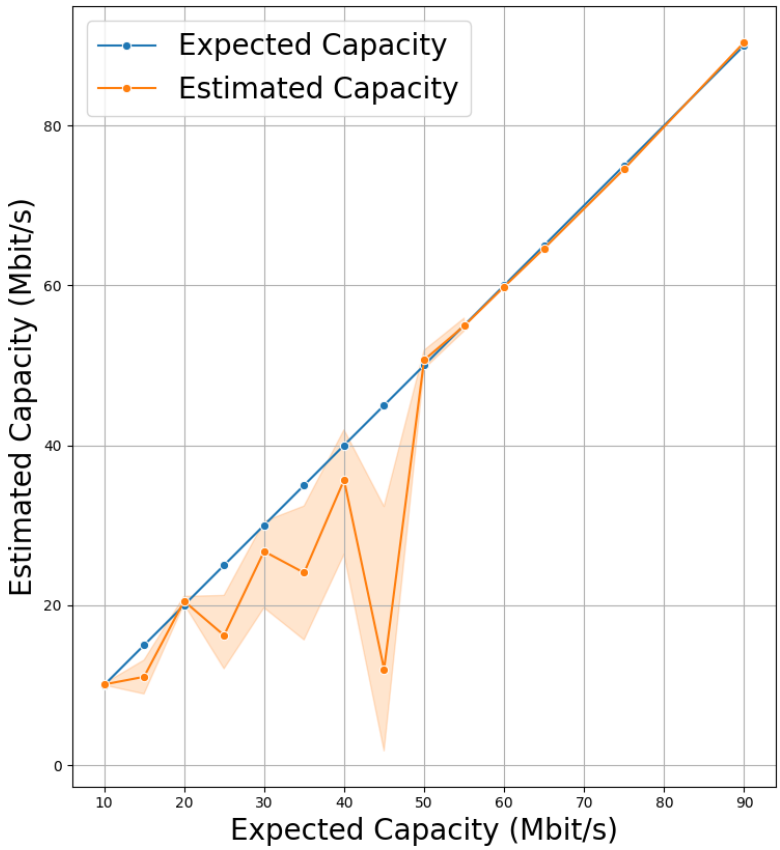

Figure 5.13: The impact of long packet trains against ICMP rate limit of 1000ms

Further research is required to find better solutions to this important problem, which it is out of scope of this thesis, however there are some efforts made in this direction, namely the papers by Ravaioli et al. [22] and H. Guo and J. Heidemann[23]

5.2.8 Summary

To summarize this section, we have examined our capacity estimation technology in empty networks and tested it against different parameters. The results so far have shown that the tool provides high accuracy on any amount of hops with small packet trains and packet size of about 800 bytes. The tool has also shown some downsides, namely the inaccuracy is significant on high capacity networks and it is not reliable when the routers on the path have limits on ICMP response rates.

In the next section we will try to test our framework in a close to real life scenario, i.e. apply artificial cross traffic to each router on the path and observe its effects on accuracy.

5.3 Evaluation during Cross-Traffic

We have conducted the previous experiments on empty networks without any flow interference. However this is not the case in the real Internet. The packets in real networks can be prone to higher rate of loss and/or delays which can distort the accuracy of our measurements.
Therefore it is vital to test our methodology in networks where routers are used by other hosts too.

We have implemented the emulation of the cross-traffic in our Mininet topologies in a way that is depicted in the figure 5.14. Each B_i host communicates with T_{i+1}. This means that the router interfaces are used not only by our test packets but also those sent by the T_i hosts. This could cause congestion at router interfaces and affect the accuracy of our estimations.

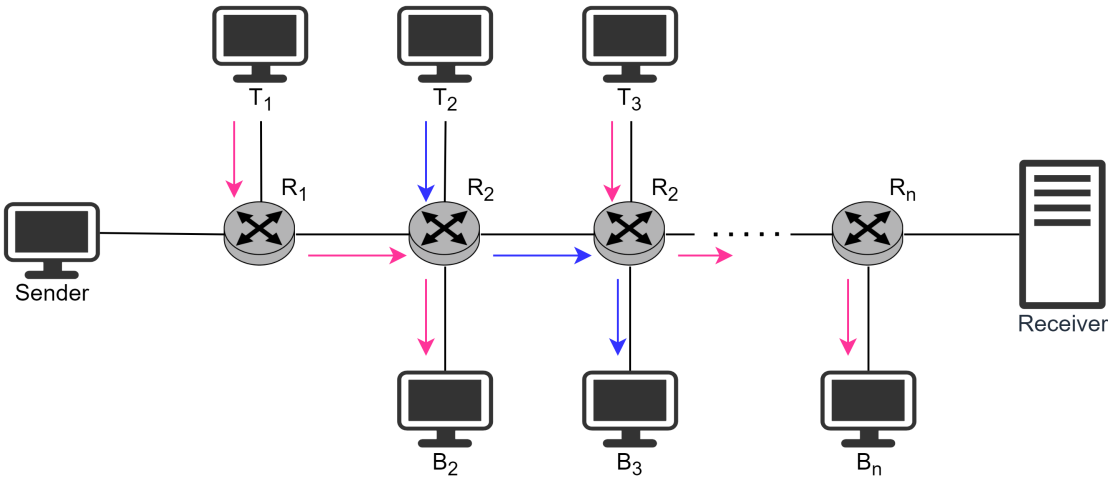

Figure 5.14: Flow of the cross traffic packets

For generating the cross-traffic we use `iPerf`[11]. The traffic load per each hop is generated based on the hop capacity.
Moreover, based on the results from empty network measurements we decided to reduce the packet size from 1500 to 1000 bytes by default.

We first conducted measurements with different cross-traffic loads, namely 50%, 80% and 100%. The results were varying in all three cases.

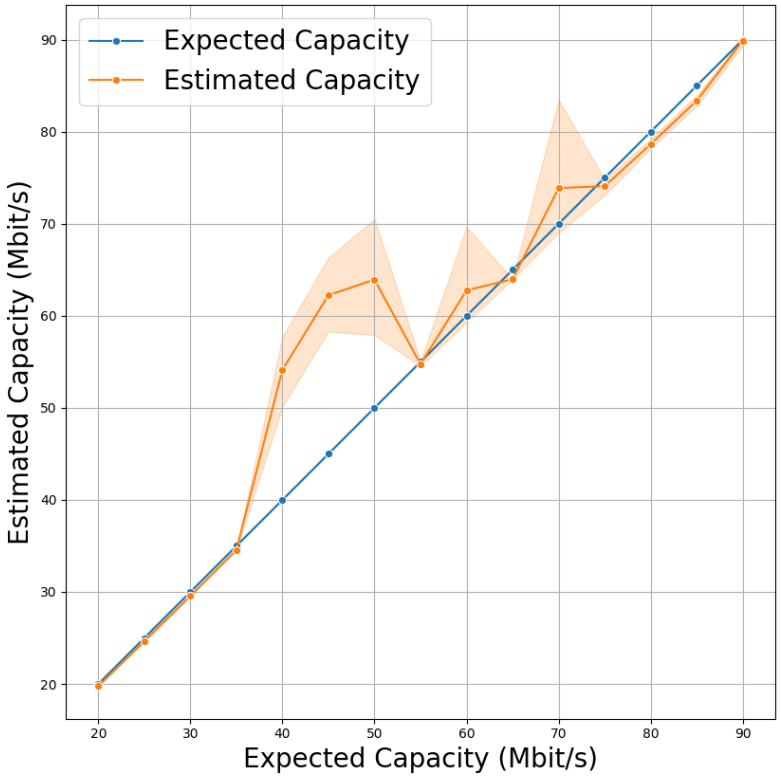

Figure 5.15: Cross traffic with 50% load

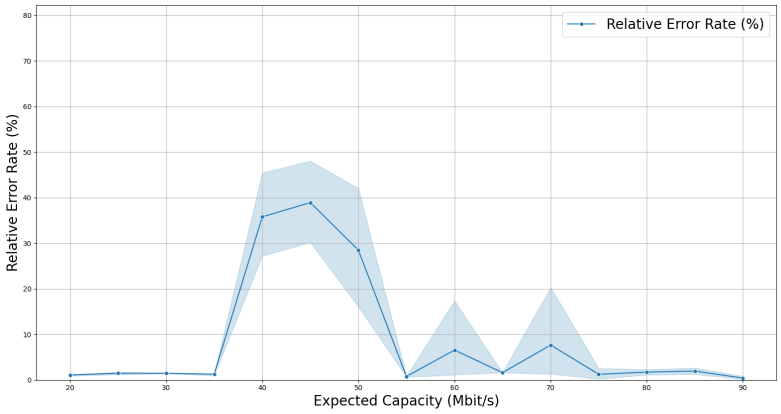

Figure 5.16: Cross traffic with 50% load relative error rate

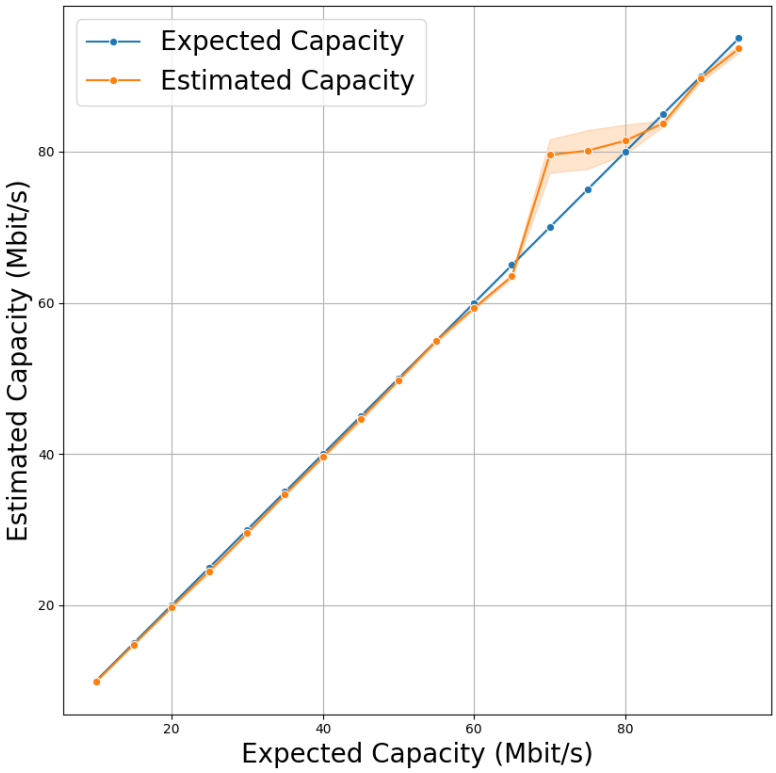

Figure 5.17: Cross traffic with 80% load

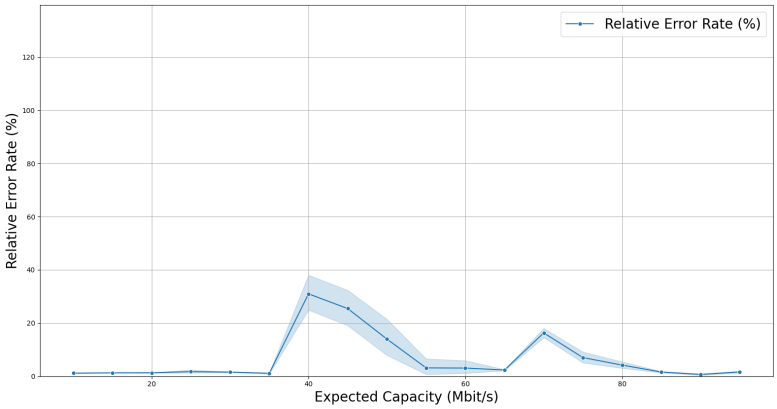

Figure 5.18: Cross traffic with 80% load relative error rate

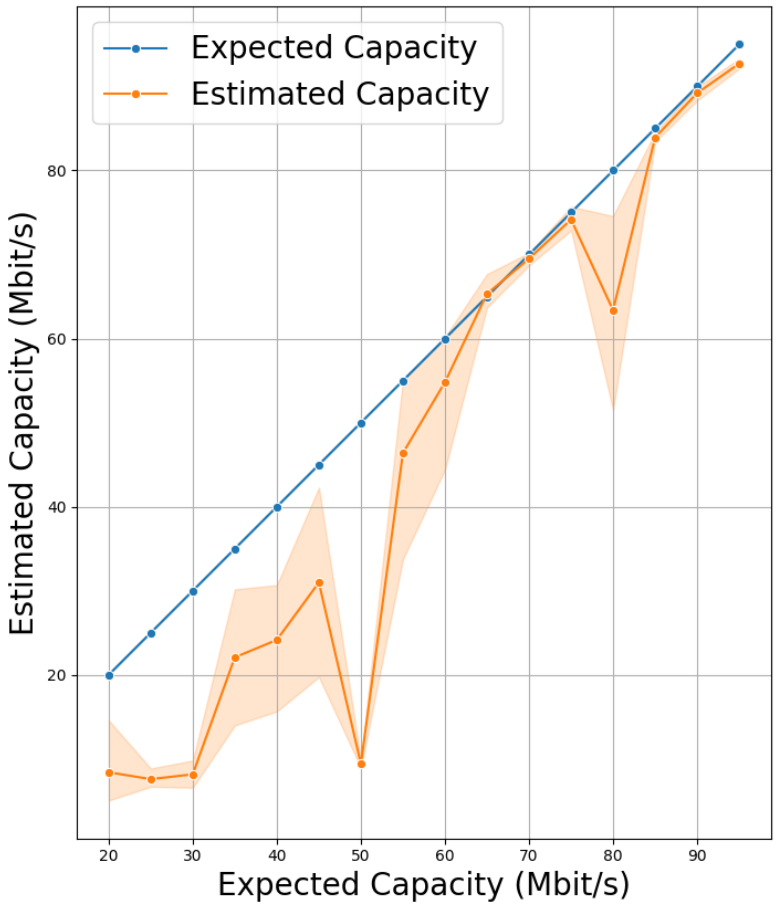

Figure 5.19: Cross traffic with 100% load

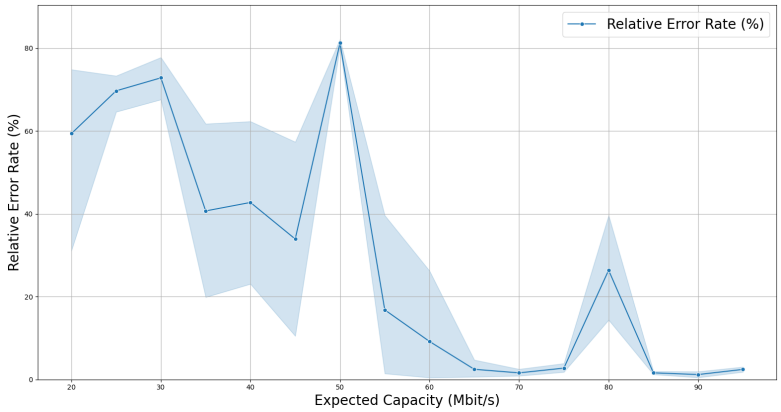

Figure 5.20: Cross traffic with 100% load relative error rate

As one can see from the figures 5.15, 5.16, 5.17, 5.18, 5.19 and 5.20 the results are mostly unreliable and despite similar circumstances apart from the cross-traffic load no

obvious pattern can be recognized among these results.

5.3.1 Packet Size

We have tested different packet sizes in networks with 100% cross traffic load and as it appears increasing packet size does not much improve the accuracy.

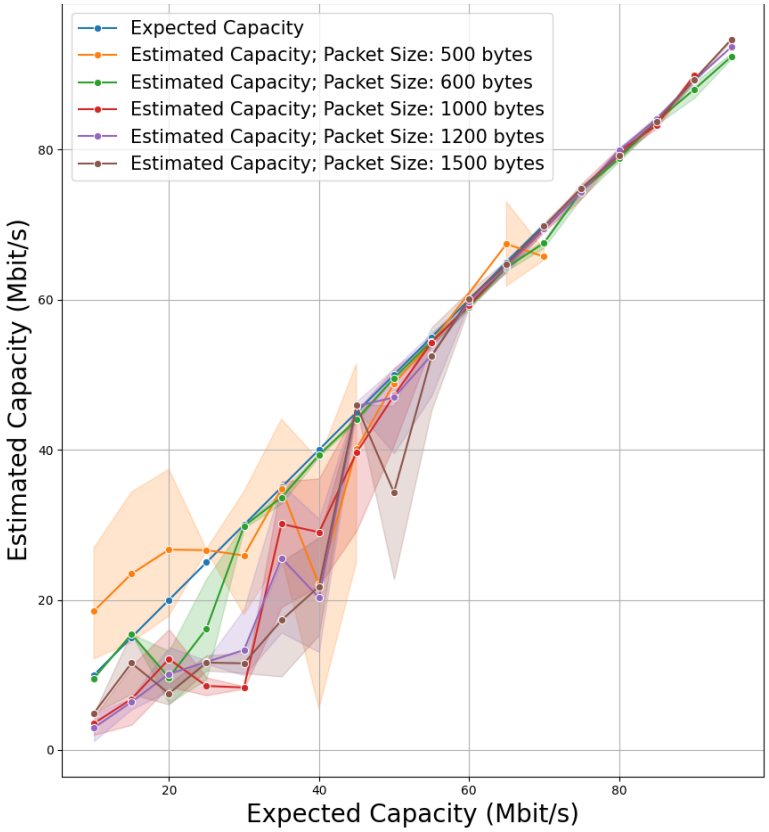

Figure 5.21: Impact of packet size during 100% cross traffic load

5.3.2 Train Length

Testing train length gave relatively good results, however the maximum value of relative error was 621%.

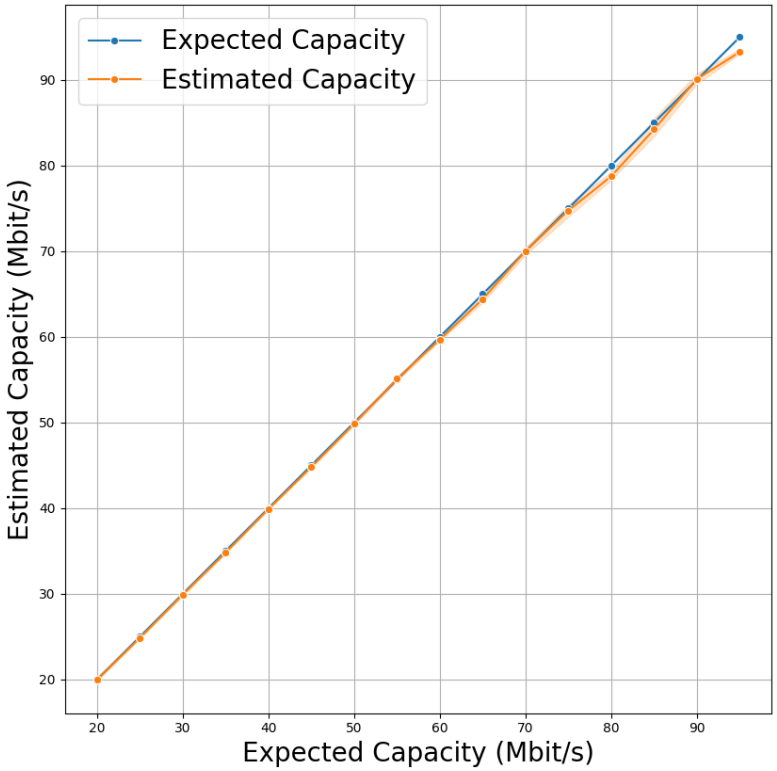

Figure 5.22: Impact of packet train length during 100% cross traffic load

5.3.3 Path Length

To check the effects of the path length on the estimation resuts during cross-traffic, we conducted measurements on a 63-router path once again. This time with 80% cross traffic load.

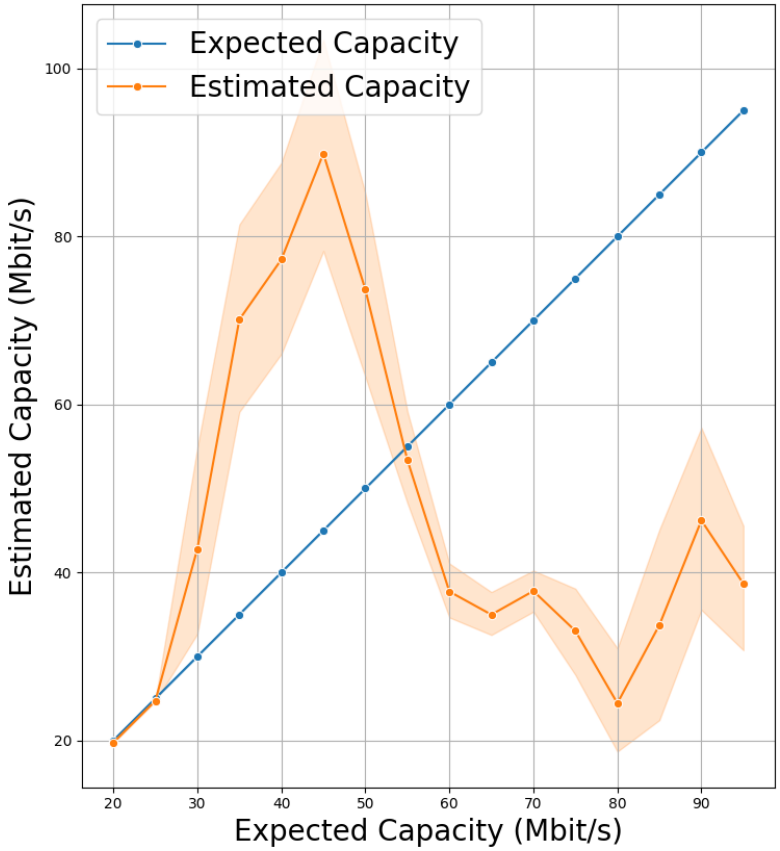

Figure 5.23: Measurement results on 63-router path with 80% load

5.3.4 Packet Loss

It is worth mentioning that the packet loss during cross traffic experiments was over 50% most of the time regardless the configuration.

5.3.5 Summary

To summarize this section, the results from the cross-traffic tests were not as good as we expected. Despite a significant amount of correct estimations, the overall results look quite random and fail to achieve the desirable accuracy levels. However there is still

Chapter 6

Conclusion

In this thesis we have implemented a hop-by-hop capacity estimation methodology and evaluated it based on various parameters.

The following sections will shortly summarize our work throughout the timespan of the thesis and address all the concerns and questions that we had in the beginning.

6.1 Evaluation Results

The evaluation has shown that our proposed methodology is significantly flawed when it comes to the inspection of overloaded networks. On the other hand, it delivers high accuracy when it does not face the flow interference. Therefore, like passive capacity estimation tools, ours also depends on the network that has to be measured.

Moreover, our tool is unable to work as supposed when it communicates with ICMP rate limited routers and requires improvements in terms of this problem.

6.2 Answers to the Research Questions

After the careful and thorough research we can already answer the questions we were aiming to address in the beginning of this thesis:

- **How to measure network capacity hop-by-hop?**
  We have developed a way to estimate capacity hop-by-hop by sending ttl-exceeding packets to all the routers on the path to the destination. The

- **How to optimize the trade-off between accuracy and intrusiveness regarding large-scale measurements?**
  When it comes to the empty networks, we have managed to find an optimal rate of intrusion of around 700 packets with the train length less than 100. (We were able to get accurate results with as few as 10 packets per train). However we do not guarantee the consistency of these numbers.

- **How robust is the proposed solution regarding the handling of flow-interference, such as cross-traffic?**
  The robustness in terms of cross traffic has appeared to be a big problem in our measurements. So far we cannot recommend our solution for the networks that handle large traffic on regular basis.

- **Are we able to locate the capacity bottlenecks of a network?**
  The answer to this question greatly depends on the accuracy of the hop-by-hop estimation. As we have seen in the Evaluation chapter, there are situations when we get accurate results and there are cases when the results are somewhat inaccurate or not reliable at all. In case of the reliable results, the answer will be 'yes'. We can locate the capacity bottlenecks based on the first smallest capacity value from the sequence of results.

  There is a catch, however: if the path consists of multiple hops and there are several bottlenecks, we are able to locate only one, namely the first one. For example, if there are 15 hops on the path and the bottleneck is located on the first hop, but the capacities of several other links equals to that of the first link, we will not be able to see it. In such situations, we can only assume that the subsequent capacities are more than or equal of the capacity of the first hop.

6.3 Future Work

As our evaluation has shown, there still is a lot of room for improvement for our methodology.
First and foremost, we have not conducted any test runs on a real network, as the performance of Mininet greatly depends on the machine it runs on and this could compromise the accuracy of the tool. Therefore, internet measurements should be performed in order to see how our tool handles the real traffic in the Internet.
Secondly, further research is necessary for handling the random inaccuracy issues with cross-traffic as it is the key factor when it comes to the real-life usage of our framework.
And finally, ICMP rate limiting represents the biggest challenge to our tool. This prob-

lem has been tackled by several researchers, most importantly, Guo and Heidemann who developed the ICMP rate limit detection software - FADER[23].

Chapter A

Reproducibility

This chapter describes the source code repository and provides the information and instructions of how to use, alter and/or improve the current version of the Hop-by-hop measurement framework.

A.1 The Source Code Repository

The source code repository for this thesis is located on the GitLab instance belonging to the TUM. The `ma-andguladze/App` directory contains the whole source code.

The tool consists of the following files:

- `PPrate.py` - The pprate algorithm implementation by Patryk Brzoza[15]
- `TrafficGenerator.c` - As the name suggests, this program generates TCP traffic from one host to another. It takes the IP addresses of two hosts as arguments and creates the traffic via raw sockets. The file should be compiled before the first run.
- `mininet_topo.py` - Builds an instance of the network in Mininet considering all the necessary parameters passed, conducts the traffic generation, captures the traffic and saves the results in pcap files.
- `prepare_test.py` - Contains several helper functions, most notably the config file parser and the capacity generator.
- `process_tcp_csv.py` - Converts the Calculates the end-to-end capacity of the network. Implementation by Brzoza[15]

- `process_icmp_csv.py` - Converts the `icmp.pcap` file into csv and delivers the hop-by-hop capacity estimations based on the latter
- `run_test.py` - The main file that bootstraps the tool and runs the experiments in one setting. It requires a `JSON` configuration file to be passed as an argument with the help of which it is possible to manipulate several key parameters necessary for testing.

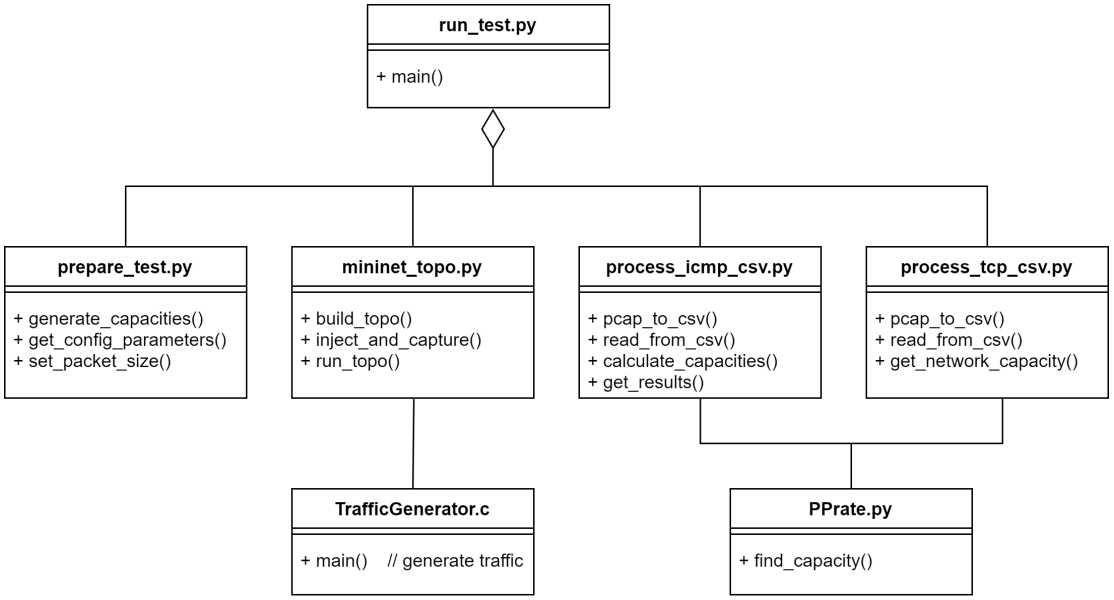

Figure A.1: The architecture of the framework

A.2 Setup

In order to run the test measurements with our tool, several dependencies and software should be installed on the Linux machine.

The experiments conducted by us were run only on Mininet - a network emulator tool which creates a realistic virtual network of virtual hosts, switches, controllers and links and runs a real kernel and application code[17]. To run the experiments on Mininet, it is necessary to install the emulator first. It is highly recommended to install Mininet from the source GitHub repository instead of packages, as the Mininet homepage[24] states that packages could give an older version.

Mininet installation:

```
1  $ git clone git://github.com/mininet/mininet
2  $ mininet/util/install.sh [options]
```

The default branch normally will install the latest stable version, however in case a user wishes to have any other version, it also possible to check out the branch with the following script before running `install.sh`:

```
$ cd mininet
$ git tag  # list available versions
$ git checkout -b mininet-2.3.0 2.3.0  # or any desired version from the list
$ cd ..
```

Note that Mininet must run as a root.

As for the rest, the following dependencies are required to be installed to run experiments:

- pandas
- NumPy
- SciPy
- tshark
- iPerf

For setting up these and compiling the `TrafficGenerator.c` program, run `install.sh` script in the `ma-andguladze/App` directory.

A.3 Running Tests

The measurements must be run with a `JSON` config file containing all the parameters that are interesting and important for our measurements. Manipulating these parameters gives us a better picture about strengths and weaknesses of our approach.

The following script executes the experiment:

```
$ sudo python run_test.py config.json
```

The following `JSON` serves as an example of a configuration file used for experiments. It contains default parameter values, each of which will be manipulated respectively in upcoming sections:

```
{
  "topo_size": 3, // path length
  "capacity_range": [10, 100],
  "capacity_delta": 5
  "packet_size": 1400,
  "packets_per_hop": 300,
  "icmp_ratelimit": 0,
  "packet_loss": 0,
  "cross_traffic": 0.0,
```

```
10   "output": "results/results.csv"
11 }
```

Disclaimer: *`packet_loss` is a probability, therefore it does not always cause the packet loss during test runs; On the other hand, regular test runs with `packet_loss = 0` can result in lost packets.*

Chapter B

List of acronyms

ICMP	Internet Control Message Protocol.
IP	Internet Protocol.
ISO	International Organization for Standardization.
MTU	Maximum Transmision Unit.
OSI	Open Systems Interconnection. (Reference model for layered network architectures by the OSI.)
PPTD	Packet Pair/Train Dispersion.
RTT	Round Trip Time.
TCP	Transmission Control Protocol. (Stream-oriented, reliable, transport layer protocol.)
TTL	Time-To-Live.
UDP	User Datagram Protocol. (Datagram-oriented, unreliable transport layer protocol.)
VPS	Variable Packet Size.

Bibliography

[1] ISO, *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model*, https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html/, [Online; accessed 11.2021].

[2] B. Sandberg, "The OSI Reference Model", in *Networking - The Complete Reference, 3rd edition*, 2015, pp. 40–65.

[3] Internet Engineering Task Force (IETF), *Internet Protocol*, https://datatracker.ietf.org/doc/html/rfc791, [Online; accessed 11.2021].

[4] *TCP/IP Header*, https://www4.cs.fau.de/Projects/JX/Projects/TCP/jxtcp.html, [Online; accessed 11.2021].

[5] ——, *Internet Control Message Protocol*, https://datatracker.ietf.org/doc/html/rfc792, [Online; accessed 11.2021].

[6] *Linux man page - icmp*, https://man7.org/linux/man-pages/man7/icmp.7.html, [Online; accessed 11.2021].

[7] ——, *Transmission Control Protocol*, https://datatracker.ietf.org/doc/html/rfc793, [Online; accessed 11.2021].

[8] ——, *User Datagram Protocol*, https://datatracker.ietf.org/doc/html/rfc768, [Online; accessed 11.2021].

[9] R. Prasad, C. Murray, C. Dovrolis, and K. Claffy, "CSMA/CA Bandwidth Estimation: Metrics, Measurement Techniques, and Tools", Dec. 2003.

[10] Silver Moon, *How to Code Raw Sockets in C on Linux*, https://www.binarytides.com/raw-sockets-c-code-linux/, [Online; accessed 11.2021].

[11] Dugan et al, *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*, https://iperf.fr/, [Online; accessed 11.2021].

[12] F. Abut, "CSMA/CA Through the Diversity of Bandwidth-Related Metrics, Estimation Techniques and Tools: An Overview", Aug. 2018.

[13] C. Dovrolis, P. Ramanathan, and D. Moore, "Pathrate: Packet-Dispersion Techniques and a Capacity-Estimation Methodology", Dec. 2004.

[14] T. En-Najjary and G. Urvoy-Keller, “PPrate: A Passive Capacity Estimation Tool”, May 2006.

[15] P. Brzoza, “Implementation and Evaluation of Passive Capacity Estimation”, B.Sc. Thesis, Technische Universität München, Munich, Germany, 2019.

[16] *TCPDUMP man page*, `https://www.tcpdump.org/manpages/tcpdump.1-4.99.1.html`, [Online; accessed 11.2021].

[17] *Mininet Home*, `http://mininet.org/`, [Online; accessed 2021].

[18] S. Iveson, *IP Time to Live (TTL) and Hop Limit Basics*, `https://packetpushers.net/ip-time-to-live-and-hop-limit-basics/`, May 2019.

[19] Alexander S. Gillis, *Packet Loss*, `https://www.techtarget.com/searchnetworking/definition/packet-loss/`, [Online; accessed 11.2021].

[20] *icmp - Linux IPv4 ICMP kernel module*, `https://man7.org/linux/man-pages/man7/icmp.7.html`, [Online; accessed 11.2021].

[21] Juniper Networks, *icmp - Error Message Rate Limit*, `https://www.juniper.net/documentation/us/en/software/junos/transport-ip/topics/ref/statement/icmp-edit-chassis.html/`, [Online; accessed 11.2021].

[22] R. Ravaioli, G. Urvoy-Keller, and B. Chadi, “Characterizing ICMP Rate Limitation on Routers”, 2015.

[23] H. Guo and J. Heidemann, “Detecting ICMP Rate Limiting in the Internet(Extended)”, Apr. 2017.

[24] *Mininet Installation*, `http://mininet.org/download/`, [Online; accessed 11.2021].