NTNU – Trondheim
Norwegian University of
Science and Technology

Real-Time Applications in Smart Grids Using Internet Communications:

Latency Issues and Mitigations

Ole-Henrik Borlaug

Master of Science in Cybernetics and Robotics
Submission date: June 2014
Supervisor: Sverre Hendseth, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Project Description

The Smart Grid is an evolving vision comprising bidirectional flow of both energy and information in the power grid. There are several real-time applications in Smart Grids including control loops, sensory systems, and alarm systems.

Traditionally deterministic communication approaches are used to deploy real-time applications. The internet is not inherently deterministic, making it difficult to deploy real-time systems. However, for a system as geographically widespread as the Smart Grid, the economic gains of deploying real-time applications over the Internet are great.

Therefore, the task is to examine latency in today's Internet through:

1. a literature study on sources of latency in Internet communications and existing mitigations.
2. examining the presence of the problems in end-users last mile links in Norwegian Internet.
3. discussing the mitigations.
4. giving recommendations to minimize latency in end-user links.

Preface

This is my master thesis, the final work of my master's degree in Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). The thesis has been written for the department of Engineering Cybernetics at NTNU, and was developed together with Associate Professor Sverre Hendseth.

My motivation for choosing this work has been my wish to help the development of a Internet of things and a great interest into Smart Grids.

I would like to thank my supervisor Sverre Hendseth, for providing me with motivation and direction during the work. I would also like to thank my co-advisor Roberto Rigolin Ferreira Lopes for discussions.

Finally I would like to thank my family for their support and encouragement, especially my brother, Jan-Trygve for opening my eyes to the field of cybernetics.

Trondheim, June 2. 2014

Ole-Henrik borlaug

Abstract

The Smart Grid will be a geographically widespread system, with real-time applications spanning large distances. To deploy real-time applications traditionally Industrial networks designed to specification are used. Since the Smart Grid will be geographically widespread and consist of many devices there is a great economic gain by utilizing the Internet as its communication network.

This thesis presents literature describing problems with congestion, bufferbloat and active queue management in today's Internet. Additionally existing solutions and mitigations to these problems are presented and discussed.

In addition, two end-user last mile links connected to the Norwegian Internet are characterized in terms of delay and passive traffic measurements. One link utilizes fiber to the home and one utilizes Asymmetric Digital Subscriber Line (ADSL) technology. The characterization results show evidence of both bufferbloat and lack of active queue management in the ADSL link. The characterization results provide no conclusive evidence of neither bufferbloat nor active queue management in the fiber to the home link.

The guaranteed timely delivery problem over an Internet path has been analysed. This leads to recommendations on minimizing latency at the end-user based on the discussion on existing solutions and the characterization experiments performed.

Sammendrag

Smart Grid vil være geografisk spredte systemer og vil inneha sanntidsapplikasjoner med sanntidskommunikasjon over lange avstander. Vanligvis brukes industrielle nettverkssystemer ved ønske om sanntidskommunikasjon. Siden Smart Grid vil være geografisk spredte systemer er de økonomiske fordelene ved å bruke Internett som kommunikasjonsmedium store.

Denne oppgaven presenterer litteratur som beskriver problemer med opphoping, for store buffere og aktiv kø-administrasjon i dagens Internett. Eksisterende løsninger og bøtemidler på disse problemene blir også presentert og diskutert.

I tillegg, er to sluttbruker tilkoblinger til det Norske Internett blitt karakterisert med tanke på forsinkelse og passive trafikkmålinger. Den ene av tilkoblingene bruker fiberteknologi og den andre bruker ADSL teknologi. Resultatene fra karakteriseringen viser bevis for at ADSL tilkoblingen har problemer med for store buffere og mangler aktiv kø-administrasjon. Ingen konkluderende bevis er funnet som taler for hverken for store buffere på fibertilkoblingen eller bruk av aktiv kø-administrasjon.

Problemet med å levere innen et tidskrav er analysert for en sti over Internett. Dette gir utspring for gitte anbefalinger for å minimere forsinkelsestiden over Internettilkoblingen til sluttbruker basert på diskusjonen om eksisterende løsninger og resultatene fra karakteriseringen.

Contents

List of Figures

List of Tables

List of Acronyms

ADSL	Asymmetric Digital Subscriber Line.
AMS	Automatic Metering Stations.
AQM	Active Queue Management.
BDP	Bandwidth-Delay Product.
BIC	Binary Increase Congestion.
CDN	Content Delivery Network.
CoDel	Controlled Delay.
CRNA	Centre for Resilient Networks and Applications.
DSL	Digital Subscriber Line.
DSLAM	Digital Subscriber Line Access Multiplexer.
FTP	File Transfer Protocol.
HTTP	Hypertext Transfer Protocol.
ICMP	Internet Control Message Protocol.
IP	Internet Protocol.
ISP	Internet Service Provider.
LAN	Local Area Network.
NAT	Network Adress Translation.
NIX	Norwegian Internet eXchange.
PI	Proportional Integral.
PIE	Proportional Integral controller Enhanced.
QoS	Quality of Service.
RED	Random Early Detection.
RTT	Round Trip Time.
SCADA	Supervisory Control and Data Acquisition Systems.
SD	Standard Deviation.
SG	Smart Grid.

TCP	Transmission Control Protocol.
TCP/IP	Internet Protocol Suite.
UDP	User Datagram Protocol.
xDSL	x Digital Subscriber Line.

1 Introduction

The structure of the current power grid has been the same for several years. There have been smaller additions to the way the power grid is controlled and how the energy is produced, but centralized power production plants is still the industry standard. The Smart Grid (SG) comprises bidirectional flow of both energy and information in the power grid, and distributed energy production.

The e-GOTHAM project [3] is a project defining complete solutions for microgrids in the residential, tertiary and industrial sectors including different loads, distributed generators and energy storage components. Both the Norwegian University of Science and Technology (NTNU) and SINTEF are collaborators on the project. NTNU and SINTEF mainly focus on designing a middle-ware that can handle the communication securely and reliably, and compile recommendations about the communication infrastructure.

The Internet is a ubiquitous network and is already in place. The economic prospects of using the Internet as communication network are therefore great. The SG is dependent upon real-time measurements and information. Traditionally industrial networks have been used to deploy real-time applications. However, the Internet is designed to be a best effort network, it does not provide guarantees.

This thesis tries to encourage the use of Internet communications for real-time applications in SGs by investigating latency problems in today's Internet and future solutions for tomorrow's Internet.

This thesis is divided into seven chapters. Chapter 2 presents the SG concept and the communication requirements. Chapter 3 will present literature describing the problems with congestion and bufferbloat that exist in today's Internet, existing solutions and mitigations to these problems, information on the Norwegian Internet, and finally measurement techniques used to investigate problems with Internet links.

Chapter 4 will present the characterization experiment performed in this thesis, and Chapter 5 will present the results from this. Chapter 6 will discuss the results found, discuss existing solutions and mitigations to congestion and bufferbloat, and give recommendations to reduce latencies at end-users based on these discussions.

A short and concise conclusion and some suggestions for future work is found in Chapter 7.

2 The Smart Grid Concept

2.1 The Smart Grid: The Next Generation Power Distribution System

This section, 2.1, was mainly written in the semester project Borlaug et al. [10] and is only repeated here for context and as a short introduction to Smart Grids (SGs).

The SG is the next generation of power distribution systems and entails two-way flow of both information and electricity. It uses two-way communication throughout the power grid to embed smartness and improve reliability and robustness with supervision and control. Two-way flow of energy entails that those considered consumers of electricity today, end-users, can produce energy. SGs are considered a solution to the want of widely distributed power generation, better power quality and more optimization of both production and consumption of energy. The SG can briefly be summarized and compared with the existing grid in Table 1.

Over the last years, the discussion on global warming has provided new interest for the adoption of renewable energy sources. Renewable energy sources are paramount for a sustainable future. Additionally, such energy sources are often environmentally friendly. However, renewable energy sources are inherently random and intermittent in nature [28], and when such sources are employed in a widely distributed manner, problems concerning power fluctuations and general power supply quality arises. To control power grid characteristics such as the power fluctuations effectively, real-time information about the state and output of the distributed power generators is necessary.

Real-time entails that there exists timing requirements on the information. Information is no longer valid if delivered too late or out of order. Burns and Wellings [11] states that, "the correctness of a real-time system depends not only on the logical result of the computation, but also on the time at which the results are produced".

The vision of the SG should handle distributed power generation from renewable energy sources, and the communication infrastructure of the SG should therefore handle real-time requirements.

Table 1: A brief comparison between existing grid and the smart grid by Fang et al. [15]

Existing Grid	**Smart Grid**
Electromechanical	Digital
One-way communication	Two-way communication
Centralized generation	Distributed generation
Few sensors	Sensors throughout
Manual monitoring	Self-monitoring
Manual restoration	Self-healing
Failures and blackouts	Adaptive and islanding
Limited control	Pervasive control
Few customers choices	Many customer choices

2.2 Communication Requirements of Smart Grids

Fang et al. [15] states that: "The most important question in the communication subsystem is, 'What networking and communication technology should be used?". To answer this question they list a set of basic requirements; the communication platform must:

1. support the Quality of Service (QoS) of data. Critical data should be delivered within delivery time requirement.
2. be highly reliable in terms of up-time and delivering QoS.
3. cover large geographical areas, and be ubiquitous.
4. guarantee security and privacy.

As mentioned earlier the communication network needs to handle real-time requirements because of renewable energy sources. Niyato et al. [32] point out the economic aspects of a reliable communication network. They showed that optimizing the power consumption with information on demand and supply from distributed energy resources gives considerable savings in comparison to not having this information, e.g. breakdown of the communication system. According to their findings, the savings could be over 900 USD on a bill of 2000 USD, i.e. 45%.

The efforts of solving the communication problem can be divided into two major paths, using either a public or private communication infrastructure [21]. One such already existing public network is the Internet. The economic prospects of using the Internet as communication medium in SGs are great. Although using public networks is less expensive than using private networks, problems with ensuring security, privacy and QoS exist [14, 21].

However, several studies have considered using the Internet in control loops [12, 28]. The `openADR` project aspires to support Internet communications [1]. Liyanage et al. [28] performed an experiment running a control loop in a segment of a university network connected to the Internet. They show from experimental results that their proposed control strategy is adequate. They achieve satisfactorily control of the power fluctuations introduced by simulated renewable energy sources.

Internet Protocol Suite (TCP/IP) is a widespread and well known communication technology, and the focus on using TCP/IP as the common topology or layer to manage end-to-end communication is increasing not only in SGs, but in industrial instrumentation too [14, 15, 21, 28]. TCP/IP can be deployed on top of several different physical layers and tunneling can be employed to update legacy protocols.

Expanding the existing power grid into a fully operational SG the protocols already in use, e.g. Supervisory Control and Data Acquisition Systems (SCADA), need to be updated to protocols that can support the SG requirements. RS485, RS232 and such protocols can easily be implemented in semitransparent tunnels over TCP/IP. However, Sauter and Lobashov [35] state that they do not consider encapsulating protocols like SCADA, protocols using a simple request/respond scheme with its own addressing, into TCP/IP protocols because this generates an overhead without additional benefit.

The private communication infrastructure approach is expensive but has the benefits that the network would not have to accommodate other communication than the SGs commmunication. This separate network could be built to specification, as a deterministic and QoS enabled network.

2.2.1 Latency Reliability Requirements of Smart Grid Apllications

SGs comprise several different applications. Each application has different requirements for the communication. These can be classified by their QoS requirements. Table 2 shows some communication requirements of different SG applications compiled by Hossain et al. [25].

Table 2: Latency and reliability requirements of Smart Grid services, from Hossain et al. [25]

Service	**Troughput**	**Latency**	**Reliability**
Price	Low	1-10 min	98%
Metering	Low-High	0-15 min	98-99%
Consumer information messaging	Medium-High	5-30 s	99%
Electric car (PHEV)	Medium-High	5-60 s	98-99.5%
Dispatch distributed user storage	Medium-High	0-5 s	98-99.5%

Table 2 shows that different applications have different requirements. The most demanding task is the dispatch of user storage since the required maximum latency is lowest and more of the packages need to arrive in time.

2.3 Lyse Energi's Virtual Sensors

At Teknisk Ukeblad Smart Grid Summit 2014, Gundegjerde [20] presented Lyse Energi's vision for the SG. They envision to utilize the readings gathered from end-users Automatic Metering Stations (AMS) to virtualise sensors throughout the grid.

They are operating in Rogaland. This regions grid is not built for the current population. In the 1970's Rogaland saw a great increase in population due to the oil industry's boom. The grid has therefore experienced blackouts, and a lot of the substations do not have adequate measurement sensors, some do not even have any sensors. Lyse Energi therefore struggle with determining the state of their grid.

However, they see the deployment of AMS at end-users as their opportunity to utilize real-time measurements from these to virtualize sensors in substations thereby saving huge expenses in sensory systems in these substations.

Lyse Energi is also a provider of fiber to the home connectivity and they utilize this connectivity in their deployment of AMS.

3 Internet - A Myriad of Uncertainties and Random Issues

The Internet is a vast and globally ubiquitous communication network built on the ideas of the ARPANET. It is a global collaboration between nations and companies, and connects millions of different devices. The Internet is additionally a rapidly changing network

The Internet is a vastly heterogeneous network with many different devices communicating in a common language, TCP/IP protocols, and a great diversity of topology and technology. The internet has always had problems with packet-loss and intermittent high latencies [16, 19, 26, 30]. All of the issues throughout history have been extensively researched as more and more researchers have become aware of them. Many of the problems have been solved, some are not solved and are still a persistent part of the Internet.

As Jim Gettys of Bell Labs so adequately put it, "We are flying on an Internet airplane in which we are constantly swapping the wings, the engines, and the fuselage, with most of the cockpit instruments removed but only a few new instruments reinstalled. It crashed before; will it crash again?" [19]. Where he is referring to the congestion collapses of the 1980's.

3.1 Latency Issues

Real-time applications rely on timely delivery. The usual way of ensuring timely delivery is to use a deterministic communication path. There are several industrial networks that are deterministic by design, which entails that all aspects, including delay and bandwidth, can be determined with a maximum and/or a minimum quantity. In terms of QoS and the reliability metric, deterministic communication networks usually have 100 % reliability within the designed limits.

The Internet on the other hand is not inherently deterministic and therefore ensuring that critical information is passed to the recipient in time, is difficult.

Reliability is defined by Hossain et al. [25] as the probability that the message is successfully received within the latency limit. The formal definition of a reliability requirement is given by:

$$R \leq P_s \int_0^{L_r} f_{D|s}(t)dt \tag{1}$$

Where R is the reliability requirement, L_r is the latency requirement or limit, P_f is the probability that the message is lost somewhere in transit, $P_s = 1 - P_f$ is the probability that the message reaches the destination and $f_{D|s}$ is the density of the delay of the message, conditioned on successful delivery.

From (1) it can be extrapolated that packet-loss and latency are two very important metrics comprising the reliability of a connection. Packet-loss arises when a node in the route from sender to receiver drops the packet for various reasons including:

1. Full buffers.
2. Congestion conscious drop.
3. Corrupt packets.
4. The packet time to live has run out.

Congestion conscious drop entails here that a router or node in the Internet drops a packet to avoid congestion or because of congestion. Drops due to full buffers are not defined as congestion conscious because there is no actual weighing of options. Both packet-loss due to full buffers and congestion conscious drops are related to congestion, i.e. the load or traffic in the network.

End-to-end latency is defined by Hossain et al. [25] as the total processing time and network transit time, measured from the time of message departure from the source actor to the time of message reception at the destination actor. The sources of delay in the Internet include:

1. Transmission time, i.e. transmission of data over a link.
2. Processing of packet at each node, e.g. routing decisions.
3. Queuing in buffers. e.g traffic from other users, cross-traffic, filling buffers.
4. Route selection through the Internet.

As stated in the previous section the Internet is heterogeneous in many aspects. A normal communication between two end-points in the Internet usually involves several hops through this heterogeneous network over several different links. The limiting factor on the quality of a communication is the weakest link. I.e. the max throughput of a Transmission Control Protocol (TCP) flow is the max bandwidth that the slowest link in the communication path can deliver, the bottleneck rate.

Latency due to transmission time, and node processing can be viewed as constant for a given path through the Internet and a given physical infrastructure. Latency due to retransmission is related to packet-loss, and queuing in buffers is related to level of congestion or the filling grade of buffers in the Internet. When a high level of congestion is present, the filling grade of queues increase and thereby the latency due to queuing increases. As the queues fill up the packet-loss rate increases.

The varying factors of packet-loss and latency are highly dependent upon the congestion level of the Internet [19, 26, 30, 37].

3.1.1 Congestion

Congestion is a term used when the arriving rate of traffic at a link is higher than the bandwidth capacity of the link.

E.g. a link constantly sending $5\,\mathrm{Mbit\,s^{-1}}$ of data into a link only able to send at a rate of $4\,\mathrm{Mbit\,s^{-1}}$, then clearly this is too much data for this link to handle, and this

link is then congested. The buffers of this link would rapidly overflow and a packet-loss rate corresponding to $1\,\mathrm{Mbit\,s^{-1}}$ would result.

A way to understand congestion problems is to look into the past. In 1986 a series of "congestion collapses" hit the Internet, slowing down the actual throughput by a factor of a thousand [19, 26]. According to Jacobson [26] this was due to the implementation of TCP used at the time, `4.3BSD TCP`. The next section describes the problem as Van Jacobson saw it.

3.1.2 Tackling Congestion

Jacobson [26] states that TCP is self-clocking, meaning a new packet is sent upon an `ACK`. To start the clock, at first packets must be sent without an `ACK`. The way that `4.3BSD TCP` started the clock, too many packets were sent at the beginning.

Figure 1 shows that the `ACK`'s of the receiver are spaced out according to the delay in the network. This results in a state where the time between the packets sent, is exactly the same as the time it takes a packet to flow through the bottleneck link. However, if too many packets are sent at the beginning, and there is not enough room for the packets in the buffers, there is congestion at once, resulting in high packet-loss.

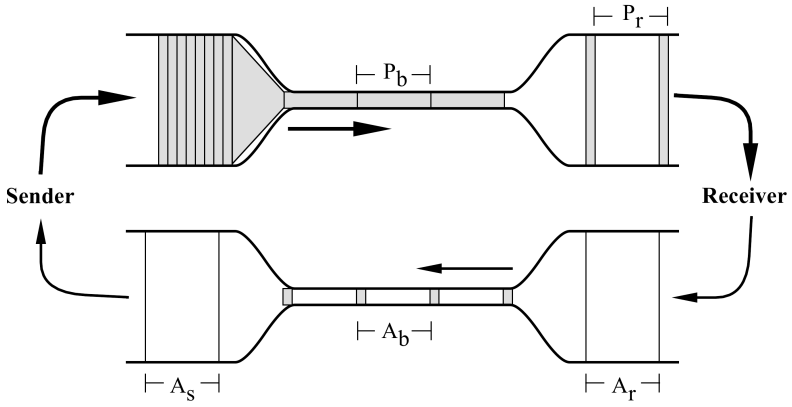


Figure 1: Window Flow Control 'Self-clocking', from Jacobson [26]

Figure 2 shows the start of a TCP flow without using slow-start. The window size of the TCP connection was 32 packets, and total storage in the network path is large enough to accommodate the window size, but the buffer at the bottleneck gateway can only accommodate 30 packets. Each dot in figure 2 represents a packet number, two dots on the same y-axis but different x-axis represents a retransmit. The dotted line is the optimum transfer. Figure 2 shows many retransmits, and the utilization of the available bandwidth was 35 % [26].

Therefore TCPs slow-start was developed by Jacobson [26]. The slow-start algorithm ensures that the rate at which data is sent does not exceed twice what is possible on the path.

Figure 3 shows the same plot as figure 2 but with slow-start implemented. There are

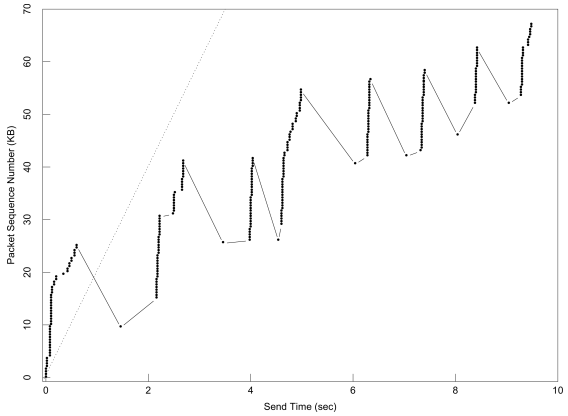


Figure 2: Startup behavior of TCP without Slow-start, from Jacobson [26]

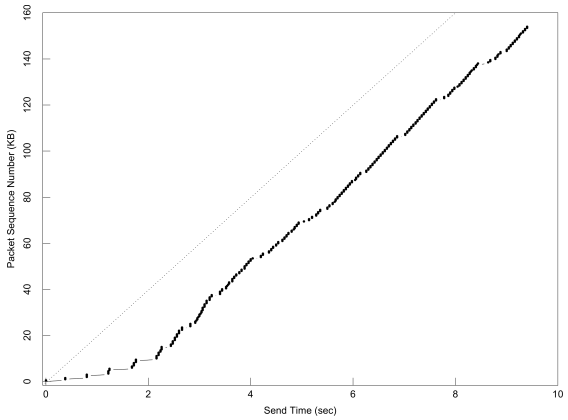


Figure 3: Startup behavior of TCP with Slow-start, from Jacobson [26]

considerably less retransmits and the effective bandwidth utilization after one minute with slow start would be $19\,\mathrm{kB\,s^{-1}}$ of the available $20\,\mathrm{kB\,s^{-1}}$, i.e. 95 %.

Jacobson [26] pointed to two more problems that could lead to too many packets being sent in bursts: incorrect round trip time estimator leading to retransmission of non `ACK`ed packets too early, and resources along the communication path not being able to adequately handle equilibrium throughput resulting in packet-loss due to buffer overflow.

Jacobson [26] presents a new algorithm to determine the Round Trip Time (RTT) to remedy the early packet retransmission problem. Additionally they present a congestion avoidance algorithm to remedy the lack of resources problem. Figure 4 shows multiple TCP flows without congestion avoidance transmitting data over a shared communication path. 4 000 out of 11 000 packets were retransmissions, and the division of resources was uneven with one flow getting $8\,\mathrm{kB\,s^{-1}}$, two flows getting $5\,\mathrm{kB\,s^{-1}}$, one getting $0.5\,\mathrm{kB\,s^{-1}}$ and $6\,\mathrm{kB\,s^{-1}}$ not being utilized.

Figure 5 shows multiple TCP flows with congestion avoidance transmitting data over

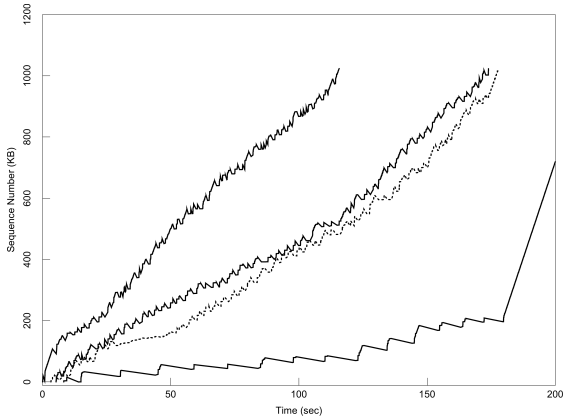


Figure 4: Multiple, simultaneous TCPs with no congestion avoidance, from Jacobson [26]

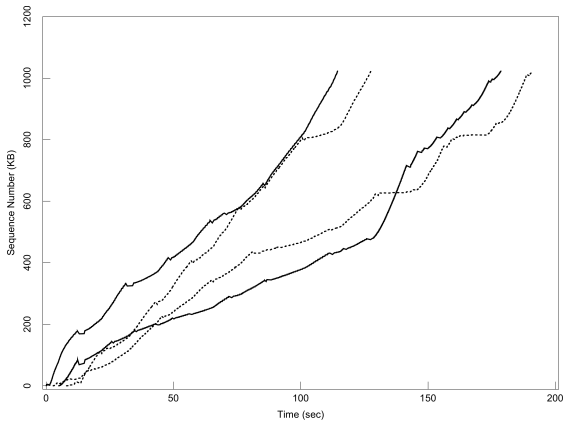


Figure 5: Multiple, simultaneous TCPs with congestion avoidance, from Jacobson [26]

a shared communication path. 89 out of 8281 packets sent were retransmissions, and the division of resources was more even. Two flows got $8\,\mathrm{kB\,s^{-1}}$ and two flows got $4.5\,\mathrm{kB\,s^{-1}}$, utilising all of the available bandwidth.

The congestion avoidance algorithm presented by Jacobson [26] can be summarised as:

1. On any timeout, set the congestion window, W_i, to half the current window size W_{i-1}, see (2) (multiplicative decrease).

2. On each `ACK` for new data, increase the congestion window, W_i, by $1/W_{i-1}$, see (3) (additive increase).

3. When sending, send the minimum of the receiver's advertised window and the congestion window.

$$W_i = dW_{i-1} \quad (d < 1) \tag{2}$$

$$W_i = W_{i-1} + u \quad (u \ll W_{max}) \tag{3}$$

The rationale behind this is an uncongested network at interval i has the load L_i given in (4), where N is constant. But load under congested conditions is given by (5), where γ is the fraction of data left over from the last interval. In a heavily congested network γ is great and the queue lengths will start increasing exponentially. If the amount of traffic does not decrease, the load L_i is always increasing. The amount of traffic needs to decrease more than the queues are increasing.

$$L_i = N \tag{4}$$

$$L_i = N + \gamma L_{i-1} \tag{5}$$

The efforts by Jacobson [26] and others to develop congestion control and avoidance algorithms are the first approaches to a more robust and stable operating Internet. Since then several additions to the congestion avoidance scheme of TCP have been made, but the basic idea is still the same, large decrease of the congestion window during congestion and small increase during non congested periods to probe for more bandwidth.

3.1.3 Buffering

In a heterogeneous packet switching network like the Internet, buffering is paramount to keep the infrastructure cost down. If there are no buffers, a global synchronization scheme would have to be adopted [19]. The arriving packets at the bottleneck link must be stored somewhere while the bottleneck link is busy. Therefore buffers are placed in almost every device in the network with a link.

Problems determining the size of buffers exist. The ground breaking work done by Nagle [30] showed that the buffers should not be too large, otherwise a 100 % packet-loss might occur. Furthermore Villamizar and Song [38] recommend that buffers be equipped with queueing capacity greater than or equal to the Bandwidth-Delay Product (BDP), but with Random Early Detection (RED). The BDP is a measure of how much data a given path trough a network can store. The BDP is given by (6), where l_{rtt} is the RTT of the path and C is the rate of the bottleneck link on the path.

$$BDP = l_{rtt} \cdot C \tag{6}$$

The recommendation made by Villamizar and Song [38] lead to the adoption of BDP sized buffers almost in all devices. The adoption of RED is a different story, covered more in depth in the next section and section 3.2.3. BDP sized buffers, with size B, are calculated by (7),where $\bar{l}_{rtt}$ is the average RTT of a flow utilising the link, and C the immediate outgoing link's bandwidth.

$$B = \bar{l}_{rtt} \cdot C \tag{7}$$

3.1.4 The Bufferbloat Problem

This section is mainly based on Gettys [19] article about bufferbloat. Gettys [19] starts off stating: "Today's networks are suffering from unnecessary latency and poor system performance. The culprit is bufferbloat, which is the existence of excessively large and frequently full buffers inside the network." Gettys [19] does not claim to be the first to announce this problem, but they try to leave no doubt that bufferbloat is a problem. They proclaim that bufferbloat is very often mistaken for congestion, leading to the wrong solutions. Sometimes even making the situation worse.

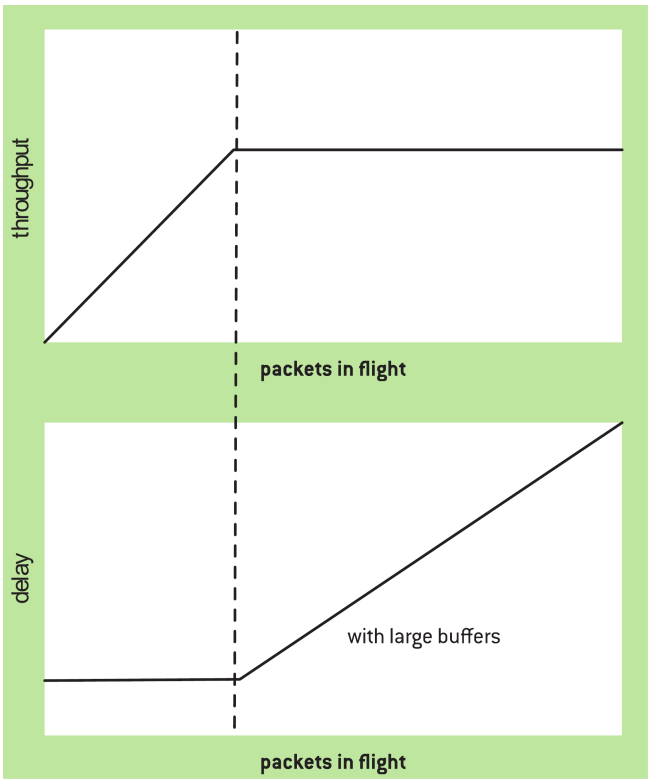


Figure 6: Throughput and delay as packets in flight increase, altered from Gettys [19]

Figure 6 shows how throughput increases as packets in flight increases until the bottleneck rate is reached. Furthermore, the figure shows how the delay is constant until excess queueing due to overbuffering starts becoming a factor, as the bottleneck rate is reached [19, 37].

TCP has a window of packets in flight, filling the imaginary pipe in the network, and therefore is fully dependent upon buffers at some critical points in the network, to not suffer heavy packet-losses. But congestion avoidance algorithms only throttling back the

window at packet-loss fill the entire buffer until buffer overflow occurs. As the buffers are filled, the queuing delay increases. Van Jacobson recognised the "persistently full buffers" problem while working on congestion avoidance, and started the work on RED with Sally Floyd [19].

Floyd and Jacobson [16] developed RED which is an Active Queue Management (AQM) scheme. The idea behind AQM is that the gateways or routers should signal congestion to TCP and similar protocols by e.g. dropping packets, so that the protocol could rewind its window and ease the congestion.

However,Gettys [19] argues that AQM is not widely used. Furthermore they state that the existence of cheap memory and the misguided desire to avoid packet-loss has led to larger and larger buffers being deployed in the devices making up the internet.

TCP is impacted in two major ways by bufferbloat [19]:

- TCPs reaction time to competing flows to avoid congestion is quadratic to the amount of overbuffering. A 10 times too large buffer results in 10 times the latency and a 100 times the reaction time to congestion.
- Competing flows might be starved of bandwidth due to the slow reaction time.

Appropriate buffer sizing has been researched extensively recently, and this is covered more in-depth in section 3.2.2. Gettys [19] does point to a difficulty with static sizing of buffers though, i.e. rapidly changing available bandwidth. E.g. in wireless network cards, even a 256 packet buffer at the lowest Wi-Fi rate of $1\,\mathrm{MB\,s^{-1}}$ can add three seconds of delay. This suggests that a dynamic approach should be used to solving the bufferbloat problem.

Gettys [19] states that the advent of Content Delivery Networks (CDNs) has brought common RTTs down to 10 ms from 30 ms. Therefore even if the bandwidth is constant the delay might change over time resulting in the wrong buffer size. Gettys [19] is of the opinion that researchers tuning TCP implementation for networks with high BDP, have worked with a model of high bandwidth and AQM enabled buffers, not realising that the large pipe size is because of large buffers and not caused by high bandwidths, and therefore tuning TCP to fill these buffers efficiently. Both Xu et al. [39] and Ha et al. [23], presenting Binary Increase Congestion (BIC) and CUBIC respectively, take a high BDP and the deployment of AQM into account.

Gettys [19] argues that there cannot be "correct" static sized buffers, and that a self adaptive AQM is the only viable long term solution in *any* device.

3.2 Mitigations

This section will present some of the mitigations tried to solve congestion, and bufferbloat problems in today's Internet.

3.2.1 TCP tuning

What usually drives TCP tuning is the desire to achieve more bandwidth utilization, better congestion control, and less packet-loss. The drive behind BIC and CUBIC was indeed the quest to utilize more of the bandwidth in what is known as long fat networks, i.e. networks with a high BDP, while still being able to exist alongside standard TCP flows.

BIC

From the desire to utilize more of the bandwidth in high-speed networks with high delays, many congestion control and avoidance algorithms have been proposed focusing on TCP friendliness and bandwidth scalability. However,Xu et al. [39] argue that these suggestions suffer from RTT unfairness.

Therefore Xu et al. [39] suggest a different congestion control called BIC control. BIC combines both additive increase and binary search increase. For large congestion windows, large additive increase should ensure linear RTT fairness and good scalability. Furthermore for small congestion windows the binary search increase should provide TCP friendliness.

BIC uses the maximum window to deploy a binary search within this window size, to find an equilibrium. If the jump from the minimum to the midpoint between the maximum and the minimum is too large, the algorithm uses an additive increase instead of the binary search increase.

CUBIC

Ha et al. [23] present CUBIC, a TCP friendly high-speed TCP variant. The growth of the window is dictated by a cubic function. The cubic function uses the elapsed time from the last congestion event. This RTT independent growth of the window ensures that CUBIC entails more equivalent resource allocation between flows with different RTTs. CUBIC is the current default TCP algorithm in `Linux`.

When the congestion window is far from the equilibrium, the window is increased rapidly. When it is close, the window is increased more slowly. This ensures scalability in high BDP networks and at the same time stability and fairness towards standard TCP flows. The window growth function is given by (8), where C is a CUBIC parameter, t is the elapsed time from the last window reduction, and K is the time period that (8) takes to increase W to W_{max} when there is no further loss event and is calculated by (9).

$$W(t) = C(t - K)^3 + W_{max} \tag{8}$$

$$K = \sqrt[3]{\frac{W_{max}\beta}{C}} \tag{9}$$

3.2.2 Buffer Sizing

The rule of thumb was for a long time that the buffer size should as stated in 3.1.3 be the BDP. The discovery of bufferbloat and difficulties implementing BDP sized buffers has lead to more research into the necessity of BDP sized buffers.

Appenzeller et al. [6] argue that BDP sized buffers are outdated, because backbone links multiplex a large number of TCP flows. Appenzeller et al. [6] argues that for a single TCP flow to keep a bottleneck link fully utilized the buffer must hold enough packets to keep the link fed from a packet-loss is recorded until the sender restarts transmission, as seen in Figure 7.

Furthermore the number of outstanding packets at the time of decreasing the window size W is equal to $W_{max}/2$, if the window size is halved. The time it takes the bottleneck link to empty $W_{max}/2$ packets from the buffer at rate C is $W_{max}/2/C$. I.e. the sender's pause lasts for $W_{max}/2/C$. The time it takes the buffer of size B to empty is B/C. To ensure that the buffer does not empty, (10) must hold. W_{max} can be determined by (11), because the sending rate, R, of TCP is $R = W/RTT$ and R must be equal to C, T_p is propagation time between sender and receiver.

$$B \geq \frac{W_{max}}{2} \tag{10}$$

$$C = \frac{W}{RTT} = \frac{\frac{W_{max}}{2}}{2T_p} \tag{11}$$

Solving for $W_{max}/2$ leads to (12), which is the BDP.

$$B \geq 2T_p \cdot C = \overline{RTT} \cdot C \tag{12}$$

However,when there are multiple TCP flows, above 500, to avoid synchronization effects, (12) is unnecessary large. The maximum buffer size needed is argued by Appenzeller et al. [6] to be given by (13), with buffer size B, average round trip time of $\overline{RTT}$, link capacity of C, and n number of flows. This could reduce buffer sizes by 99 % [6].

$$B = \frac{\overline{RTT} \cdot C}{\sqrt{n}} \tag{13}$$

Appenzeller et al. [6] defines the queue occupancy at time t for n flows in (14), where $W_i(t)$ is each flows window size at time t, T_p is still propagation delay, C is the links bandwidth, ϵ is dropped packets. They argue that if the buffer is large enough and TCP is operating correctly, then ϵ is negligible compared to $(2T_p * C)$. The distribution of $Q(t)$ is then given by (15), and plotted in Figure 8. Notice how the queue occupancy is almost never below 10 500 packets.

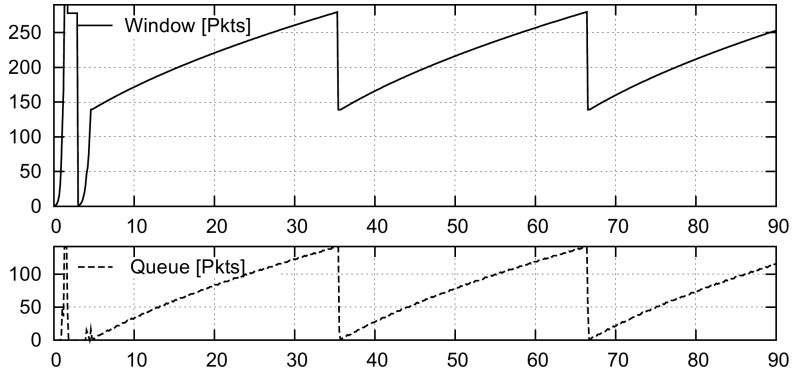


Figure 7: A single TCP flow's congestion window in the top graph through a router with BDP sized buffers, and the queue length in the buffer in the bottom graph. From Appenzeller et al. [6]

$$Q(t) = \sum_{i=1}^{n} W_i(t) - (2T_p \cdot C) - \epsilon \tag{14}$$

$$Q \stackrel{d}{=} W - 2T_p \cdot C \tag{15}$$

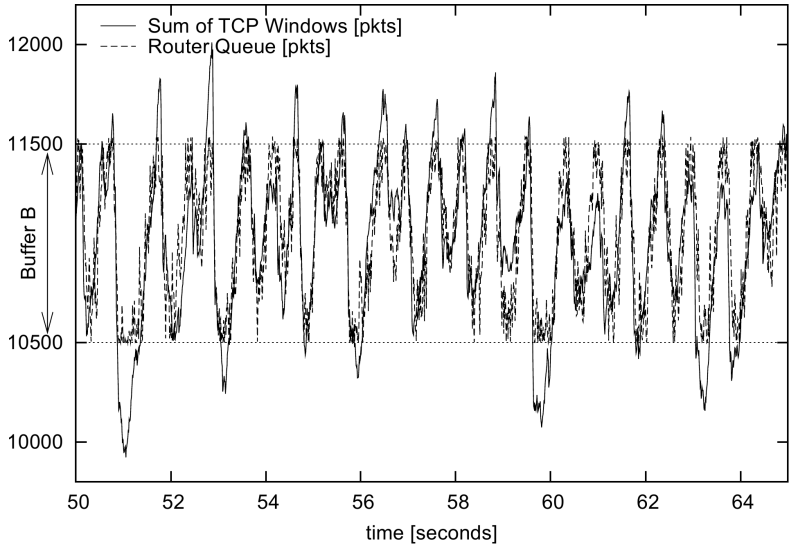


Figure 8: Plot of $\sum W_i(t)$ of all TCP flows, and of the queue, Q, offset by 10 500 packets. From Appenzeller et al. [6]

W has a normal distribution, therefore Q has a normal distribution restricted to the allowable range for Q shifted by a constant [6]. Because Q has a known distribution a buffer size can easily be checked for underbuffering or overbuffering. Proof of (13) can then be found.

Appenzeller et al. [6] verifies the buffer size given by (13) through experiments. Beheshti et al. [8] reports that they have built confidence that the size given by (13) will hold in general for backbone routers, by performing real world experiments.

Beheshti et al. [7] further decrease the buffer sizing to a logarithmic size of the congestion window, W, for smooth TCP traffic. Because the maximum window, W_{max}, of a TCP is a function of the rate of a link, the buffer sizing only needs to be increased logarithmically as long as the traffic is smooth, without sacrificing too much of the link utilization.

Small buffers are not only beneficial regarding reduced delay, Beheshti et al. [9] talks about optical packet buffers, and states that it is both difficult and expensive to implement large optical buffers. They argue that, using amongst others results from Appenzeller et al. [6] and Beheshti et al. [7], buffers only need to hold 20 packets per linecard, and that this is possible to implement on photonic chips.

3.2.3 Active Queue Management

Usually, without Active Queue Management (AQM), a packet is dropped because of buffer overflow, i.e. there is no memory to store the packet. This is called a drop-tail scheme.

The quest to solve congestion problems lead Van Jacobson and Sally Floyd to working on algorithms to manage the queues at gateways in a more elaborate fashion. Floyd and Jacobson [16] suggests the RED algorithm placed in gateways estimating the average queue size. Based on this packets are either dropped, with a calculated drop probability, or marked to notify of congestion. The latter notification method requires support of the protocols ensuring packet flow.

AQM is way of altering the queue at a gateway, router or similar device to ensure a certain behaviour of the traffic flowing through it. There are a many different schemes to actively manage queues including: RED from Floyd and Jacobson [16], and Proportional Integral (PI) controller from Hollot et al. [24].

Nichols and Jacobson [31] continues the work of RED and similar schemes to remedy the bufferbloat problem. They state that a good AQM should:

1. be parameterless. I.e. the scheme does not need tuning to work on two different links.
2. treat good queues and bad queues differently. I.e. keep delay low while allowing bursts.
3. control delay, while being insensitive to RTTs, link rates, and traffic loads.
4. adapt to dynamically changing link rates with no negative impact on utilization.
5. be simple and efficient. I.e. be implementable in a large variety of devices.

Nichols and Jacobson [31] state that no known schemes fulfil the characteristics mentioned above. The scheme proposed in Nichols and Jacobson [31] is Controlled Delay

(CoDel), pronounced "coddle". Nichols and Jacobson [31] state that "this is a "no-knobs" AQM that adapts to changing link rates and is suitable for deployment and experimentation in Linux-based routers, as well as silicon." This is partly verified through their simulations and experiments. In 2012 CoDel was adopted into the Linux kernel. CoDel has shown better performance than both drop-tail and RED. Importantly, in a simulated consumer edge experiment CoDel showed equal or less packet-loss compared to drop-tail.

Proportional Integral controller Enhanced, PIE

Pan et al. [33] presents a lightweight design proportional integral controller for AQM, Proportional Integral controller Enhanced (PIE). They argue that PIE does not require per packet processing , and has self tuning parameters. Furthermore they show through simulation results, theoretical analysis and Linux testbed results that PIE can ensure low latency and achieve high link utilization under varying congestion situations.

The goals of Pan et al. [33] are to achieve:

- low latency control. While RED and similar schemes manages the queue length, and latency is handled implicitly the aim of PIE is to achieve control of latency. This is because the queue size changes with different draining rates, and different RTTs.
- high link utilization.
- a simple implementation. The aim is to ensure scalability and ease of use by simple implementation.
- guaranteed stability and fast responsiveness. The controller should be stable for various network topologies.
- automatically set performance parameters. Only performance-related parameters like target latency should be required.

The foundation of PIE is that of the PI controller. I.e. PIE uses standard control theory to achieve a stable latency. PIE is basically a control-loop with a drop probability p as output, and the calculated latency as input.

The controller is given by (16), where at iteration k, p_k is the current drop probability, p_{k-1} is the previous drop probability, l_k is the current latency, l_{ref} is the desired latency, l_{k-1} is the previous latency. α and β constitute the gain of the control-loop, or scaling factors as Pan et al. [33] present them. The update of drop probability is run periodically with period T_{update}.

$$p_k = p_{k-1} + \alpha \cdot (l_k - l_{ref}) + \beta \cdot (l_k - l_{k-1}) \tag{16}$$

The latency, l_k, in PIE is derived from Little's law given in (17), describing the average amount of customers L in a store, with the average arrival rate of λ and the average time spent in the store W. The current latency, l_k, is given by (18), where n_{queue}

is the length of the queue in bytes, and $\overline{C}$ is the average drain rate of the queue (the rate of the link).

$$L = \lambda \cdot W \tag{17}$$

$$l_k = \frac{n_{queue}}{\overline{C}} \tag{18}$$

PIE's auto tuning of the parameters α and β follow the scheme:

$$\begin{aligned} \alpha = \frac{\hat{\alpha}}{8},\ \beta = \frac{\hat{\beta}}{8} \quad & p < 1\% \\ \alpha = \frac{\hat{\alpha}}{2},\ \beta = \frac{\hat{\beta}}{2} \quad & p < 10\% \\ \alpha = \hat{\alpha},\ \beta = \hat{\beta} \quad & p \geq 10\% \end{aligned}$$

Pan et al. [33] include a way of estimating the average drain rate of the queue to ensure that the scheme will work on links with variable transmission rate. To ensure that the measurement of drain rate is not afflicted by empty queues from short non-persistent bursts, the estimation of drain rate is only done when the queue has more than a certain number of bytes, $n_{threshold}$. The threshold, $n_{threshold}$, dictates how fast the response to fluctuations in drain rate is.

The estimation algorithm of drain rate, $\overline{C}$, is as follows:

- Upon departure of a packet, if the length of the queue, n_{queue}, is greater than the threshold, $n_{threshold}$, increase the number of bytes since last estimation, $n_{departed}$, by the size of the packet departing.
- Upon the increase of number of bytes, $n_{departed}$, above the threshold, $n_{threshold}$, calculate the parameters listed by (19) through (21) where ϵ is an averaging parameter, and then reset the number of departed bytes, $n_{departed}$, counter.

$$\Delta t = t_k - t_{k-1} \tag{19}$$

$$C_{measured} = \frac{n_{departed}}{\Delta t} \tag{20}$$

$$\overline{C}_k = (1 - \epsilon) \cdot \overline{C}_{k-1} + \epsilon \cdot C_{measured} \tag{21}$$

To allow bursts Pan et al. [33] include a max burst length parameter, b_{max}, so that upon packet arrival the probability drop is bypassed if the time, b_{allow}, since, $p = 0$, $l_k < l_{ref}/2$, and $l_{k-1} < l_{ref}/2$, is less than b_{max}. The default value for b_{max} is 100 ms. b_{allow} is updated when $C_{measured}$ is updated by decreasing b_{allow} by Δt

In simulations to verify the functionality of PIE Pan et al. [33] used the default values, $l_{ref} = 20$ ms, $T_{update} = 30$ ms, $\hat{\alpha}$ =0.125 Hz, $\hat{\beta} = 1.25$ Hz, $n_{treshold} = 10$ kB, and

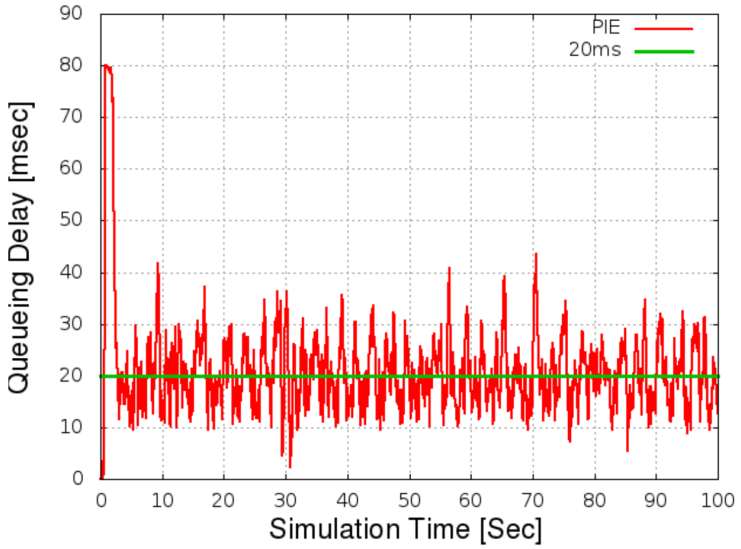


Figure 9: Queueing delay with a mix of five TCP and two UDP flows. From Pan et al. [33]

$b_{max} = 100$ ms. The simulations were done over a 10 Mbit s^{-1} link with a RTT of 100 ms and a buffer of 200 kB.

Simulations show that PIE functions as it should. Fair bandwidth division, full utilization of the link's capacity, and burst allowance is achieved. Figure 9 shows that with a load of five TCP flows and two User Datagram Protocol (UDP) flows each transmitting at 6 Mbit s^{-1}, PIE manages to control the queueing delay around 20 ms. Figure 10 shows that for the same simulations as in Figure 9 the throughput utilization is 100 %.

Figure 11 shows the queuing delay of both PIE and RED in a simulation with a buffer of 2 MB, and a varying bandwidth of 100 Mbit s^{-1} from 0 s to 50 s, 20 Mbit s^{-1} from 50 s to 100 s, and 100 Mbit s^{-1} from 100 s to 150 s. RED's configuration parameters are as follows, $min_{th} = 20\%$, $max_{th} = 80\%$, $max_p = 0.1$, and $q_{weigth} = 0.002$.

Pan et al. [33] state that because RED only controls the queue length and not the actual queue delay, the queue delay changes as the bandwidth changes. Furthermore PIE controls the latency and therefore reacts accordingly. A similar simulation where the number of TCP connections were varied shows that RED can not control the latency effectively because it is statically configured, while PIE's autotuning feature adjusts accordingly.

Pan et al. [33] performed a testbed comparison with CoDel, with an implementation of PIE in the `Linux` kernel. The implementation of CoDel used was that in `Kernel Version 3.5-rc1`. First an experiment with 20 TCP NewReno flows with 100 ms RTTs flowing over a 10 Mbit s^{-1} link was performed. This showed that both PIE and CoDel could control the queueing delay to the references of 5 and 20 ms. The bandwidth utilization of both PIE and CoDel for both references were above 96.6%.

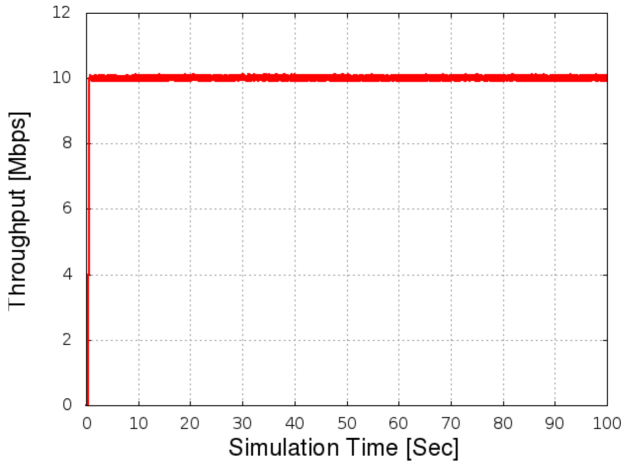


Figure 10: Throughput with a mix of five TCP and two UDP flows. From Pan et al. [33]

Figure 12 shows the cumulative distribution function of the queue delay for both PIE and CoDel, with five TCP, and two UDP flows of 6 Mbit s^{-1} over a 10 Mbit s^{-1} link. The figure shows that CoDel can not handle the congestion that is imposed by these flows. The two UDP flows alone result in traffic that is 20% above what the link can actually serve.

A theoretical analysis by Pan et al. [33] shows that PIE is a stable controller, and that the suggested values for α and β together with the autotuning feature ensures stability. The analysis will not be repeated here. Since PIE uses standard control theory, standard control theory stability analysis is sufficient.

As of `Linux Kernel Version 3.14` PIE was included as an alternative algorithm to combat the bufferbloat problem.

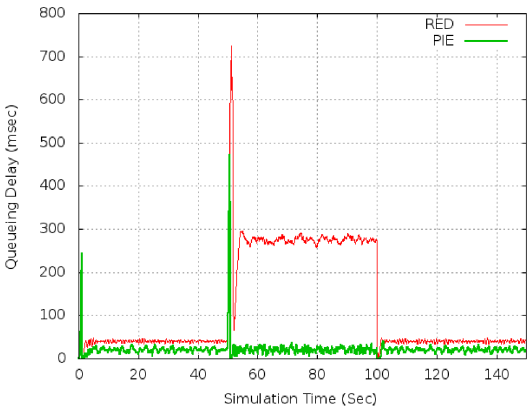


Figure 11: Performance comparison between PIE and RED with varying link capacity. From Pan et al. [33]

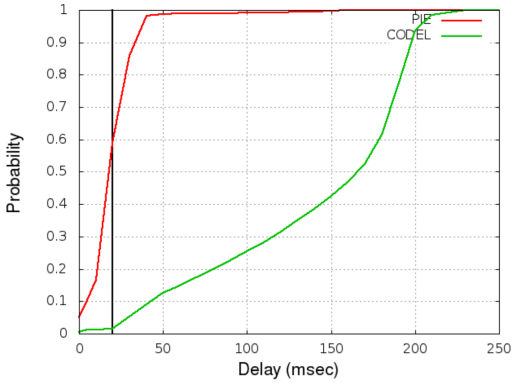


Figure 12: CDF curve of queueing delay for both PIE and CoDel, with 20 ms reference delay, five TCP, and two UDP flows. From Pan et al. [33]

3.2.4 Differentiated Services

Differentiated services is the idea of servicing different traffic according to different rules, to achieve different QoS goals. Pan et al. [33] state that their PIE scheme can be implemented side by side with differentiated services in several forms. They mention different queues with different queue delay references for different traffic and different drop probability per flow.

Differentiated services are supported by modern Internet Protocol (IP) networks.

3.3 Uncertainties in Internet Communications

Floyd and Paxson [17] state that modelling and simulation of the Internet is not an easy task, and say that often real measurements are needed for a crucial "reality check" to challenge implicit assumptions. They argue further that measurements can only depict the situation at the moment of measurement. The speed at which new software and features can propagate through the internet entails that the situation can be a totally different one in a short timespan.

Floyd and Paxson [17] state that the Internet is difficult to characterise because the IP protocol is made to "unify diverse networking technologies", and hence the Internet is made up of a large variety of devices with wildly different characteristics.

The Internet is a moving target. What is state of the art today might not be state of the art tomorrow, and what is large scale today might be small scale in a short time. E.g. from January 1997 to December 2000 the number of connected computers increased from 16 million to 100 million, a 60% yearly increase.

Heterogeneity is widespread, because the Internet is built by competing entities who might not want to share topological information, there exists no typical Internet topology. Even TCP, the most used protocol, exists in many different implementations.

Floyd and Paxson [17] list some ways that the Internet can change drastically:

1. **Scheduling:** Routers, nodes and gateways connected to the Internet might change their scheduling algorithms, e.g. from FIFO and drop-tail to PIE.

2. **Wireless:** A new connection technology comes along and changes the characteristics of connected entities in a large manner, e.g. as has already happened with cellular internet connectivity.

3. **Differentiated services:** Can lead to different connections having widely different performances.

4. **A new "killer app":** While web traffic today is dominating the Internet's traffic there might come along a new "killer application" that changes the dominant traffic entirely.

However,Floyd and Paxson [17] additionally list promising invariant methods to describe the Internet:

1. **Diurnal patterns of activity:** Different times of day, week and month shows different amounts of traffic. Also different protocols show different loads throughout time. These diurnal patterns might be predicted and used in simulation

2. **Self-Similarity:** On a small time scale, i.e. non diurnal and on a millisecond to minutes scale, packet arrivals are not smooth.

3. **Poisson session arrivals:** Poisson processes with different hourly rates following the diurnal patterns describe Network user "session" arrivals well. I.e. when users start an FTP transfer or log on to a terminal. However,the arrivals of the multiple

network connections comprising each session are not well described by Poisson processes.

4. **Log-normal connections sizes:** The lenght of a connections is well described using a log-normal distributions.

5. **Heavy-tailed distributions:** Characterization of network activity are often heavy-tailed, i.e. follow a Pareto distribution with shape parameter $\alpha < 2$.

6. **Invariant distribution for Telnet packet-generation:** Users typing at a keyboard has an invariant distribution. The distribution has a Pareto upper tail and a Pareto body.

7. **Invariant characteristics of the global topology:** While there are many different and rapidly changing Internet topologies, there are some aspects that will never change. I.e. aspects related to geographical location. E.g. There will always be a lower bound on the RTT delay from Paris to New-York because of the limitation on the speed of light.

3.4 Internet Connectivity in Norway

This section presents information about the Norwegian internet, what type of technology is used for internet connectivity in private housing and a summary of a report on robustness in Norwegian cellular networks.

3.4.1 The Norwegian Internet eXchange

The Norwegian Internet eXchange (NIX) are six connections points provided by four universities in Norway to make it easy for Internet Service Providers (ISPs) to interconnect on a national and regional scale. The six connection points are located in: Bergen, Oslo (two connections), Stavanger, Trondheim, and Tromsø. These are the main connection points between several ISPs in Norway, and the main connection point to the rest of the world in Norway.

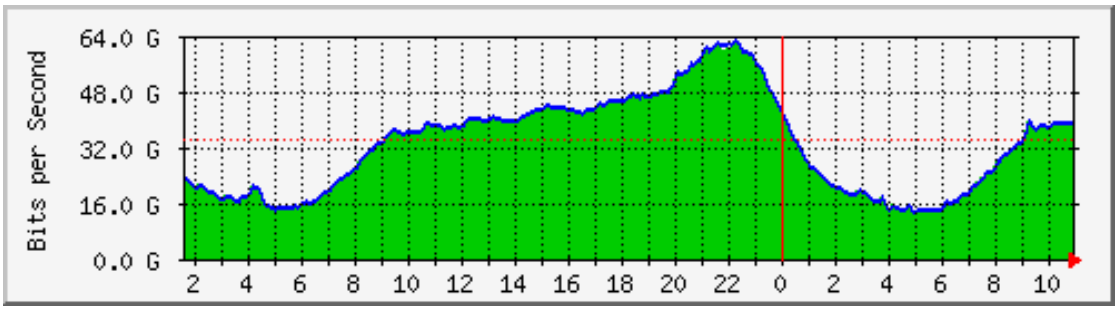


Figure 13: Daily traffic graph, five minute average, shows traffic from approximately 02:00 on 13. May to approximately 11:00 on 14. May 2014. The graph is taken from [4].

Figure 13 shows a graph of the daily traffic, as seen at 11.02 on 14. May 2014, at NIX. This is not all traffic in Norway, but a large portion of it. The maximum traffic in Figure 13 is at approximately 22:00 on 13. May 2014.

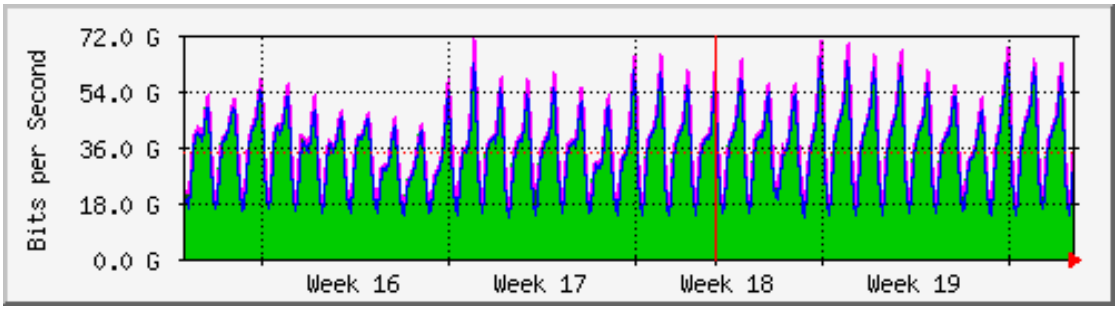


Figure 14: Monthly traffic graph, two hour average, shows traffic from approximately 16. May to approximately 14. May 2014. The graph is taken from [4].

Figure 14 shows a graph of the monthly traffic, as seen at 11.02 on 14. May 2014, at NIX. This figure shows that the daily traffic showed in Figure 13 has the same shape

for most days.

3.4.2 Internet Connectivity Technology

The Internet connectivity of private homes in Norway can briefly be summarised in Table 3. The percentages showed in the table are the percentages of Norwegian households that have access to and subscribe to an internet connection with a given technology. The number of broadband connected homes in Norway has been steadily increasing over the last few years.

Table 3: Norwegian household Internet subscriptions. From Statistisk Sentralbyrå (SSB) [36]

Technology	**2009**	**2010**	**2011**	**2012**	**2013**
Broadband Internet	78 %	83 %	80 %	86 %	88 %
ADSL or xDSL	58 %	57 %	48 %	42 %	37 %
Cable (TV)	19 %	21 %	22 %	28 %	29 %
Fiber Cable	9 %	12 %	14 %	17 %	23 %
Radio or Satellite	2 %	2 %	6 %	8 %	12 %
Cell phone with UMTS/HSDPA	12 %	22 %	29 %	48 %	54 %
Cell phone with GSM/GPRS	23 %	24 %	12 %	12 %	4 %
Dial-Up	37 %	37 %	29 %	24 %	18 %
Dial-Up Modem	11 %	11 %	.. %	.. %	.. %
ISDN	11 %	10 %	.. %	15 %	14 %
Other	3 %	2 %	2 %	.. %	.. %

3.4.3 Robustness in Norwegian Cellular Networks

Simula is a Norwegian research laboratory. The Centre for Resilient Networks and Applications (CRNA) is a part of Simula. In February 2014 CRNA released a state report for 2013 on the robustness of Norwegian cellular networks CRNA [13]. Since this thesis focuses on latency issues to achieve a certain reliability in communication, the main findings and some key aspects of CRNA's report follow. Unfortunately this report is only available in Norwegian at the time of writing this thesis. The main focus of the report has been the perceived robustness of Norwegian cellular networks. The report tries to describe the stability of the service that the end-user experiences.

The data has been collected from nodes in the `NORNET` multi homing project. The measurement terms are as follows: connection stability, data delivery stability and performance stability. Connection stability is a measure of how frequent downtimes are and the duration of downtimes. Data delivery stability is a measure of the ability to deliver data once the connections is up. Performance stability is a measure of how predictable and even the performance is.

Connection Stability

The main conclusions regarding connection stability are:

1. There are significant downtimes, i.e. the connection to the cellular network is interrupted, in several connections. Between 15-37% of the connections loose their connection more than ten minutes a day on average, depending on provider.
2. There are major differences in the downtime patterns between network providers.
3. 2G connections are less stable than 3G connections with all providers. 2G connections loose their connections in 10 % of all hours during the measurement period, while 3G connections loose their connection 4 % of all hours during the measurement period.
4. There is a strong correlation between stability of connection and the quality of the signal. Connections with a weak signal and/or high interference experience more frequent and longer downtimes.

Data Delivery Stability

The main conclusions regarding stability of data delivery are:

- Many connections experience high packet-loss rates. Between 49% and 18% of the connections experience more than 1% packet-loss depending on provider.
- There are considerable differences in packet-loss between providers.
- Packet-loss varies throughout the day and night. Packet-loss is greater at periods of high network load.

Increased Robustness Through Multi Homing

Multihomed networks are networks where there are two or more gateways from one network to another network, usually involving address translation. E.g. a weather station with a Local Area Network (LAN) and one cable link (Asymmetric Digital Subscriber Line (ADSL)) and one wireless link to the internet is then multi homed. Multi homing can be used to increase uptime of a service and to increase throughput. There are however routing difficulties when using multi homing though.

A list of conclusions found in the report regarding the prospects of multi homing follows:

- There is a great degree of independence in signal quality amongst different providers on the same geographical location. This is positive for the prospects of multi homing.
- By connectiing two independent network providers over 50 % of the nodes achieve "five nines" on connection uptime, i.e. at least one network providers connection is up 99.999 % of the time.

- There is a relatively high degree of independence in packet-loss between connections to different providers at the same node. The dependency is greater for providers sharing a network infrastructure.

3.5 Quantitative Characterization and Measurement Techniques

Most of the research presented so far is focused research, i.e. research with a problem to identify the source of and solve. A lot of this research has emerged from larger studies where e.g. the broadband connections of several thousand homes have been monitored and characterised with respect to including bandwidth utilization, and experience of latency. These studies are important to locate areas where more research is needed.

This section will present the measurement techniques used to characterize internet connections including those used by two large characterization studies, and the main findings in their dataset.

3.5.1 Measurement Techniques

There are several ways of measuring different aspects of a broadband connection. As Sundaresan et al. [37] state and show in figure 15, various methods give different results. This section will present the measurement techniques used to characterize internet connections including those used by two large characterization studies.

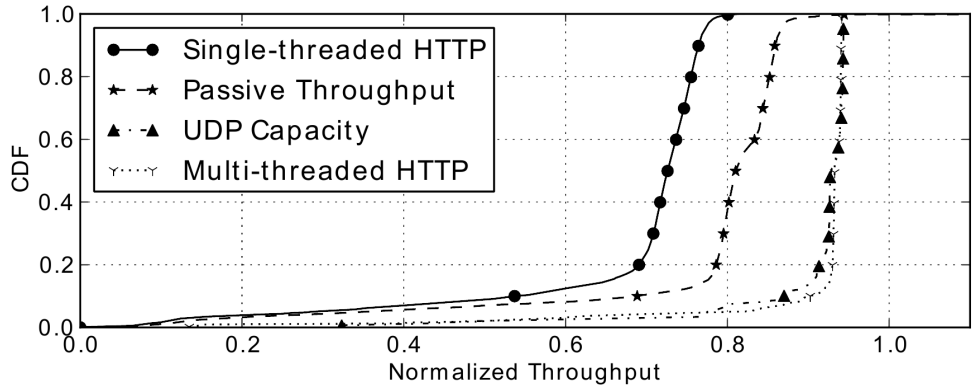


Figure 15: Comparison of various methods of measuring throughput. From Sundaresan et al. [37]

Employing the End-User

There are several online services providing both delay and bandwidth measurements, e.g. `speedtest.net`. These are frequently used by end-users to measure their Internet service quality. Gettys [19] and Sundaresan et al. [37] state that these measure transient network quality, and therefore do not describe the connection adequately. They give an example: tests on a connection with a service plan providing 6 Mbit s^{-1} download and 512 kbit s^{-1} upload speed yielded 4.4 Mbit s^{-1} down and 140 kbit s^{-1} up from `speedtest.net`, 4.8 Mbit s^{-1} down and 430 kbit s^{-1} up from `Netalyzr`, and 5.6 Mbit s^{-1} down and 452 kbit s^{-1} up from long term measurements.

As pointed out by Gettys [19], most of these services, not including `Netalyzr`, do not measure latency under load, and therefore do not provide any indication of problems like bufferbloat.

A Centralised Measurement Technique

Maier et al. [29] observed network activity of 20 000 residential Digital Subscriber Line (DSL) customers in an urban area. Their work is based upon packet-level observations collected at aggregation points within a large European ISP. The monitor is placed at the broadband access router, linking customers to the ISP backbone.

This is what is known as packet sniffing. Packets sent through the network are analysed and information is extracted from the header fields. By gathering enough packets from a flow, metrics including latency and throughput can be estimated. Furthermore, this is a great way to measure the composition of the traffic in the network.

A View From the Gateway

Sundaresan et al. [37] utilize measurement software implemented at the end-user gateways. It includes measurements from nearly 4 000 gateway devices across eight different ISPs.

According to Sundaresan et al. [37] deploying the measurements at the gateways offer advantages including:

- Direct measurement of the ISPs access link. The gateway sits behind the customer's modem. Furthermore, the effects of wireless and cross-traffic can be isolated.
- Continuous/longitudinal measurements.
- The ability to instrument a single home with different hardware and configurations. All conditions in the network but one can be kept the same, e.g. as done in the study the modem can be swapped.

Figure 16 shows the gateway used by Sundaresan et al. [37] placed at an end-user. All traffic to and from the end-user flows through the gateway.

Sundaresan et al. [37] argue for the following conclusions regarding measuring bandwidth:

- Different throughput measurement techniques capture different aspects of throughput.
- A single-threaded TCP session is sensitive to packet-loss. Augmenting this measurement with passive usage measurements improves its accuracy.
- Multi threaded TCP and the UDP capacity measurements measure the access link capacity more accurately and are more robust to loss.

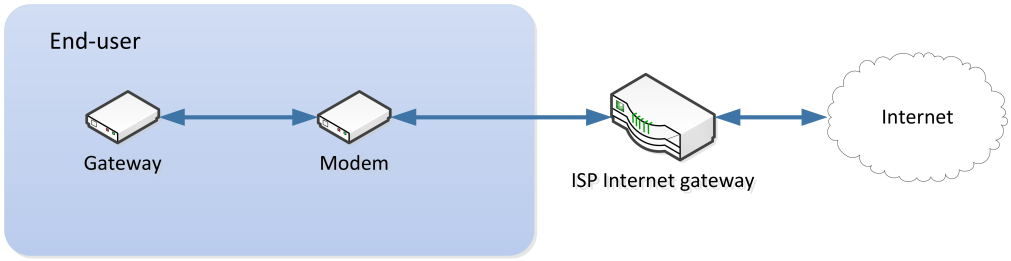


Figure 16: Placement of gateway at end-user used in Sundaresan et al. [37]

Sundaresan et al. [37] use two large gateway deployments to gather data: `SamKnows` and `BISMark`. `SamKnows` uses the `WNR3500L v1` router described in appendix B.1. `BISMark` uses the `NOX Box`, a small-form-factor computer described in appendix B.3. Table 4 shows the number of deployments for each gateway at the ISPs.

The `SamKnows` gateway measures the bulk transfer capacity, the amount of data a congestion-aware protocol like TCP can transfer, by spawning three parallel threads with a Hypertext Transfer Protocol (HTTP) client, increasing the likelihood of saturating the link. The actual measurement is done after a warmup period, to ensure that the measurement is not affected by TCP slow-start.

Further `SamKnows` measures latency in three different ways.

1. end-to-end latency: with a UDP client sending approximately 600 packets an hour, recording both packet-loss and latency.
2. latency to the first IP hop inside the ISP, last mile latency: with Internet Control Message Protocol (ICMP) ping sending 5 packets an hour.
3. latency under load: by measuring end-to-end latency during the bandwidth capacity measurements.

The `SamKnows` gateways additionally measure jitter and the download times of ten popular websites. All tests are started only if cross-traffic on the link is below $64\,\mathrm{kbit\,s^{-1}}$ down, and $32\,\mathrm{kbit\,s^{-1}}$ up. If these limits are exceeded during the test, the test is aborted.

The `BISMark` gateways measure bulk transfer capacity by setting up a single-threaded HTTP connection performing both download and upload regardless of cross-traffic. To account for cross-traffic the gateway records the counter for transmitted bytes in the network card, and compute the difference between before and after the test, the "passive throughput". According to Sundaresan et al. [37] this yields the combined throughput of the HTTP transfer and the cross-traffic. Capacity is measured with a tool called `ShaperProbe`.

The latency measurements of `BISMark` are similar to those of `SamKnows`. End-to-end latency to a nearby wide-area host, last-mile latency using `traceroute`, and latency under load to the last-mile router. Measurements are performed regardless of cross-traffic.

Table 4: Total `SamKnows` and `BISMark` deployments. From Sundaresan et al. [37]

ISP	Technology	`SamKnows`	`BISMark`
Comcast	Cable	864	4
AT&T	DSL/FTTN	787	10
TimeWarner	Cable	690	-
Verizon	DSL/FTTP	551	-
Cox	Cable	381	-
Qwest	DSL/FTTN	265	-
Charter	Cable	187	-
Cablevision	Cable	104	-

3.5.2 Performance and Characterization Data

This sections presents the main results and findings gathered in the studies described in the previous section.

Dominant Characteristics at DSL Customers

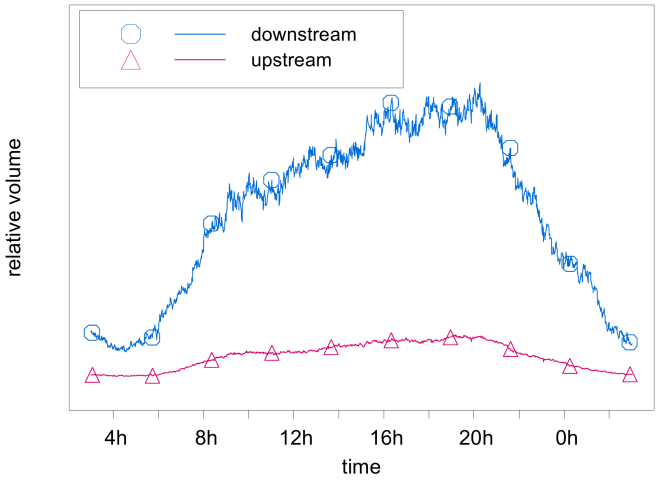
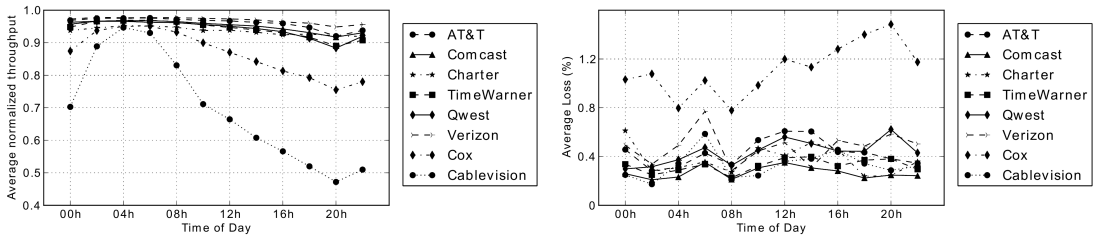


Figure 17: Bandwidth usage of all DSL connections over the course of a day, 1 minute bins. From Maier et al. [29]

Maier et al. [29] list some results that they found surprising:

- HTTP traffic dominates making up nearly 60% of the total traffic. Their surprise was that it was not peer-to-peer traffic that dominated.
- DSL sessions are short in duration. The median connected time between a modem and the line card is 20 min to 30 min. Resulting from this 50% of IP addresses are therefore reassigned at least twice every 24 h.



(a) Average normalised throughput.

(b) Average packet-loss.

Figure 18: Diurnal packet-loss rates and normalised throughput, average throughput divided by the 95^{th} percentile. From Sundaresan et al. [37]

- The latency experienced from a end-user to the ISP's Internet gateway is often greater than the latency experienced in the Internet. A 46 ms median from end-user to gateway and 17 ms median in the rest of the path was experienced.
- There is rarely 100% utilization of the available bandwidth. They state that it appears TCP settings in the residential systems and wireless networks limit the throughput.

Figure 17 shows the diurnal patterns on bandwidth utilization that emerged in Maier et al. [29].

A View From the Gateway

Figure 18a and 18b show that the average normalised throughput and packet-loss is mostly consistent throughout the day for all ISPs but Cablevision's throughput and Cox's throughput and packet-loss.

Sundaresan et al. [37] saw that cable providers in general have lower last-mile latency and jitter than providers utilising different access technology. Furthermore baseline latencies for DSL users may vary significantly based on physical factors such as distance to the Digital Subscriber Line Access Multiplexer (DSLAM) or line quality. The last-mile latency generally makes up 40 - 80 % of the end-to-end path.

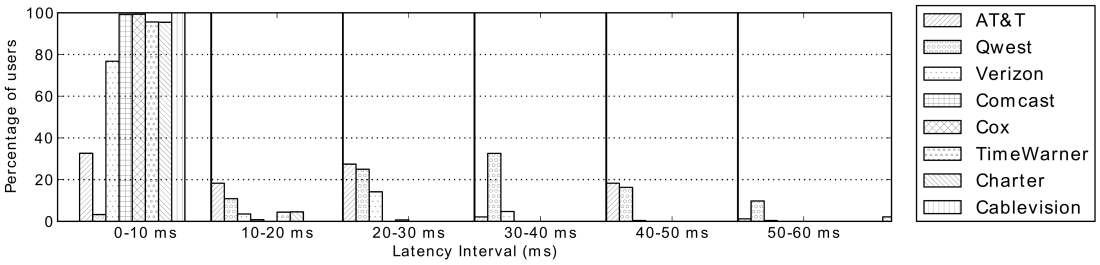


Figure 19: Baseline last mile latency, tenth percentile. From Sundaresan et al. [37]

Bufferbloat is confirmed by the latency under load measurements of Sundaresan et al. [37]. Figure 19 shows the baseline last mile latency found in Sundaresan et al. [37] and Figure 20 shows the factor by which latency increases with load. The histogram in Figure 20 shows the latencies when the download and upload buffers are saturated independently. Sundaresan et al. [37] argue that the higher increase factor shown in Figure 20 for upload saturation might be because the upload rate is usually slower than the download rate, and the same buffer sizes might be used both in uplink and downlink.

Sundaresan et al. [37] argues that because ISPs offer several different service plans, with different rates, and they usually only offer one modem with one static buffer size, bufferbloat will be a problem if not handled by managing the queues.

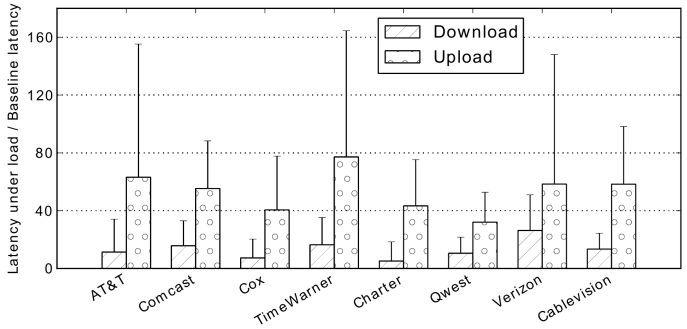


Figure 20: The factor by which latency increases during load. From Sundaresan et al. [37]

Figure 21 shows the latency of three different modems on a DSL link with 6 Mbps download rate and 512 kbit s^{-1} upload rate flooded with 1 Mbit s^{-1} of UDP traffic using `iperf` after 30 s for a duration of 30 s. The figure shows that the latency experienced drastically increases when the link is saturated. Sundaresan et al. [37] emulated the same setup with buffer sizes of 512 kbit $s^{-1} \cdot l_{max}$, where l_{max} is the maximum experienced delay. This produced an almost perfect replica of Figure 21. The same emulation with a buffer size of 20 kB gave no decrease in throughput performance, suggesting that these buffers indeed are too large.

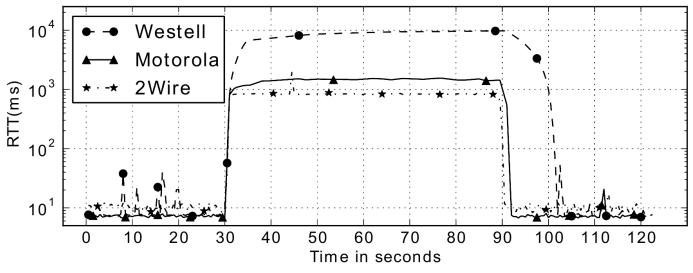
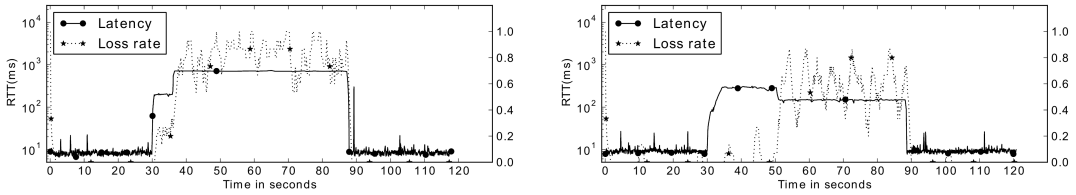
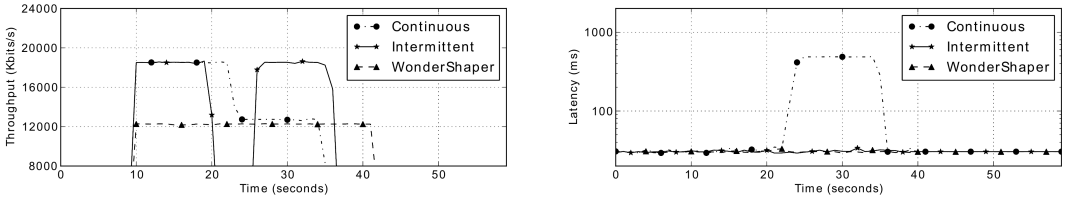


Figure 21: Buffering in three different AT&T modems. From Sundaresan et al. [37]



(a) Comcast `D-LINK` modem (b) Comcast `RCA Thomson` modem.

Figure 22: Latency increase and packet-loss due to overbuffering. From Sundaresan et al. [37]



(a) Throughput. (b) Latency.

Figure 23: Shaped traffic's impact on latency. From Sundaresan et al. [37]

Figures 22a and 22b show the same experiment performed to produce Figure 21 only with a DSL link with 12.5 Mbit s^{-1} download rate and 2 Mbit s^{-1} upload rate, and upload flooded with 10 Mbit s^{-1}. According to Sundaresan et al. [37] the two step increase in delay in Figure 22a is because of PowerBoost. A feature that some ISPs enable, which yields higher throughput for a short period offer. However,Figure 22b shows a two step decrease in delay. At the same time as the delay decreases the packet-loss increases. Sundaresan et al. [37] states that this might correspond to the behaviour of dynamic buffer sizing or AQM.

Figures 23a and 23b show a test of traffic shaping [37]. The figures shows the traffic and delay during unshaped traffic (plot marked continuous), during intermittent traffic at the link capacity (plot marked intermittent, and during traffic shaped using `WonderShaper`. `WonderShaper` was a big hit for Internet cafés and the like in the early 2000's.

These two figures show that it is possible to have low latency and still achieve approximately the same transfer time for a file. Notice that all three approaches finishes almost at the same time.

4 Characterizing Norwegian Internet Last Mile Links

One of the aims of this thesis is to uncover if the last mile links, the link between end-user and ISP, in the Norwegian Internet suffer from high and varying latencies due to bufferbloat, and either lack of or wrongly tuned AQM.

Bufferbloat is the existence of buffers that are so large that they increase the latency of a link without increasing utilization. Lack of or faulty AQM schemes do not improve latencies.

After a search for literature on the presence of bufferbloat or AQM in Norwegian Internet last mile links it was determined that there is a lack of sufficient literature. One of the research reports found was CRNA [13] only concerning cellular networks in Norway. While this is interesting, it does not provide information of how good a wired Internet last mile link can be in Norway.

Floyd and Paxson [17] state that "Measurement is needed for a crucial "reality check", when talking about simulation of the internet. Therefore it was decided to perform a characterization of two or more Internet last mile links. Preferably two or more sites with different link technology and different characteristics.

Additionally experimentation might add additional insight into the impact of bufferbloat and AQM on deploying real-time applications over household last mile links.

4.1 Selection of Experiment Sites

Section 3.4.2 presents Internet connectivity statistics in Norway. The percent of private housing in Norway using ADSL or x Digital Subscriber Line (xDSL) links to the Internet is 37 %, Table 3. ADSL is asymmetric and can not deliver as high bandwidth as e.g. fiber and cable links. ADSL was therefore chosen as a possible characterization candidate. A fiber connection to the Internet is seen as the future of Internet connectivity, and is viewed as the best way of connecting to the Internet in Norway. Fiber connections have not been very common in comparison to ADSL connections and cable connections, but are beginning to be more and more common, Table 3. Therefore a characterization of a fiber connection to see how good an Internet connection can be in the future is interesting.

The house at Rindadalsvegen 30 on Askøy uses a ADSL connection and was chosen as a candidate for characterization. This connection is served by the ISP Telenor, and the service plan subscribed to is "Bredbånd 8". The download rate in the plan is $8\,\mathrm{Mbit\,s^{-1}}$, and the upload rate is $700\,\mathrm{kbit\,s^{-1}}$. A quick test at speedtest.net shows a ping of 29 ms, a download rate of $8.3\,\mathrm{Mbit\,s^{-1}}$ and a upload rate of $620\,\mathrm{kbit\,s^{-1}}$. In demographic terms this household consists of two people in their 50's.

The house at Tistelvegen 24 at Nardo in Trondheim uses a fiber connection and was chosen as the second candidate for characterization. The ISP is NTE and the service plan subscribed to is "Internet 120". The download rate in the plan is $60\,\mathrm{Mbit\,s^{-1}}$, and the upload rate is $60\,\mathrm{Mbit\,s^{-1}}$. A quick test at speedtest.net shows a ping of 17 ms, a download rate of $61.59\,\mathrm{Mbit\,s^{-1}}$ and a upload rate of $55.61\,\mathrm{Mbit\,s^{-1}}$. In demographic terms this household consists of four people in their 20's.

Additionally an office at NTNU Gløshaugen was picked as a candidate, mainly because this office has an internet IP, and is not behind a Network Adress Translation (NAT) router.

4.2 Choice of Measurement Techniques

Sundaresan et al. [37] argue that the choice of measurement technique is very important. Different methods have different results when trying to measure the same aspects.

Section 3.5.1 presents some different approaches to measurement techniques. The approach with a packet sniffer used in Maier et al. [29] will be difficult to use because this entails instalment of hardware in the ISP's own network, which might be challenging to do.

The use of measurement software on a computer behind the gateway at the end-user would have similar issues as using online measurement services like `speedtest.net`. The issue of transient network behaviour could be circumvented by running the measurements continuously though. A latency under load measurement could be performed. But cross-traffic from other devices behind the same gateway could not be accounted for.

Sundaresan et al. [37] argue that the best way to characterise an Internet connection is to deploy measurement software to the gateway at the end-user. They argue that this removes the issue with cross-traffic and it is easy to get continuous measurements. They present a way to measure the "passive" throughput at the gateway, i.e. bandwidth utilization, by recording the amount of data being sent on the network card at the gateway.

4.3 Hardware at Sites

The sites at Rindadalvegen 30 and Tistelvegen 24 are equipped with `NETGEAR WNR3500lv2` routers. The `WNR3500lv2` is equipped with 128 MB NAND flash and 128 MB RAM. It uses a 480 MHz MIPS processor and has `gigabit` capability. More information on `NETGEAR WNR3500lv2` is available in appendix B.2.

Figures 24 and 25 show how the modems and gateways are connected at Rindadalsvegen 30 and Tistelvegen 24 respectively. The gateway at Rindadalsvegen 30 sits directly behind the modem provided by Telenor and all other devices including wireless devices are connected through this gateway.

The gateway at Tistelvegen 24 however sits behind another gateway, because this link is shared between two households and the other household did not want their traffic recorded. The measurements at this location might therefore suffer from this.

A `DELL optiplex 9010` desktop computer is located at the NTNU Gløshaugen office. More information about this computer can be found in appendix B.4.

4.4 Choice of Measurements to Perform

The focus of this thesis is latency issues. Therefore active bandwidth measurements are not done. However,the "passive" throughput is measured, i.e. the bandwidth utilization.

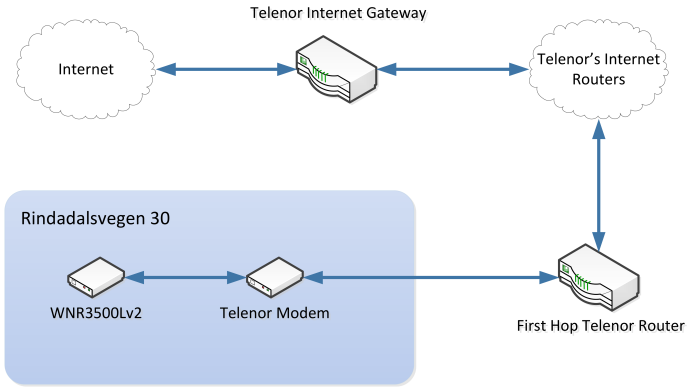


Figure 24: Hardware setup at Rindadalsvegen 30.

A high resolution bandwidth utilization measurement together with a high resolution active delay measurement should be sufficient to uncover issues with bufferbloat, AQM, and congestion.

4.5 Implementations of Measurement Tools

Measurement tools are needed to characterize the Internet connections at each site in terms of "passive" throughput, i.e. amount of traffic, and latency. The open source project continued by Michal Rupental, `TomatoUSB` can run on the routers at Rindadalsvegen 30 and Tistelvegen 24. `TomatoUSB` provides the ability to run `optware`, and was therefore chosen as router firmware.

Installing `TomatoUSB` on the routers is fairly straight forward. The package provided in appendix A can be flashed from the stock firmware in the update firmware section. If this fails, which happened on the router at Rindadalsvegen 30, there are numerous tutorials online on how to use trivial FTP to flash the firmware.

Optware provides a small packet repository to install different tools, including development tools, see appendix C.1.2 for more information on `optware` and appendix C.1.1 for a step-by-step guide to developing software in `c` on TomatoUSB with `optware`.

However,there are no measurement tools provided by `optware` to passively measure bandwidth utilization at higher rates than once a second and the tool provided to measure latency, `ping`, can not do so at higher rates than once a second. Therefore new measurement tools were implemented.

4.5.1 Traffic Monitor

The traffic monitor tool was implemented to monitor and record the amount of traffic on the communication link to the Internet at each site. It runs on the router at each site and is written in `c`.

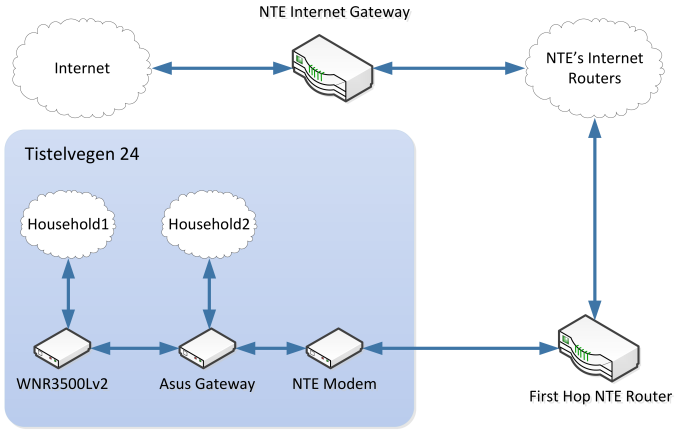


Figure 25: Hardware setup at Tistelvegen 24.

Similar to the `BISMark` project presented in Sundaresan et al. [37] the traffic monitor measures the bandwidth utilization passively, and does not perform active measurements. The traffic monitor only records the amount of bytes both sent and received on the outbound link.

The traffic monitor takes the measurement interval, a silent flag, a prefix, and name of the network interface as input. Respectively this is the interval in milliseconds that the monitor should try to record the bandwidth utilization at, a flag set if no command line output is wanted, the prefix of recorded data (`k` or `m`), and the name of the network interface that is to be monitored.

The data collected is stored in a file with a name on the form: `traffic yyyy-mm-dd HH-MM.csv`. Where the date and time is the date and time the file was created. For each measurement interval, time of measurement on the format `yyyy-mm-dd HH:MM:SS:FFF`, time since last measurement in seconds, and the calculated throughput for this interval in bps with a voluntary prefix is stored. Every day at 00:00 the traffic monitor creates a new file so that data can be copied without stopping the monitor.

The throughput is calculated by subtracting the counters found in:

- `/sys/class/net/NIC_NAME/statistics/rx_bytes`
  and:
- `/sys/class/net/NIC_NAME/statistics/tx_bytes`

from the same counter for the last interval and dividing by the elapsed time since last measurement.

Some of the code and more information can be found in appendices C.2.1 and D.1. All source code and makefile is provided in appendix A.

4.5.2 Delay Monitor

The delay monitor tool was implemented to actively probe the outgoing link's latency. It consist of a client that sends UDP packets of minimum size, and a server that responds to the packets. Each packet contains a number incremented for each packet, so that each packet can be uniquely identified.

The delay monitor client takes the measurement interval, a silent flag and an IP address as input. Respectively these are the interval in milliseconds that the client sends a new UDP packet, a flag set if no command line output is wanted, and the IP address to the delay monitor server. The server only sends back the same packets that it receives and has no inputs.

The data collected by the client is stored in files with names on the form: `delay yyyy-mm-dd HH-MM.csv`, `dropped yyyy-mm-dd HH-MM.csv`, and `log yyyy-mm-dd HH-MM.csv`. Where the date and time is the date and time the file was created. For each measurement interval, the packet number, time of packet departure and time of packet response on the format `yyyy-mm-dd HH:MM:SS:FFF`, RTT in seconds, RTT in milliseconds and a calculated moving average of the RTT in milliseconds is stored in the `delay yyyy-mm-dd HH-MM.csv` file. For each packet that the RTT exceeds 10 s, an entry in the `dropped yyyy-mm-dd HH-MM.csv` file is made. Every day at 00:00 the traffic monitor creates new files for the delay and dropped recordings so that data can be copied without stopping the monitor. The log file are only closed when the program is halted.

Some of the code and more information can be found in appendices C.2.2 and D.2. All source code and makefile is provided in appendix A.

4.5.3 Problems Developing on TomatoUSB

During the measurement period, the file system of `optware` became corrupt, and with it all collected data. Therefore there are some gaps in the collected data. The file system might have become corrupt because of the frequent writes imposed by the measurement tools. Also the use of old memory sticks might have been a problem, although one of the memory sticks, one that got locked in read only mode in the memory stick firmware, was brand new.

At first the measurement tools created a new file every time they were started, and continued to write to this file until it was stopped. This meant that if the tool was not frequently stopped and data was retrieved and the system got corrupt, the potential loss of data was large.

After a few incidents where several days worth of data was lost, a new implementation of the tools was done. Now the tools create a new file every day at midnight, so that the data can be copied without stopping the measurement tool, and the potential loss of data is less for a corruption incident.

To further decrease corrupt behaviour, the file system used on the memory sticks was switched from `ext2` to `ext3` because the journaling feature of `ext3` might prevent some corrupt behaviour.

4.6 Description of Long Term Experiments

The measurement tools at Rindadalsvegen 30 where run by connecting to the router using `ssh` from the computer at the NTNU office. This means that the terminal session running the tools is running over this same link being characterised. It was difficult to run it in different manner. This does have potential problems, and will be discussed accordingly.

The measurement tools at Tistelvegen 24 were run by connecting to the router using `ssh` from a computer on the local network behind the router.

A long term experiment where the tools were left running for a long period of time, at least one week, was performed.

4.6.1 Traffic Measurement

The bandwidth utilization at each site is recorded at 100ms intervals.

The traffic monitor described in 4.5.1 is run with the following command at both Rindadalsvegen 30 and Tistelvegen 24:

```
#  ./TrafficMonitor -s -p k -i 100 -I vlan2
```

`vlan2` is the name of the wide area network interface of the routers.

4.6.2 Delay Measurement

The latency at each site is measured at 100ms intervals.

The delay monitor client described in 4.5.2 is run with the following command at both Rindadalsvegen 30 and Tistelvegen 24:

```
# ./DelayMonitor -s -i 100 -a 129.241.154.125
```

`129.241.154.125` is the IP address of the server running at the computer at the NTNU office.

4.6.3 Router Load

Together with the traffic measurements and delay measurements the `dstat` tool is run to record load on the router hardware and to record the bandwidth utilization at 1s intervals.

`dstat` is run by the following command:

```
# dstat -c -n -N vlan2 -t -output dstat yyyy-mm-dd HH-MM.csv
```

This records the load on the cpu and the traffic on `vlan2` in bytes at 1s intervals and stores the output in `dstat yyyy-mm-dd HH-MM.csv`

4.7 Description of Short Term Experiments

Additionally, some more direct approach measurement are performed. These experiment still utilize the same measurement tools, but under more controlled conditions. Both the experiments where performed with only one computer attached to the gateway and wireless access switched off.

4.7.1 Different Application's impact on Traffic and Latency

This experiment is done to analyse the impact and behaviour of different applications in terms of latency and traffic.

First a download over FTP is performed, followed by an upload over FTP. Furthermore some HTTP browsing and a `skype` session was performed while FTP was still transferring.

4.7.2 First Hop Latency Measurement During Saturation

This experiment is done to show that the increase in latency during saturation is at the last mile link.

An upload and a download of a large file is performed separately saturating the link. During these uploads the RTT to the first hop with an Internet IP is measured. The RTT is measured using either `traceroute` or ping. Ping might not always work because the first hop might not respond to ping or ping messages might be prioritised by ISP gateways.

This was performed only at Rindadalsvegen, due to difficulties saturating the link at Tistelvegen.

5 Measurement Results

5.1 Comments on Measurements

Packet-loss rate is calculated from the delay monitor, i.e. it is not actual packet loss of the link, it is inferred packet-loss rate. Unless otherwise noted, delay and latency is the measured RTT by the delay monitor. Reliabilities are calculated by (1).

5.2 Rindalsvegen 30

5.2.1 Entire Measuerment Period

The measurements at Rindadalsvegen 30 were collected from May 4. 15:55 to May 24. 23:59. There are gaps in the measurements, some because the measurement tool crashed, and some because the measurements were deemed faulty. The gaps are presented in Table 5. They have been taken out before calculating any metrics unless otherwise noted.

Table 5: Gaps in measurements of RTT at Rindadalsvegen 30.

From	To	Reason
May 4. 23:59	May 5. 00:59	Bug resulting in too many files. Fixed after this incident.
May 5. 03:00	May 5. 10:00	Increase in baseline RTT. See section 5.2.6 and 6.1.1.
May 5. 15:00	May 5. 18:30	Increase in baseline RTT. See section 5.2.6 and 6.1.1.
May 9. 11:42	May 9. 15:50	Connection lost and tool crashed.
May 10. 09:00	May 10. 10:00	Increase in baseline RTT. See section 5.2.6 and 6.1.1.
May 11. 06:00	May 11. 13:42	Connection lost at 06:58:19. See section 5.2.7 and 6.1.1.
May 12. 22:00	May 13. 08:30	Increase in baseline RTT. See section 5.2.6 and 6.1.1.
May 15. 16:00	May 15. 17:14	Tool crashed.
May 19. 03:00	May 19. 08:18	Connection lost at 03:01:24. See section 5.2.7 and 6.1.1.

Table 6 presents some statistical metrics of the measured RTT from May 4. to May 25. Table 7 presents the calculated latency reliability for some different latency requirements during the same period.

Metric	Value
Packets sent	14 405 250
Recorded packets dropped	2092
Recorded packet-loss	0.0145%
Median of RTT	19 ms
SD of RTT	72
variance of RTT	5 253

Table 6: Calculated statistical metrics of the RTT at Rindadalsvegen 30, May 4. to May 25.

Requirement [ms]	Reliability
30	97.65%
100	98.21%
200	98.60%
300	98.74%
500	99.32%
700	99.94%

Table 7: Reliability for different RTT latency requirements at Rindadalsvegen 30, May 4. to May 25.

Figure 26 shows the measured RTT, download traffic, and upload traffic from May 4. to May 25.

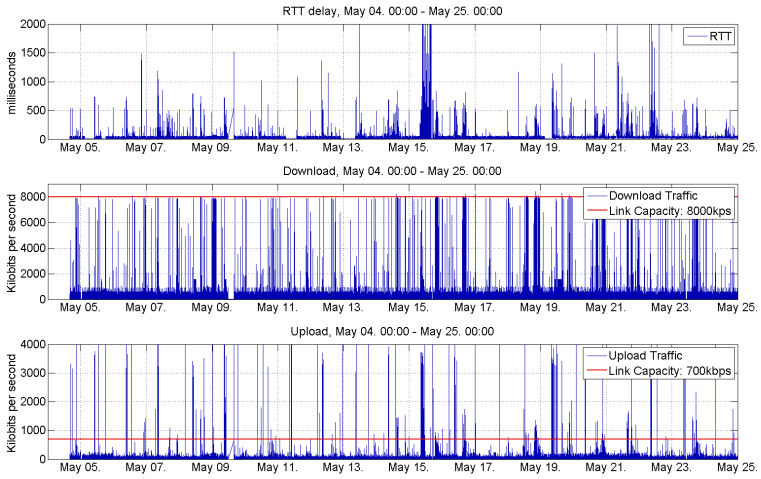


Figure 26: RTT delay, download traffic, and upload traffic at Rindadalsvegen 30, May 4. to May 25.

Figure 27 shows the cumulative distribution function of the measured RTT delay at from May 4. to May 25.

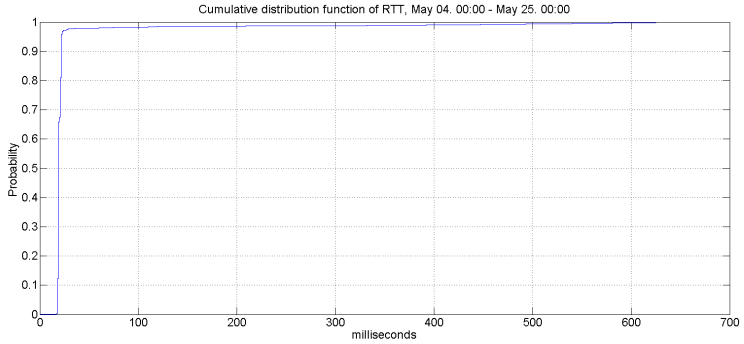


Figure 27: Cumulative distribution function of RTT delay at Rindadalsvegen 30 May 4. to May 25.

5.2.2 Daily Traffic

Figure 28 shows a bar graph of the total amount of traffic in MB both upload and download at Rindadalsvegen 30.

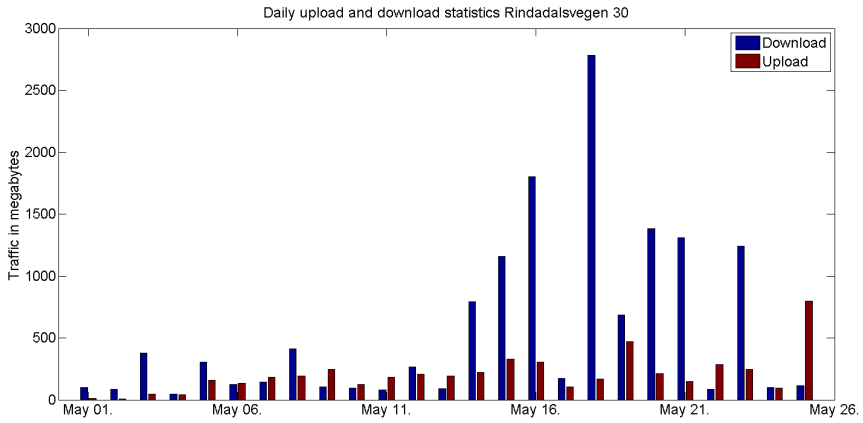


Figure 28: Daily download and upload statistics in megabytes at Rindadalsvegen 30.

5.2.3 First Hop Latency During Saturation

At Rindadalsvegen 30 ICMP ping showed to have a max RTT of 25 ms to the first hop during both upload and download. However, `traceroute` under the same conditions showed considerably higher RTTs. The delay monitor showed the same results as `traceroute`. In this experiment `traceroute` is therefore used at Rindadalsvegen 30.

Table 8: RTT measured to the first hop and second hop during upload saturation at Rindadalsvegen 30.

Samples	**First hop $\overline{RTT}$**	**Second hop $\overline{RTT}$**	**First hop SD**	**Second hop SD**
24	478 ms	498 ms	88.5	88.3

Table 9: RTT measured to the first hop and second hop during download saturation at Rindadalsvegen 30.

Samples	**First hop $\overline{RTT}$**	**Second hop $\overline{RTT}$**	**First hop SD**	**Second hop SD**
21	263 ms	272 ms	5.0	2.1

5.2.4 Measurements from May 16.

Figure 29 shows the RTT delay and the traffic over the gateway at Rindadalsvegen 30 throughout May 16. Table 10 lists statistical metrics of the RTT during this period, and Table 11 lists the reliability for different RTT latency requirements

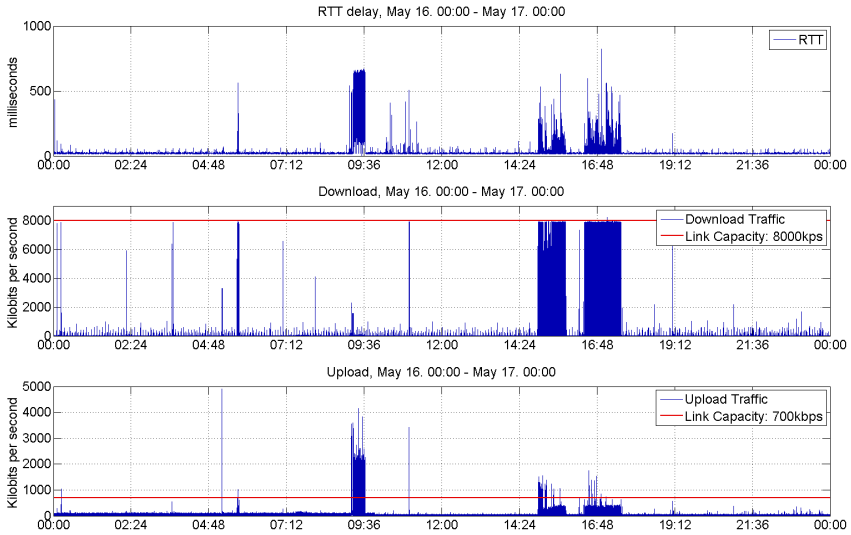


Figure 29: RTT delay, download traffic, and upload traffic at Rindadalsvegen 30 on May 16. 2014

Metric	Value
Pings sent	771 986
Pings dropped	63
Packet-Loss	0.008%
Median of RTT	19ms
SD of RTT	62
variance of RTT	3 786

Table 10: Calculated statistical metrics of the RTT at Rindadalsvegen 30 throughout May 16.

Requirement [ms]	Reliability
30	96.34%
100	97.54%
200	98.00%
300	98.42%
500	99.24%
700	99.98%

Table 11: Reliability for different RTT latency requirements at Rindadalsvegen 30 on May 16.

Figure 30 shows the cumulative distribution function of the measured RTT on May 16. Figure 31 shows the same data as Figure 29 only smoothed.

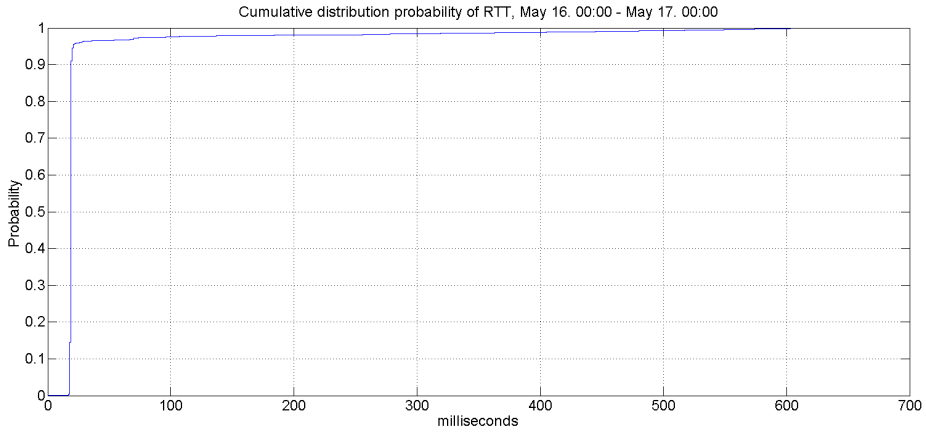


Figure 30: Cumulative distribution function of RTT at Rindadalsvegen 30 on May 16. 2014

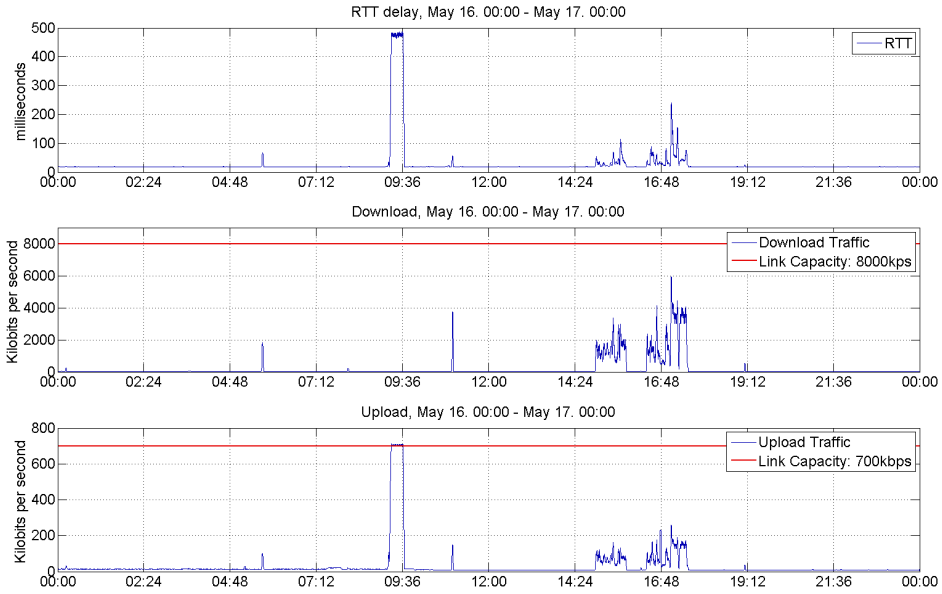


Figure 31: Smoothed RTT delay, download traffic, and upload traffic at Rindadalsvegen 30 on May 16. 2014

Figure 32 shows a short time period on May 16. during upload of a large file. The packet loss during this period is 0.6598%. The reliability calculated by (1) for a RTT latency requirement of 200 ms gives a reliability of 15.41%. However,a RTT latency requirement of 670 ms results in a reliability of 99.15%.

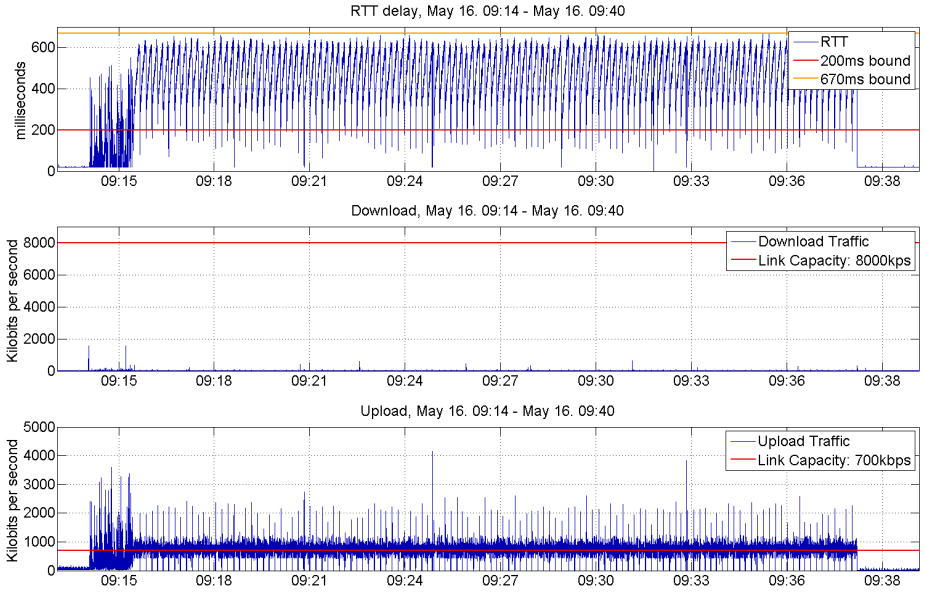


Figure 32: RTT latency, download traffic, and upload traffic during upload of large file at Rindadalsvegen 30, with RTT latency bounds.

Table 12 presents some statistical metrics of the measured RTT between 09:14 and 09:40 on May 16. Table 13 presents the reliability of different latency requirements during the same period.

Metric	**Value**
Pings sent	13 958
Pings dropped	58
Packet-Loss	0.4155%
Median of RTT	458 ms

Table 12: Calculated statistical metrics of the RTT at Rindadalsvegen 30 on May 16. between 09:14 and 09:40.

Requirement [ms]	**Reliability**
30	12.34%
200	**15.41%**
460	49.78%
670	**99.15%**

Table 13: Reliability for different RTT latency requirements at Rindadalsvegen 30 on May 16. between 09:14 and 09:40.

Figure 33 shows the cumulative distribution function for the measured RTT between 09:14 and 09:40 on May 16.

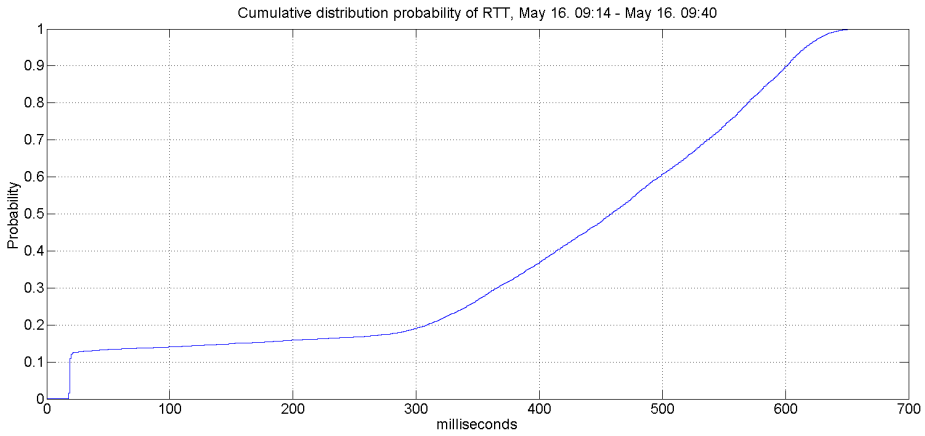


Figure 33: Cumulative distribution function of RTT at Rindadalsvegen 30 on May 16. between 09:14 and 09:40

Figure 34 shows a plot of the traffic and measured RTT on May 16. 09:17 - 09:24. A recorded packet loss is marked by a coloured bar. Take particular notice of how the dropped packet markers are positioned on a delay peak. Table 14 presents some statistical metrics of measured RTT between 09:17 and 09:24 on May 16.

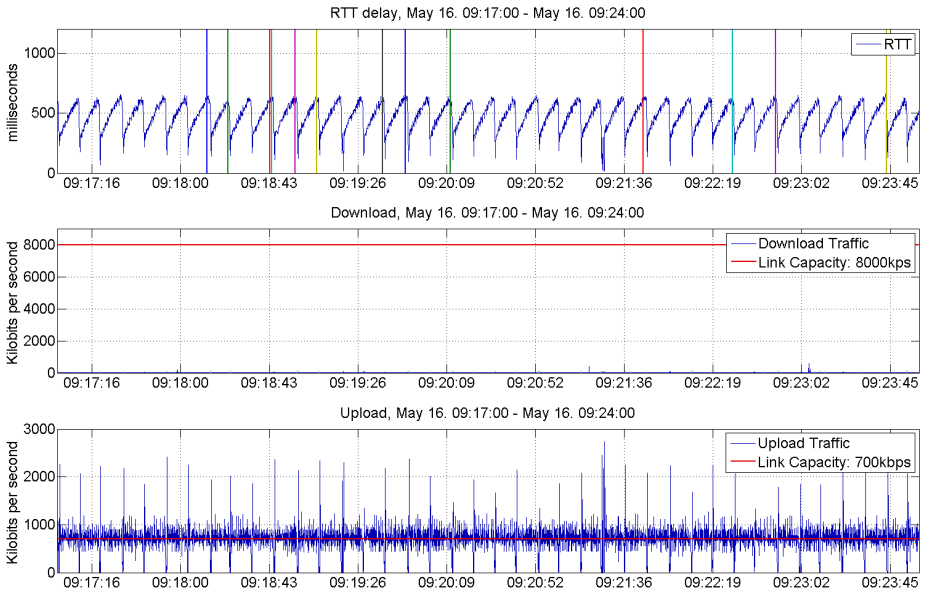


Figure 34: Delay and upload during upload of large file at Rindadalsvegen 30 on May 16, 09:17 - 09:24. where a packet-loss from the delay monitor is marked by a coloured bar.

Table 14: Calculated statistical metrics of the RTT at Rindadalsvegen 30 on May 16. between 09:17 and 09:24.

Metric	**Value**
Pings sent	3 779
Pings dropped	13
Packet-Loss	0.344%
Median of RTT	486ms
Mean of RTT	475ms

Figure 35 shows a plot of the traffic and measured RTT during the upload of a large file on May 16. between 09:20:20 and 09:20:36. Notice the sawtooth form on the delay plot and how the RTT latency rarely dips below 250 ms. The period between the colored bars $t1$ and $t2$, where both the upload rate and the RTT shows an almost flat curve should also be kept in mind.

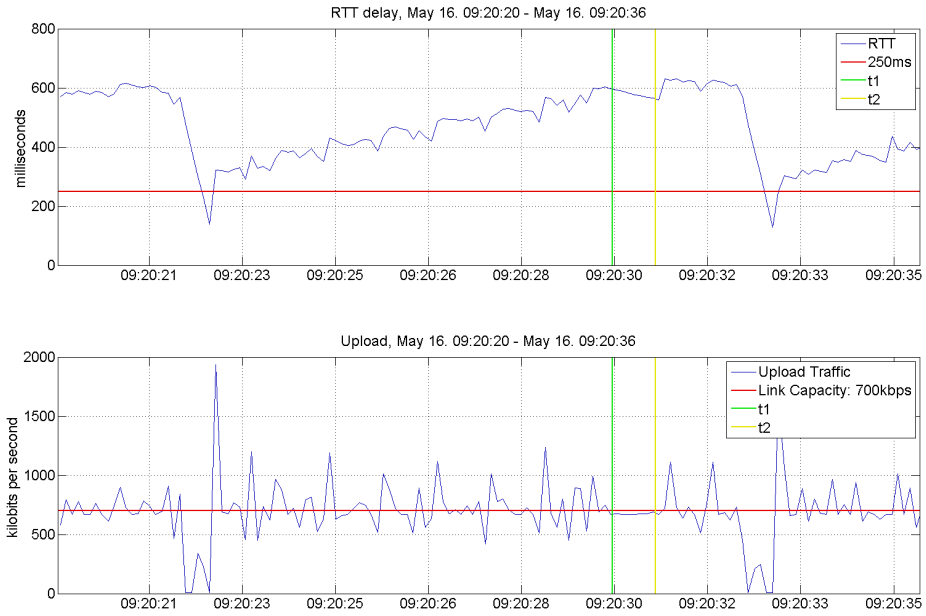


Figure 35: Close up of delay and upload during upload of a large file at Rindadalsvegen 30 on May 16. Shows TCP slow window increase before congestion, and fast decrease after congestion.

5.2.5 Application Behaviour

Figure 36 shows the RTT, download, and upload recorded during a download of a large file over File Transfer Protocol (FTP), when only one device is connected to the gateway to remove any cross-traffic sources.

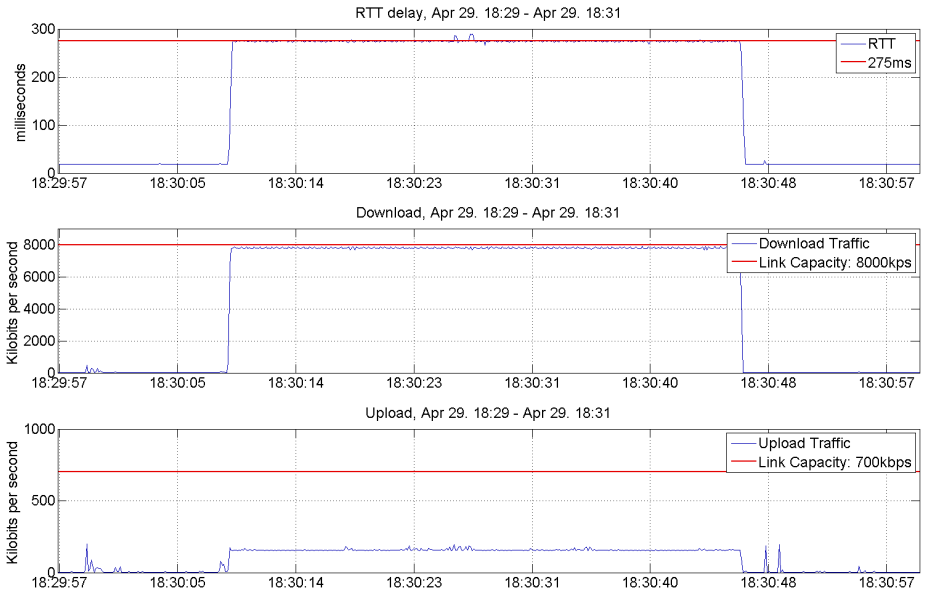


Figure 36: Undisturbed Download of a large file over FTP at Rindadalsvegen 30.

Figure 37 shows the RTT, download and upload recorded during an upload of a large file over FTP and with some HTTP web browsing and `skype` traffic at marked points. Notice the high delay at the start of HTTP traffic.

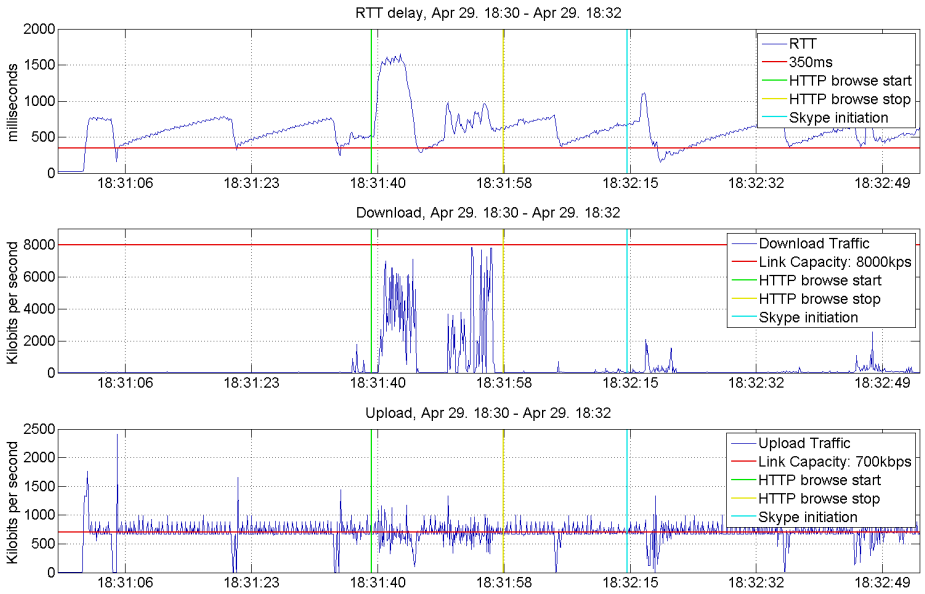


Figure 37: Upload of a large file over FTP with ingression of HTTP and Skype traffic at marked points at Rindadalsvegen 30.

5.2.6 Increase in Baseline Round Trip Time

Figure 38 shows an increase in the baseline RTT of about 100 ms measured from the gateway. This occurs several times in the data set.

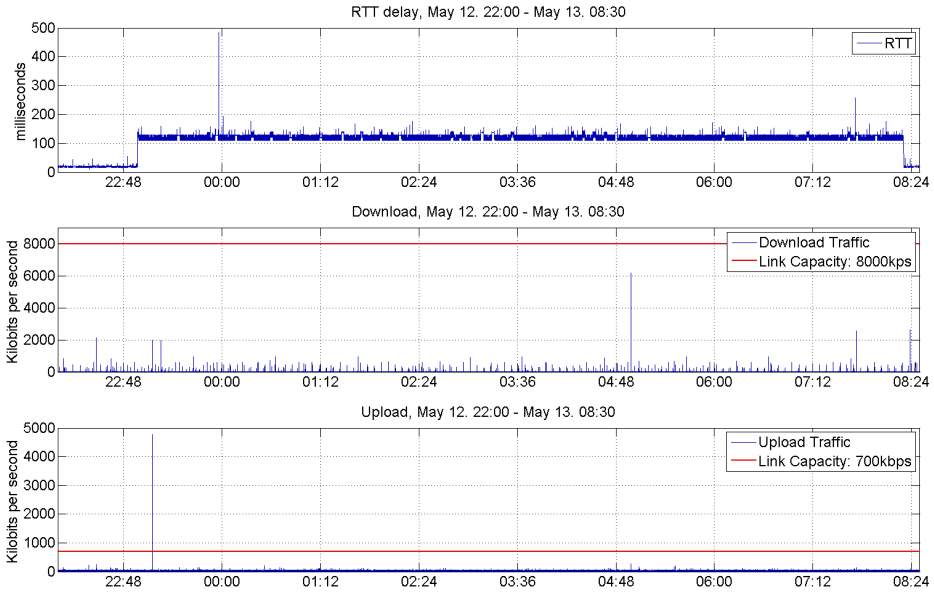


Figure 38: Increase in baseline RTT at Rindadalsvegen 30 measured by the gateway.

Figure 39 shows the RTT measured from a computer connected to the gateway at Rindadalsvegen during the same period as shown in Figure 38. This figure does not show the increase in baseline RTT.

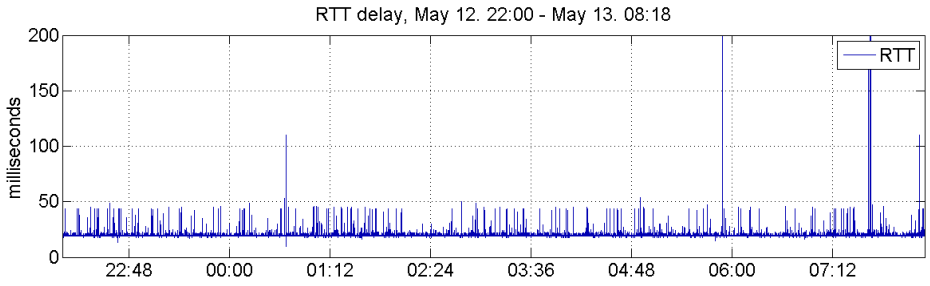


Figure 39: RTT at Rindadalsvegen 30 measured by a computer connected to the gateway during the RTT increase measured by the gateway.

5.2.7 Connection Drop

Figure 40 show one of two occurrences of very high delay and heavy packet-loss during the measurement period. Notice how the download measurements flat line during these high delay periods. `dstat` stopped running after 03:01:50 on May 19, and both the delay monitor and the traffic monitor stopped running at approximately 08:23 on May 19. Table 15 presents the sequence of packets transmitted during these periods. The linear slopes on the figures after connection drops are not measurements, they are only a line connecting the last sample before the connection drop with the first sample after the connection has come back up.

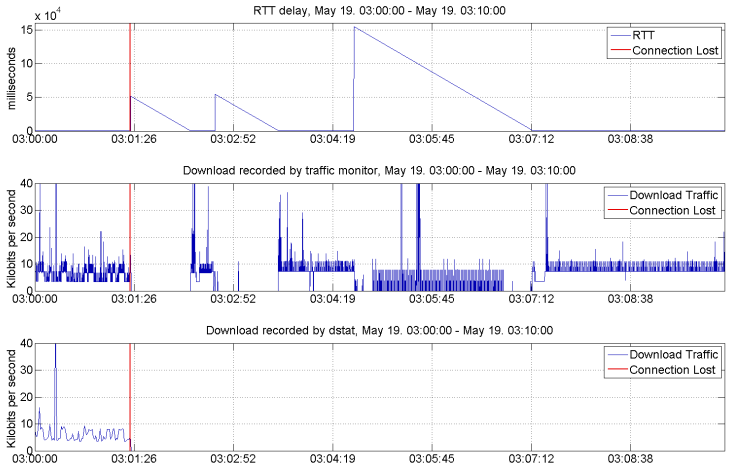


Figure 40: Delay, download measured by traffic monitor, and `dstat` at Rindadalsvegen 30 on May 19. from 03:00 to 03:10.

Table 15: Recorded packet sequence at Rindadalsvegen 30 on May 19 from 03:04:37 to 03:07:13.

Packet Number	Sent Time	Response Time	RTT [s]
2 632 814	03:04:37:791	03:04:37:811	0.019468
2 632 815	03:04:37:901	03:04:37:920	0.019210
2 634 221	03:04:37:901	**03:07:12:590**	154.689190
2 634 222	03:04:37:901	03:07:12:700	154.799308
2 634 223	03:04:37:901	03:07:12:812	154.910685
2 634 224	03:04:37:901	03:07:12:922	155.020823
2 634 225	03:04:37:901	03:07:13:030	155.128959
2 634 226	03:04:37:901	03:07:13:140	155.238852
2 634 227	**03:07:13:228**	03:07:13:252	0.023798

5.3 Tistelvegen 24

5.3.1 Entire Measuerment Period

The measurements at Tistelvegen 24 were collected from May 5. to May 20. There are gaps in the measurements, some because the measurement tool crashed, and some because the measurements were deemed faulty. The gaps are presented in Table 16. They have been taken out before calculating any metrics unless otherwise noted.

Table 16: Gaps in measurements of RTT at Tistelvegen 24.

From	To	Reason
May 7. 09:00	May 7. 11:00	Increase in baseline RTT. See section 5.3.3 and 6.1.1
May 8. 00:00	May 9. 11:00	Increase in baseline RTT. See section 5.3.3 and 6.1.1
May 11. 09:30	May 11. 12:00	Increase in baseline RTT. See section 5.3.3 and 6.1.1
May 12. 21:00	May 12. 22:30	Increase in baseline RTT. See section 5.3.3 and 6.1.1
May 15. 03:30	May 15. 08.30	Increase in baseline RTT. See section 5.3.3 and 6.1.1
May 17. 17:30	May 17. 18:30	Increase in baseline RTT. See section 5.3.3 and 6.1.1
May 18. 00:00	May 18. 02:30	Increase in baseline RTT. See section 5.3.3 and 6.1.1
May 18. 09:00	May 18. 10:30	Increase in baseline RTT. See section 5.3.3 and 6.1.1

Table 17 presents some statistical metrics of the measured RTT from May 5. to May 20. Table 18 presents the calculated latency reliability for the same period.

Metric	Value
Packets sent	10 625 065
Recorded packets dropped	1361
Recorded packet-loss	0.0128%
Median of RTT	1 ms
SD of RTT	380
variance of RTT	144 530

Table 17: Calculated statistical metrics of the RTT at Tistelvegen 24 May 5. to May 20.

Requirement [ms]	Reliability
30	96.09%
100	96.69%
200	97.37%
500	98.63%
1000	99.69%
1500	99.92%

Table 18: Reliability for different RTT latency requirements at Tistelvegen 24 May 5. to May 20..

Figure 41 shows the measured RTT, download traffic, and upload traffic May 5. to May 20.

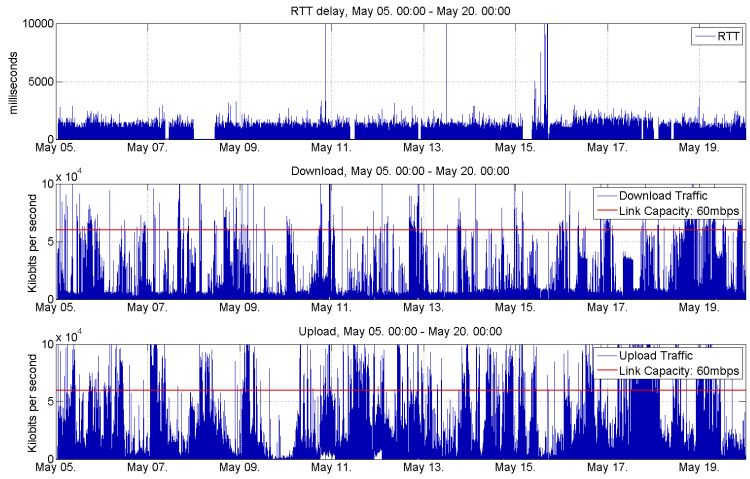


Figure 41: RTT delay, download traffic, and upload traffic at Tistelvegen 24 May 5. to May 20.

Figure 42 shows the cumulative distribution function of the measured RTT delay May 5. to May 20.

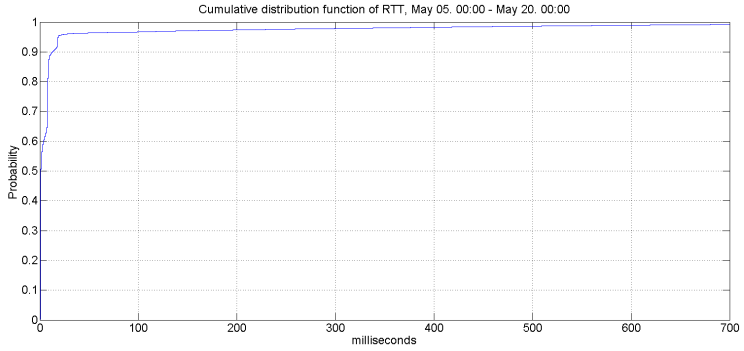


Figure 42: Cumulative distribution function of RTT delay at Tistelvegen 24 May 5. to May 20.

5.3.2 Daily Traffic

Figure 43 shows a bar graph of the total amount of traffic in GB both uplink and downlink at Tistelvegen 24.

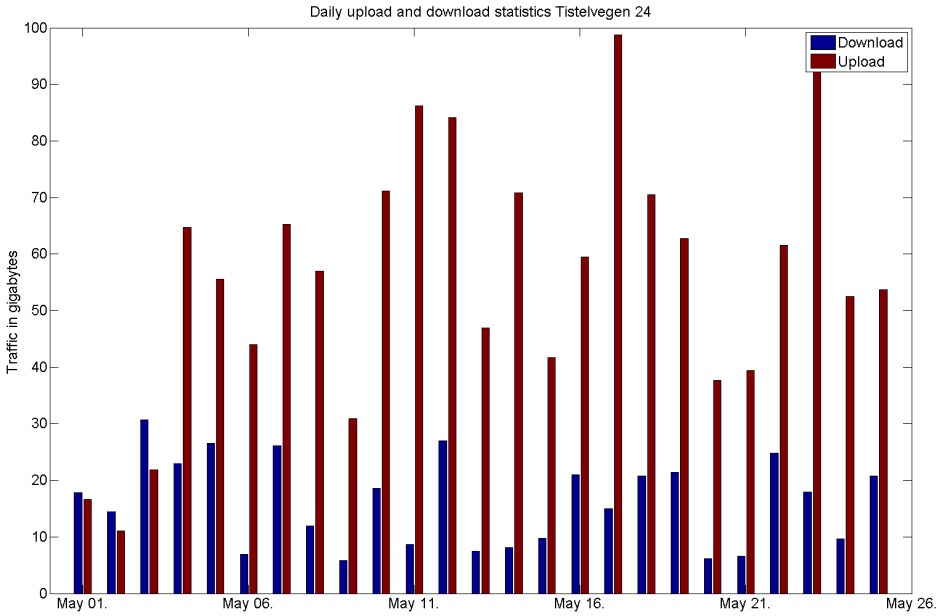


Figure 43: Daily download and upload statistics in gigabytes at Tistelvegen 24

5.3.3 Increase in Baseline Round Trip Time

Figure 44 shows an increase in the Baseline of the measured RTT. Notice how the increase is in increments of about 100 ms.

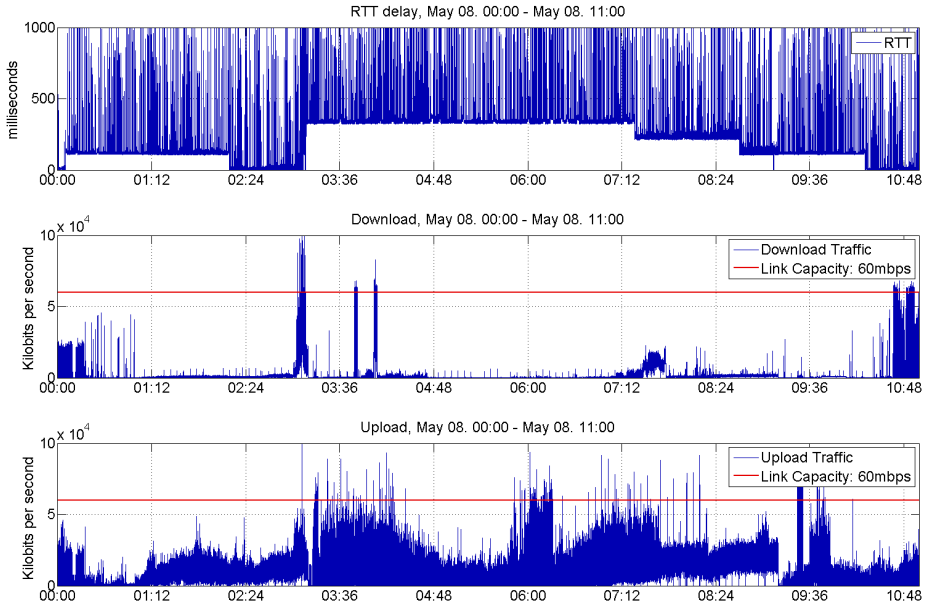


Figure 44: Increase in baseline RTT at Tistelvegen 24 on May 8. between 00:00 and 11:00

5.3.4 Measurements from May 16.

Figure 45 show the RTT delay and the traffic over the gateway throughout May 16. Table 19 lists statistical metrics of the RTT this day, and Table 20 presents the reliability for different RTT latency requirements.

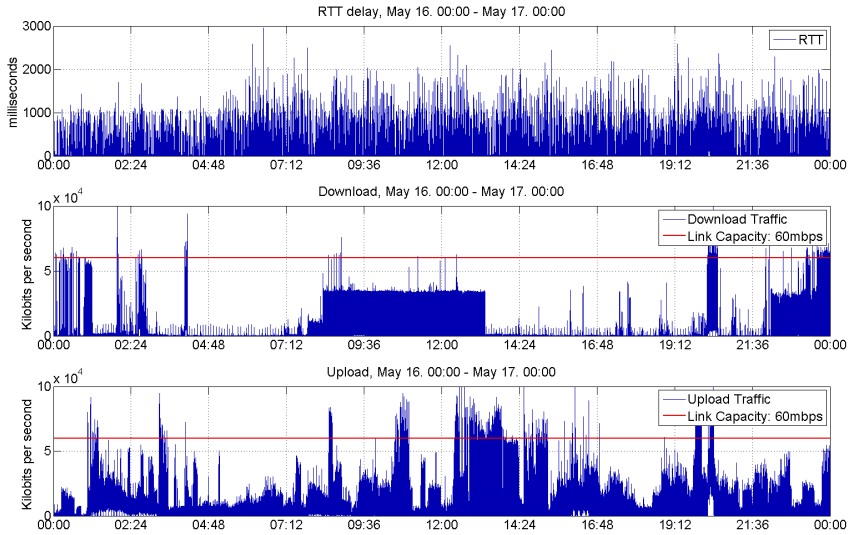


Figure 45: RTT delay, download traffic, and upload traffic at Tistelvegen 24 on May 16. 2014

Metric	Value
Pings sent	770 393
Pings dropped	51
Packet-Loss	0.007%
Median of RTT	2 ms
SD of RTT	126
variance of RTT	15 965

Table 19: Calculated statistical metrics of the RTT at Tistelvegen 24 throughout May 16.

Requirement [ms]	Reliability
30	95.14%
100	95.75%
200	96.75%
500	98.26%
1200	99.75%
1600	99.92%

Table 20: Reliability for different RTT latency requirements on May 16.

Figure 46 shows the cumulative distribution function of the measured RTT at Tistelvegen 24 on May 16. Figure 47 shows the same data as Figure 45 only smoothed.

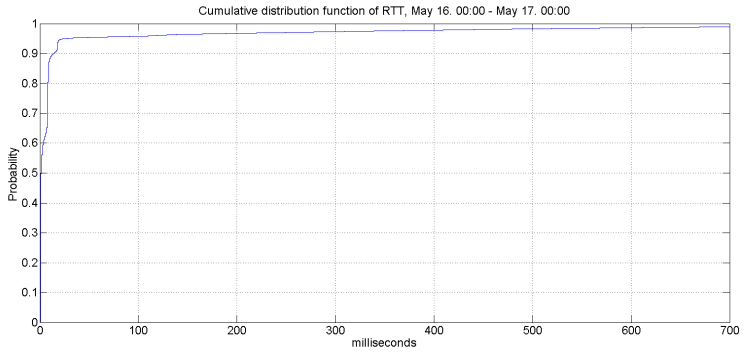


Figure 46: Cumulative distribution function of RTT at Tistelvegen 24 on May 16. 2014

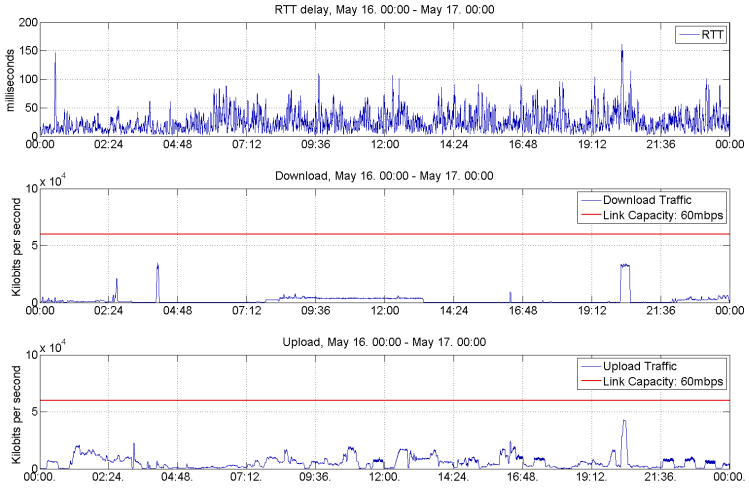


Figure 47: Smoothed RTT delay, download traffic, and upload traffic at Tistelvegen 24 on May 16. 2014

Figures 48 and 49 show a short period of measurements from 03:40 to 05:30 on May 16. It is very difficult to see in Figure 48 though a smoothed plot of the data, Figure 49, shows some distinct peaks at the same time in both upload and delay.

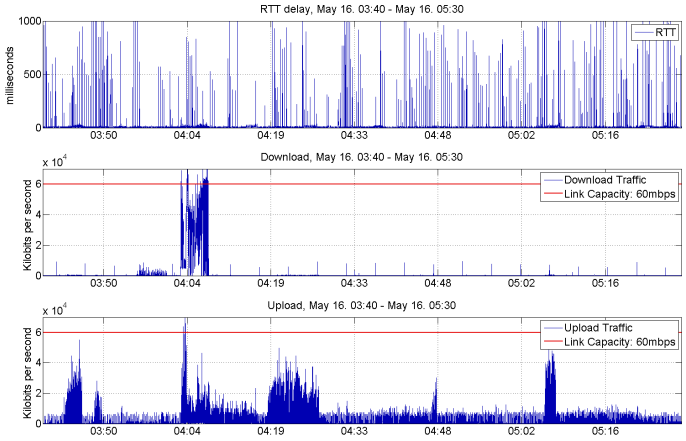


Figure 48: RTT and traffic at Tistelvegen 24 on May 16, 03:40 - 05:30. 2014

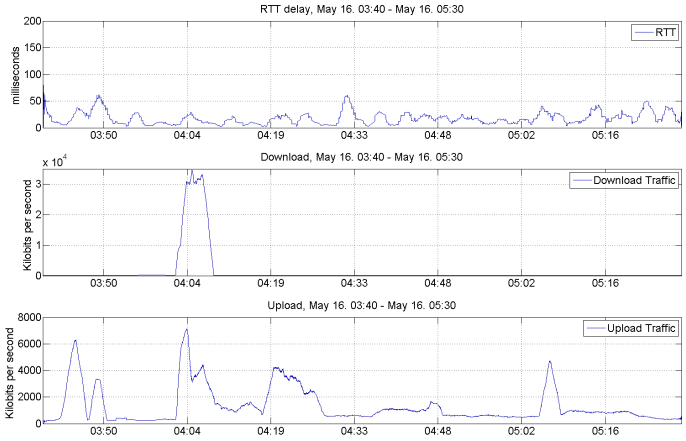


Figure 49: Smoothed RTT and traffic at Tistelvegen 24 on May 16, 03:40 - 05:30. 2014

6 Discussion

This discussion chapter will first present a discussion on the measurement results found in the characterization experiments, then a discussion on the mitigations of congestion and bufferbloat follows, and finally an analysis on guaranteed timely deliver at end-users ending in recommendations to minimizing latencies at end-users based on both results and the literature study.

6.1 Characterization of Rindadalsvegen 30 and Tistelvegen 24

The aim of the characterization study was to find presence of latencies due to congestion, bufferbloat and problems with AQM. First some odd observations in the data set will be presented and discussed, then the presence of congestion, bufferbloat, and AQM will be discussed. At the end the latency characteristics and the latency reliability of the links are discussed.

6.1.1 Odd Observations in Data Set

Connection Outages at Rindadalsvegen 30

Figure 40 shows a possible connection outage at Rindadalsvegen 30. In Figure 40 the download traffic recorded by both the traffic monitor and the `dstat` tool drops to zero at the start of the first period of high delay. The `dstat` tool stopped working after this drop in measured traffic. The tools are as mentioned in section 4.6 run using a `ssh` session over the same link being characterised, and the fact that `dstat` stopped working a moment after the first flat line of download traffic, suggest that in fact the connection was down for a short period of time.

Both the traffic monitor and the delay monitor kept on running though. `dstat` shows continuous output in the `ssh` session, while both the traffic monitor and the delay monitor do not. They only write their output to file. This might be the cause of their continuous recording of data.

Table 15 shows the sequence of packets that was sent from the delay monitor during the high delay period. Table 15 shows an increase in successful packets from packet 2 632 815 to 2 634 221, i.e. there were 1 406 dropped packets. 1 406 packets at approximately 100 ms intervals is approximately 140 s of delay, which is similar to the delay recorded by the delay monitor, i.e. 154 s.

However, the recorded sent time for the first packet that is successfully sent and received after the connection drop, shows that it was sent at the same time as the last successful packet. This suggest that there is a bug in the delay monitor. This is however, only shown at these connection drops.

The data showing this behaviour has therefore been excerpt from the data set used in calculations. This data is not necessary to conclude on bufferbloat, and AQM.

Increase in Baseline Round Trip Time

Figures 38 and 39 show RTT measurements during a period of time in the data set where the baseline RTT recorded by the delay monitor at the gateway is increased by around 100 ms. Figure 39 shows the RTT measured by a computer connected to the gateway and does not show the same increase though, which suggest that this is a bug in the delay monitor.

Figure 44 shows the same increase in RTT baseline as Figure 38. However, this figure provides more insight into this odd data. Not only has the baseline increased by approximately 100 ms but by multiples of approximately 100 ms. This further indicates a bug in the delay monitor.

The data showing this behaviour has therefore been excerpt from the data set used in calculations. This data is not necessary to conclude on bufferbloat. But if this data originates from a weird form of AQM, however unlikely, this might lead to the wrong conclusions.

6.1.2 Congestion

As stated in section 3.1.1 congestion occurs when the arriving traffic at a link is higher than the capacity of the link. Packet-loss due to full buffers might result. Congestion will always be an issue at points in a network where the traffic of high capacity links is concentrated into a lower capacity link.

As long as the last mile links are of a lower capacity than the links on the inside of the network and traffic is not shaped to the bottleneck rate of the last mile link, congestion is going to be a persistent issue.

Furthermore congestion must occur before bufferbloat is experienced.

Rindadalsvegen 30

Rindadalsvegen has an uplink rate of $700\,\mathrm{kbit\,s^{-1}}$, which entails that the $1\,\mathrm{Gbit\,s^{-1}}$ network on the local network should have no problem introducing congestion.

Figure 34 shows a saw tooth shaped RTT originating from one TCP flow over the gateway. The flow is increasing its congestion window to probe for more bandwidth filling the buffers and increasing the delay, before the buffer is full and packet-loss occurs. The TCP flow then throttles back its congestion window before it starts increasing it again. This is the behaviour expected of TCP when experiencing congestion.

Figure 35 shows how the arriving upload traffic is higher than the capacity of the link. The simple fact that the delay is increasing is proof of congestion occurring, because it means that the buffer is filling up. Figure 26 shows that the upload traffic passing the gateway is at some points over $3000\,\mathrm{kbit\,s^{-1}}$, 4 times the rate of the uplink. This further incriminate occurrences of congestion.

Tistelvegen 24

Tistelvegen has both uplink and downlink rates of $60\,\mathrm{Mbit\,s^{-1}}$, this is considerably more than the link rates at Rindadalsvegen though still below the local network's rate of $1\,\mathrm{Gbit\,s^{-1}}$.

Figure 41 shows that the upload traffic passing the gateway at some points is greater than the uplink rate. A concise correlation between congestion and delay could not be found though, as it was in the case of Rindadalsvegen. Saturating the link over time proved difficult.

6.1.3 Bufferbloat

Bufferbloat is the existence of buffers that are so large that they increase the latency of a link without increasing utilization. Gettys [19] show bufferbloat by monitoring the RTT while saturating the link. Sundaresan et al. [37] show bufferbloat using the same method. Furthermore Appenzeller et al. [6] argue that there is no need for additional buffering as long as the link is 100% utilized.

Rindadalsvegen 30

Figure 31 shows that the RTT measured by the delay monitor increases drastically at approximately 09:36 on May 16, at the same time as the uplink is saturated. A measurement of the RTT to the first Internet IP hop during a similar period where the uplink was saturated showed that the increase in delay originates at the last mile link, see Table 8.

Figure 32 shows the spike in more detail. The destination of the traffic is a server located at Tistelvegen 24. The RTT for the traffic and the delay monitor should therefore be about the same. The median RTT recorded during this period was around 450 ms, Table 12. The median RTT over the course of the day is 19 ms, Table 10. 430 ms of queuing results in a inferred buffer size of 301 kbit at a rate of 700 kbit s^{-1}. The BDP for the baseline RTT is 13.3 kbit. As showed by Appenzeller et al. [6] the BDP is the maximum amount of buffering needed for TCP to have 100% utilization of a link. This suggests that the buffer is approximately 290 kbit too large to handle one TCP flow, i.e. about 22 times too large.

Figure 35 shows that the delay almost never drops below 250 ms. Appenzeller et al. [6] argue that the perfect buffer would be exactly empty in time for the first packet from a TCP flow to arrive after a congestion avoidance period. Both figures 32 and 34 show the same behaviour with a persistently high delay, most likely originating from persistently full buffers.

The almost flat curve between the lines $t1$ and $t2$ in Figure 35 shows that when the upload traffic is stable and almost at the capacity rate of the link the delay slowly decreases. This suggest that if the traffic arriving at the link arrives at exactly the capacity of the link the delay should be constant. Suggesting that it is possible to keep the latency at around 19 ms, by shaping the traffic accordingly.

Figure 36 shows that the RTT increases as the downlink is saturated. This figure closely resembles what Sundaresan et al. [37] show in Figure 21, and what they interpret and conclude is latency from bufferbloat.

Figure 37 shows a different aspect of the bufferbloat problem. As soon as a link at a website is clicked, and the web browser opens a HTTP session pulling the website information, the RTT spikes to 1500 ms. This is a period where both the uplink and downlink is saturated.

Concluding, Figure 36 shows approximately 250 ms of excess delay in the downlink, and Figure 37 shows at least 350 ms of excess delay in the uplink.

Tistelvegen 24

Saturating the link proved difficult, and therefore evidence of bufferbloat in this link is scarce. Although Figure 49 shows some distinct peaks at the same time in both upload and delay, the figure show peaks in delay not corresponding to a peak in traffic, and vice versa.

If this irregularity is because of cross-traffic induced by the other household or because of AQM is difficult to determine. Therefore it is difficult to conclude on the presence of bufferbloat.

6.1.4 Active Queue Management

The objective of AQM has since the beginning of the development of RED been to reduce latencies by actively managing queuing so that queuing delay is reduced to a minimum. AQM can use two different approaches to achieve its goals, dropping certain packets, or notify packets or flows of congestion.

The presence of AQM is difficult to determine in general. Sundaresan et al. [37] suggested that what they found in Figure 22b might be caused by AQM.

Rindadalsvegen 30

Figure 34 shows how packet-loss is conistently recorded at peak delay. This suggest that the packet-loss is originating from buffer overflow rather than being dropped for other reasons. This figure shows typical packet-loss placement of drop-tail buffers, i.e. buffers that drop packets when full.

However,the fact that ping experiences lower latency than the delay monitor and `traceroute` suggests that the traffic is managed in some way, see section 5.2.3. The modem might incorporate some sort of differentiated services, strictly speaking though, this is not AQM.

Tistelvegen 24

As mentioned in the discussion whether the link at Tistelvegen suffers from bufferbloat there are some peaks at the same time in both traffic and delay. However, there are peaks not corresponding to each other. This might implicate that there is some kind of AQM implemented at the modem of this link.

It might be possible that the link at Tistelvegen 24 has large buffers and AQM in place to minimise the impact of large buffers. This could explain the measurements at Tistelvegen 24.

Looking at Figure 48 it might seem as though the peaks that might relate to each other in Figure 49 are at intervals where the traffic has high variation. Further implicating an implemented AQM scheme that cant quite keep up with the varying traffic.

But as mentioned earlier the cross-traffic from the other household might induce these differences. The fact that there is another gateway on the path before the first internet hop might make the measurements at this gateway more noisy. The evidence of AQM schemes at Tistelvegen are therefore inconclusive.

6.1.5 Characteristics at the Sites

Daily Traffic at the Sites

Figures 28 and 43 show how different the amount of traffic between these two sites are. This might correspond to the different demographics at the two sites.

The fact that the achievable bandwidth at both locations is very different, might explain the differences. Figures 26 and 41 show that the traffic at Tistelvegen is more frequent than at Rindadalsvegen.

Packet Loss, Standard Deviation, and Variance in Measurements

Tables 6 and 17 show statistical metrics for the entire measurement period of the respective sites. The recorded packet-loss of both sites is quite low, 0.0145% at Rindadalsvegen and 0.0128% at Tistelvegen.

The same tables show that the median of the measured RTT is 19 ms at Rindadalsvegen and 1 ms at Tistelvegen, this shows a great difference in the two communication technologies. The baseline RTT is much lower for the fiber to the home connection. The geographical distance to the office at Gløshaugen is different though. Tistelvegen is much closer to the office than Rindadalsvegen. A ping to the NIX in Oslo, a geographical location similar in distance to both sites, showed 25 ms from Rindadalsvegen and 9 ms from Tistelvegen, suggesting that there is more latency induced by the ADSL technology or the topology chosen by Telenor.

The standard deviation, Standard Deviation (SD), of the measured RTT at Rindadalsvegen is 72 and the variance is 5 253. At Tistelvegen the standard deviation, SD, is 380, and the variance is 144 530. The variance of the latency is very different between the sites. This does not have to be caused by the difference in technology. The very different traffic at the two sites could account for the high variance of the latency.

Distribution of Delay During Saturation at Rindadalsvegen

The cumulative function of the measured delay during a saturation of the uplink shown in Figure 33 closely resembles the cumulative distribution function of a normal distribution. This backs up the claim made by Appenzeller et al. [6], the filling grade of a buffer follows a normal distribution during a saturation by a TCP flow.

Reliability

The data removed from the data set would have impacted the reliability calculations in a negative manner. The data removed because of the increase in baseline has been argued as a bug in the delay monitor, i.e. the measurement would have looked differently if the delay monitor had been implemented correctly. However, the data removed because of the connections drops would obviously have impacted the reliability because during these periods transmission of data would not be possible. Had the delay monitor been implemented differently, connection drops could have been accounted for and an

availability analyses performed. The fact that this data is removed must be kept in mind while reading this section.

From a reliability perspective the link at Rindadalsvegen shows better perfomance than the other link, when the entire measurement period is taken into account. A requirement of 700 ms in RTT shows three nines, 99.94%, reliability at Rindadalsvegen while a requirement of 1500 ms shows the same reliability at Tistelvegen, tables 7 and 18. Even a requirement of 30 ms shows better reliability at Rindadalsvegen than at Tistelvegen.

The traffic at both sites is still very different though and this affects the RTT measurements. However, the link at Rindadalsvegen still shows 99.15 % reliability for a latency requirement of 670 ms during a period with much traffic, see Figure 32 and Table 13. This is is still better than the 98.26% reliability for a requirement of 500 ms for the entire measurement period at Tistelvegen.

6.1.6 Bufferbloat Effect on Real-Time Applications

Figure 32 shows how the RTT latency almost never goes below approximately 200 ms when the upload bandwidth at Rindadalsvegen 30 is fully utilized by a TCP connection. This clearly affects the possibility of utilising this link for real-time applications. The latency reliability for a 200 ms requirement is 15.41% during this period, Table 12. As mentioned earlier the source of this high latency is bufferbloat.

The latency reliability for a 200 ms requirement calculated over the entire day that this subset of data originates from, May 16. is 99.98%. If the reliability of this connection was considered only from the measurements of the entire day this link would satisfy this requirement. But clearly during the period shown in Figure 32 the link would not satisfy the same requirement.

This suggest that a latency reliability term based solely on statistical measurements is not enough to conclude if a link could handle real-time applications and their timing requirements or not. E.g. if a control loop demanding an update of the state of the system every 200 ms, the two calculations would yield different conclusions.

Figure 33 shows the cumulative distribution function of the delay over the same period showed in Figure 32. This figure shows an entirely different distribution than that of the cumulative distribution function of the entire measurement period shown in Figure 27.

Either the latency reliability term must be calculated during full link utilization for links suffering of bufferbloat or a different term must be used to conclude on a links ability to meet timing requirements.

6.2 Coping With Congestion and Bufferbloat

Latencies due to bufferbloat is induced by congestion. To cope with congestion, there are three mitigations that have been looked at:

1. Correct buffer sizing
2. Adopting AQM
3. Tuning TCP

These three mitigations will be discussed accordingly in the following three sections, with the end-user's last mile link in mind.

6.2.1 Correct Buffer Sizing

The size of buffers has been the object of many discussion throughout history, with the BDP as the more frequent adopted strategy. In the case of only one TCP flow open at any time at an end-user the correct buffer size would be the BDP to fully utilize the link [6].

However, since ISPs often only provide one modem to fit different bandwidth service plans [37], and baseline RTT might change [19], this is an unsolvable calculation. Unless the buffers in modems can be configured to the correct buffer size for the bandwidth. This would require configuring of each modem to work perfectly, and entails extra work for the ISPs.

Section 3.2.2 presents Appenzeller et al. [6] arguments to reduce buffer sizes of Internet core routers. They argue that when several TCP flows are multiplexed together the buffer size should be given by (13). This might be valid in the Internet core. The last mile links at end-user might not always multiplex enough flows to avoid synchronization between flows though. If synchronization occurs, the flows can be treated as one flow. The number of flows necessary to avoid synchronization effects is around 500 [6].

Beheshti et al. [7] argue that buffers can be sized at a logarithmic scale of the rate of a link, as long as the traffic is smooth. Figure 36 shows the arrival of very smooth traffic at Rindadalsvegen. This however might not allways be the case, and is therefore possibly not the correct solution for an end-user link.

(12) and (13) are equivalent. Tuning buffer sizes using (13) is always going to be reliant on estimates of average RTT. Therefore these approaches are not going to give 100% utilization for all RTTs. Furthermore, RTTs can change [17, 19].

Since the maximum queueing delay D_{qmax} is given by $D_{qmax} = B/C$ for a given buffer of size B and a given link with rate C, the maximum queueing delay imposed by (13) can be given by (22), where $\bar{l}_{rtt}$ is the RTT used in the buffer size calculation and n is the number of flows. I.e the maximum delay that can be imposed by a buffer is the same as the RTT that it is designed to cope with.

$$D_{qmax} = \frac{\bar{l}_{rtt}}{\sqrt{n}} \tag{22}$$

(22) shows that buffer sizing can have a huge impact on the queuing delay, especially when as Gettys [19] state, the $\bar{l}_{rtt}$ used in buffer sizing has been a continental 100 ms delay.

This thesis suggests a selection of buffer size based on maximum queue delay could be a better approach than designing it based on 100% utilization for a maximum RTT, which is what is done using (13) and (7). Calculating the buffer size, B, by maximum queue delay, D_{qmax}, is done by (23), where C is link rate.

$$B_{maxdel} = C \cdot D_{qmax} \tag{23}$$

6.2.2 Active Queue Management

While static approaches like tuning the buffer size in a modem to fit the latency, traffic and bandwidth of a link, it can not cope with changes in either of these. Using static tuning of buffers is also bad for bursty traffic arriving on top of long lived TCP flows, since TCP most likely has filled the buffers almost to capacity. Gettys [19] proclaims that there can not be a correct static buffer size in any device based on this. Furthermore, Pan et al. [33] argue that reduced buffers is not what is really wanted because then bursty traffic is handled poorly, it is reduced latencies, therefore they focus on AQM.

AQM schemes like PIE and CoDel control the effective queueing delay. I.e. the reference latency in PIE is the desired latency imposed by the queue.

This is arguably no different than a dynamic buffer sizing scheme though with a buffer size B_{dyn} given by (24), where C is the throughput of the link, and D_{ref} is the reference queue delay, since the propagation delay of the queue is controlled to the reference delay.

$$B_{dyn} = D_{ref} \cdot C \tag{24}$$

Although, in contrast to a dynamic queue sizing scheme, bursty traffic is allowable by PIE. Where a dynamically sized buffer would have a constant buffer size for a given bandwidth, and an instant burst of packets into a full buffer would result in this traffic being dropped, a scheme like PIE would allow the initial burst, because it reacts only after the queue delay is above the reference. However, if the bursts continues and induces persistently full buffers, a scheme like PIE would react accordingly and drop more of the packets.

Nichols and Jacobson [31] present five terms that a good AQM scheme should fulfil. PIE arguably fulfil all of the terms. Although the first term is not entirely fulfilled. The latency reference would have to be set, and an analysis of the effect of the reference should be performed.

6.2.3 TCP tuning

Even if buffers are large enough and AQM is in place to reduce bad queueing delay, they will not ease congestion unless the protocols themselves react accordingly to a notification of congestion. The reaction must be swift and ease the traffic to avoid too much packet-loss induced by congestion.

6.2.4 Packet-Loss of Active Queue Management

AQM schemes like PIE and CoDel drop packets to notify of congestion. An experiment performed by Nichols and Jacobson [31] showed that CoDel has less or equal packet-loss than drop-tail. The packet-loss rates of such schemes might be shown to be less or equal to that of a simple drop-tail scheme, where the size of the buffer in the drop-tail scheme is corresponding to a buffer size imposing the same max latency as the reference in the AQM.

The following argumens support this. More burst are allowed by AQM schemes like PIE and CoDel than by drop-tail buffers at a certain level of congestion. Also a packet-loss from an AQM scheme would entail the same drop in congestion as a drop-tail scheme.

6.3 Guarantee of Timely Delivery at End-User

This section will try to provide a new way of analysing a given Internet path in terms of timely delivery to try to guarantee within a measure that the path satisfies a given latency requirement. The reason for normally using deterministic networks like industrial networks when deploying real-time applications like sensory systems, is that the delivery can be guaranteed, 100%.

This can not be done for a path on the Internet, but a probability that it will be delivered within requirements can still be calculated, assuming that the max latency, l_{max}, on a given path through the internet is given by:

$$l_{max} = \sum_{i=1}^{n} l_{pi} + l_{ti} + l_{qi} \quad l_{qi} = \frac{B_i}{C_i} \tag{25}$$

, where l_{pi} is the processing at the ith node, l_{ti} is the transmission latency on the ith link from the ith node to node $i+1$, and l_{qi} is the maximum queueing delay at the ith node. If there is maximum congestion and the buffer is full at node i, a packet-loss occurs. If there is only room for one more packet, this packet is subject to a delay of B_i/C_i, where B_i is buffer size at the ith node and C_i is the draining rate of the ith link. Therefore arguably the maximum queueing delay at a node is the buffer size B divided by the draining rate, i.e. the link speed C.

Disregarding the availability factor, if the packet-loss rate is known, the probability, $P_{delivery}$ of a timely delivery for a one shot packet within the maximum latency is:

$$P_{delivery1} = 1 - P_{loss}$$

, where P_{loss} is the probability of a packet loss. The probability of timely delivery $P_{delivery}$, within the latency requirement l_{req}, on n tries on a path with maximum latency l_{max} is given by:

$$P_{delivery}(l_{req}) = \sum_{i=0}^{n} P_{loss}^{i}(1 - P_{loss}) \quad n = \left\lfloor \frac{l_{req}}{l_{max}} \right\rfloor + 1 \tag{26}$$

, where P_{loss} is the probability of a packet loss. (25) and (26) then together describe the problem that need to be solved to specify whether this path on the Internet can be used to deploy a real-time application with a given latency requirement. This however, is a daunting task. It entails calculating all the latencies that make up the path. It can be argued though that the processing at a node is constant for all loads of traffic, and so is the transmission latency.

Assuming then that except for the last mile-links the path is non congested, the contributions from the queueing delay is constant at all other links and nodes. This entails that a measurement of the latency from end-to-end is valid for all contributions to the latency except the queueing delay at the last mile links. A last mile link only consists of one node and one link for a one way link, a manageable task. The calculations of (25) can then be reduced to:

$$l_{max} = l_{meas} + \sum_{i=1}^{2} l_{pi} + l_{ti} + l_{qi} \quad l_{qi} = \frac{B_i}{C_i} \tag{27}$$

, where l_{meas} is the measured end-to-end latency, l_{pi} is the processing at the ith node before the ith last-mile link, l_{ti} is the transmission latency on the ith last-mile link, and l_{qi} is the maximum queueing delay at the ith node. The last mile links are placed right after the sender and right before the receiver of an end-to-end path.

Both Sundaresan et al. [37] and Maier et al. [29] argue that the last-links are where the majority of congestion occurs. Results from the measurements at Rindadalsvegen 30 in Chapter 5 show that when there is little to none traffic on the link the variation in the measured latency is little to none. Suggesting that the majority of the variation in the RTT of a path is induced at the last-mile links.

The next section will focus on how to keep the latency in the last-mile link down.

6.3.1 Minimizing Latency in End-User's Last Mile Link

There is a difference between reducing the max latency at the last-mile link and reducing the average latency. AQM schemes like PIE will help to keep the queue delays lower than the maximum queueing delay of drop-tail buffers. Although PIE does come very close to achieving below twice the reference delay for over 90% of packets during heavy load, 20% oversubscription to bandwidth, 90% is not 100%. It is still difficult to guarantee that the queue delay is not larger than the reference delay.

Differentiated services allows more control. By queueing different traffic in different buffers, and prioritizing one buffer over another, the packet-loss rate and the queueing delay for one type of traffic can be minimized. As shown by the `traceroute` vs. ping measurement during saturation at Rindadalsvegen, see section 5.2.3, the ICMP pings showed considerably lower latencies than the `traceroute` measurements.

Differentiated services must be supported in some way though, and the ingress of support for differentiated services approaches has not been looked at in this thesis. However, if congestion occurs only at the last mile link. Deploying differentiated services at the last mile link, thereby reducing the max latency imposed by the last mile link, might reduce max experienced latency.

To minimize the latencies in the end-user's link one of two things can be done:

1. The buffers can be tuned to achieve a given max queue delay, and thereby a given max latency, possibly sacrificing some bandwidth utilization.

2. Buffers can be chosen large enough to ensure bursts are handled and an AQM scheme can be put in place to keep the queue delay low.

The placement of buffers and AQM are shown in Figure 50. Solution one would entail no drop probability from AQM in Figure 50.

Furthermore to increase determinism of packets relating to the real-time application a prioritization of traffic similar to differentiated services could be implemented, traffic with time constraints is prioritized over other traffic and arrives at different buffers.

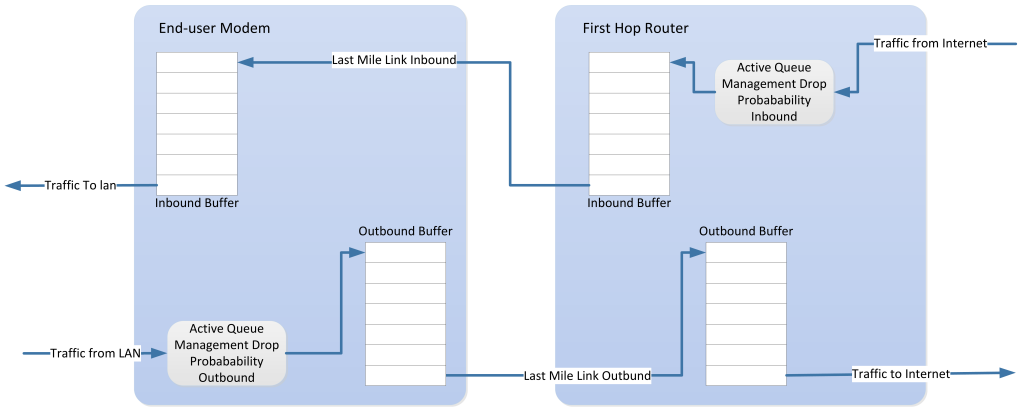


Figure 50: Placement of buffers and AQM at modem and first hop router.

There are two approaches to implement the aforementioned solutions at the end-user:

1. Cooperate with the ISP and implement either solution one or two in the modem at the end-user and the first hop router at the ISP as shown in Figure 50, for a similar approach to what Lyse Energi might do with their fiber to the home links.

2. Install a gateway incorporating solution two at the end-user as in figures 51 and 52.

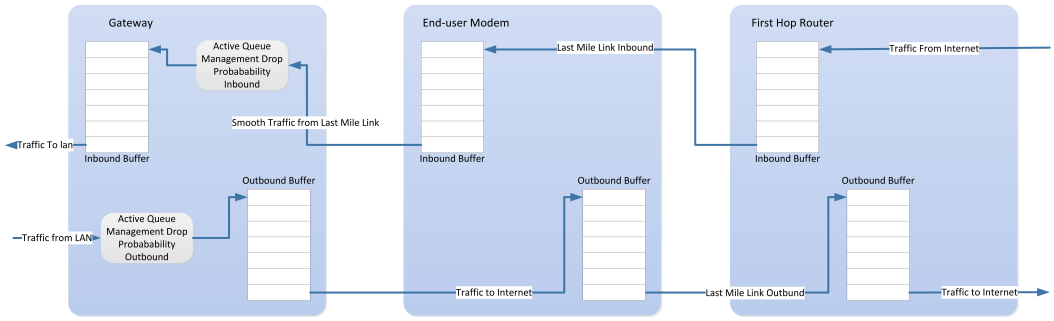


Figure 51: Placement of buffers and AQM in gateway.

Approach one is probably the most robust solution and able to deliver the best performance. Why it might be more robust becomes apparent when approach two is described. It does involve cooperation with the ISP though, and this would probably entail economic compenzation to the ISP. This approach involves changing the modem at end-users to minimize delay in the outbound direction and the first hop router at the

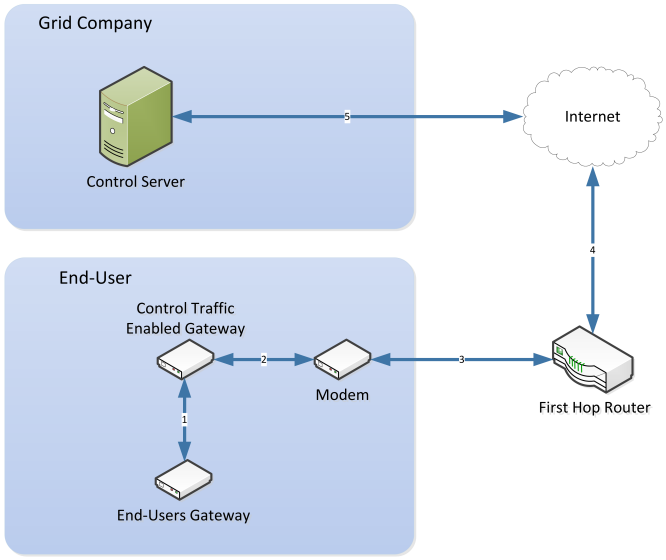


Figure 52: Gateway Deployed at Customer.

ISP to reduce delay in the inbound direction. PIE would be a adequate AQM at both the inbound buffer at the first hop router and the outbound buffer in the modem.

Approach two does entail some problems. The bottleneck link is the last mile link, this entails that the placement of congestion is still at the modem and at the first hop router. Placing a scheme like PIE at the gateway would not help to mitigate the congestion at the buffers in the modem and at the first hop router.

However, if the measured queue latency in the AQM managing the outbound buffer at the gateway was somehow reflecting the queue latency at the modem the drop probability at the gateway could be used to control this. In other words, if the queue latency at the modem can be estimated or measured, AQM deployed at the gateway could control this queue delay.

6.3.2 Estimating the Filling grade of the Buffers in the Modem

To be able to provide the AQM with a queue delay to control, the queue delay must be measured or estimated. The gateway can measure how much traffic is flowing to the modem, and the RTT to the first hop router can be measured at intervals.

An estimator of filling grade, Q_{len}, is given by:

$$Q_{len} = \sum_{i=0}^{\infty} u_i - (C \cdot s) \tag{28}$$

, where u_i is the number of bits uploaded since last iteration, C is the upload rate of the link, and s is the sample rate. The estimator sums up the difference in arriving

traffic and departing traffic at the modem, thereby estimating how much is stored in buffer.

Every measurement of the RTT, $l_{rtt}(k)$, the filling grade Q_{len} is updated to:

$$Q_{len} = D_{queue} \cdot C \quad D_{queue} = l_{rtt}(k) - \underset{i \in [0,k]}{\text{median}}\{l_{rtt}(i)\} \tag{29}$$

, where C is the upload rate, and D_{queue} is the measured queueing delay. This is done to remove estimation errors.

Figure 53 shows an estimator of filling grade of the outbound buffer in the modem at Rindadalsvegen 30. Input is link capacity C, traffic arrival at the modem u, and RTT latency measurements at 1 s intervals of the last mile link, link 3 in Figure 52.

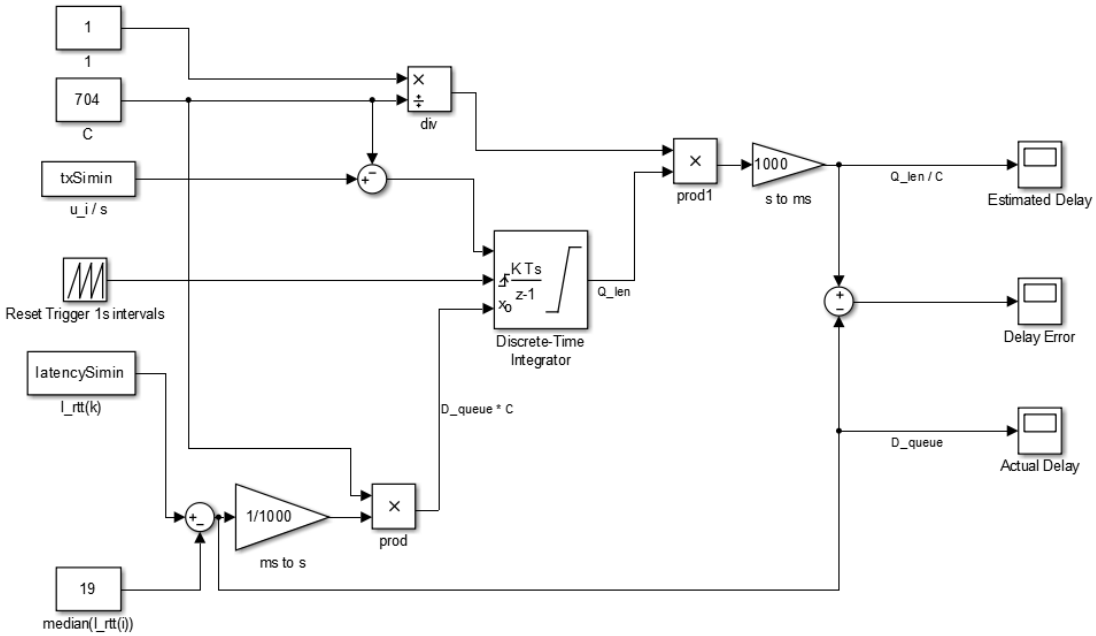


Figure 53: Estimator of queue delay, D_{queue}, at outbound buffer in modem.

The estimator in Figure 53 was run with the measurements shown in Figure 32 as input for a duration of 500 s.

Table 21: Statistical metrics of the delay estimation error, shown as "Delay Error" in Figure 53.

Duration	Max	Min	Mean	Median	SD	Variance
500 s	456 ms	−472 ms	0.22 ms	0 ms	55	3075

The metrics in Table 21 show that this approach might work for the outbound direction. The estimations of filling grade could be used as input to an AQM as shown

in Figure 54. The output of this AQM or latency controller is the drop probability for the outbound direction at the gateway in Figure 51.

The latency arising from either outbound or inbound direction can not be distinguished from each other though. This would entail that it is not possible to estimate both the filling grade in the outgoing buffer and in the buffer at the first hop router by only measuring the RTT.

However, the incoming traffic can be thought of as smoothed from the incoming buffer, i.e. never above the maximum bandwidth. If the AQM scheme for the inbound direction tries to keep the arriving traffic below the maximum bandwidth, the filling grade of the buffer in the incoming direction can be kept to a minimum at least, this would hurt bandwidth utilization though.

The input and output of the latency controller is shown in Figure 54. The output of this latency controller is the drop probability in the inbound direction of the gateway in Figure 51.

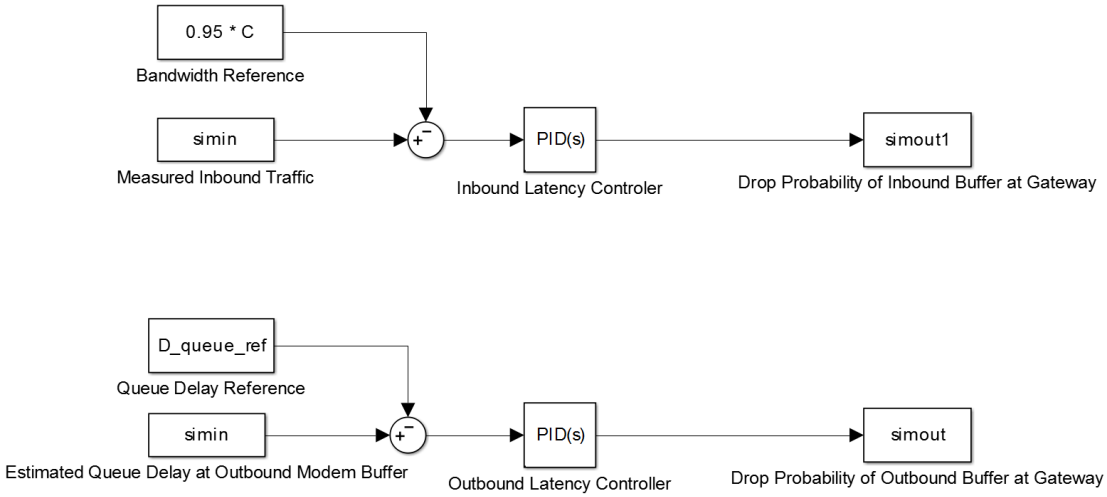


Figure 54: Latency controllers both in the outbound and the inbound direction.

The advantage of this approach, placing a gateway between the end-user and the modem is that nothing of the existing topology outside the end-user would have to be altered, further increasing the economic gains of using Internet communications.

This approach is additionally compatible with prioritization of traffic. Different buffers for different types of traffic could still be in place without hurting the estimation and the latency controllers. More verification work on the validity of the approach would have to be performed though. The estimator for filling grade of the buffer in outbound direction has been simulated and shows promising results, but the latency controller in the inbound direction has not been thoroughly tested.

6.3.3 Recomendation to Minimize Latency in End-User's Link

To reduce maximum latency, the maximum buffer size that traffic flows through is critical. So if a guaranteed maximum latency is needed, an approach where the buffers are designed to impose that maximum latency should be used, by choosing a size calculated by (23).

This might impact the utilization though, therefore to achieve better utilization of the bandwidth, the private traffic from end-user should be sent to a buffer better suited for high utilization, e.g. calculated by (12), and the real-time traffic should be sent to a buffer suited for low delay queueing, i.e. a prioritization of traffic.

The buffer with real-time traffic does not have to incorporate AQM assuming this traffic in itself does not impose congestion. The other buffer should however incorporate AQM, to minimize the experienced latency for end-user.

How to actually implement prioritization of traffic will not be suggested by this thesis, but the fact that IP supports prioritization in header fields suggests that this can be solved.

All of the above can be achieved by utilising approach one. Approach two is as mentioned in the previous section dependent upon estimation and AQM to mitigate congestion at the modem and first hop router, therefore not compatible a simple buffer sizing scheme.

6.3.4 Comment on Multihoming

As mentioned in section 3.4.3, CRNA [13] includes some promising results regarding the prospects of multihoming. Five nines availability was achieved by utilising two independent cellular network providers. An approach where a cellular connection is provided as backup in the gateway might be interesting.

6.3.5 Comment on Safety when Using Internet for Control

The Internet is a public medium, and a comment on the issues regarding safety is therefore in its place. The aforementioned approaches to minimize latency at an end-user and calculating max latency for a given Internet path might prove adequate for real-time applications. However, as argued by Floyd and Paxson [17] the Internet is a moving target, i.e. the characteristics of the traffic might suddenly change.

If an entity wants to reek havoc on the real-time application, there are measures to significantly decrease performance on the Internet. E.g. saturating buffers with UDP traffic can easily increase both packet-loss rates and delays by several orders of magnitude. Although schemes like PIE would minimize the impact on latency, packet-loss would be increased.

Therefore, a real-time application deployed over Internet communications should ensure that a breach of timely delivery can be handled without breach of safety for both people and system.

7 Conclusion

This thesis has presented literature describing the main focus to minimize delay in today's Internet. The problems investigated have been congestion and bufferbloat. Mitigations focusing on reducing buffer sizes and implementing AQM schemes have been presented.

Characterization experiments were performed to evaluate ingress of bufferbloat in Norwegian Internet last mile links. Evidence of bufferbloat, lack of AQM, and possible implementation of differentiated services has been found in the ADSL link at Rindadalsvegen 30. The confidence of the evidence is high. No conclusive evidence of neither bufferbloat nor lack of or presence of AQM could be found on the link at Tistelvegen 24.

The characterization experiments showed that delay induced by a single TCP flow saturating the uplink at Rindadalsvegen 30 followed a normal distribution. Backing up claims by Appenzeller et al. [6].

The problem with calculating max latency over an Internet path is reduced to calculating max queueing delay at last mile links and measuring the median end-to-end latency, assuming that the Internet core is not congested.

A gateway placed behind the modem at end-user is suggested, incorporating a suggested AQM scheme both estimating the queue delay at the modem to control the outbound latency, and controlling the inbound latency by control of incoming bandwidth. The estimation of outbound latency shows promising performance.

The recommendation offered by this thesis to minimize latencies in end-users links is to alter buffer sizes to impose a given maximum queue delay to real-time traffic, and providing prioritization of real-time traffic over private traffic.

7.1 Future Work

Based on the results possibly showing prioritization of ICMP ping at Rindadalsvegen, it is recommended that the ingress and usability of existing differentiated services approaches for real-time applications is investigated.

Furthermore, if the gateway presented as approach two in section 6.3.1 is to be implemented, verification work on the estimation approach for the outbound direction and the bandwidth monitoring approach for the inbound direction should be performed.

References

[1] Openadr. URL http://www.openadr.org/.

[2] Nslu2 - linux - optware, May 27. 2014. URL http://www.nslu2-linux.org/wiki/Optware/HomePage.

[3] E-gotham, May 26. 2014. URL http://www.e-gotham.eu/.

[4] Aggregate traffic for nix, May 13. 2014. URL http://mrtg.uio.no/mrtg/nix/nix-sum.html.

[5] Minor optware flaw (ipkg) with optware-devel package, May 26. 2014. URL http://blog.debruin.org/post/2012/03/19/Minor-optware-flaw-with-optware-devel-package.

[6] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. *SIGCOMM Comput. Commun. Rev.*, 34(4):281–292, Aug. 2004. ISSN 0146-4833. doi: 10.1145/1030194.1015499. URL http://doi.acm.org/10.1145/1030194.1015499.

[7] N. Beheshti, Y. Ganjali, A. Goel, and N. McKeown. Obtaining high throughput in networks with tiny buffers. In *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, pages 65–69, June 2008. doi: 10.1109/IWQOS.2008.13.

[8] N. Beheshti, N. Mckeown, Y. Ganjali, G. Salmon, and M. Ghobadi. Experimental study of router buffer sizing. In *Proc. ACM/USENIX Internet Measurement Conference*, 2008.

[9] N. Beheshti, E. Burmeister, Y. Ganjali, J. Bowers, D. Blumenthal, and N. McKeown. Optical packet buffers for backbone internet routers. *Networking, IEEE/ACM Transactions on*, 18(5):1599–1609, Oct 2010. ISSN 1063-6692. doi: 10.1109/TNET.2010.2048924.

[10] O.-H. Borlaug, R. R. F. Lopes, and S. Hendseth. Latency reliability of application protocols over broadband links in a smart grid context. 12 2013.

[11] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages*. Pearson Education Limited, fourth edition edition, 2009.

[12] W. Changhong, Z. Lixian, and W. Qiyong. Controller design based on model predictive control for real-time tracking synchronization in closed-loop control via internet. In *Control Applications, 2004. Proceedings of the 2004 IEEE International Conference on*, volume 1, pages 260–265 Vol.1, 2004. doi: 10.1109/CCA.2004.1387221.

[13] CRNA. Robusthet i norske mobilnett, tilstandsrapport 2013, (robustness in norwegian cellular networks, state report 2013). 2014.

[14] H. B. Eriksen. Struktur og sikkerhet av nettverk ved integrerte operasjoner, (structure and safety of networks in integrated operations). Master's thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2013.

[15] X. Fang, S. Misra, G. Xue, and D. Yang. Smart grid - the new and improved power grid: A survey. *IEEE Communications Surveys Tutorials*, 14(1):944 –980, quarter 2012. ISSN 1553-877X. doi: 10.1109/SURV.2011.101911.00087.

[16] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, Aug 1993. ISSN 1063-6692. doi: 10.1109/90.251892.

[17] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Trans. Netw.*, 9(4):392–403, Aug. 2001. ISSN 1063-6692. doi: 10.1109/90.944338. URL `http://dx.doi.org/10.1109/90.944338`.

[18] freonchill. best dd-wrt router for the money..., May 26. 2014. URL `http://www.dd-wrt.com/phpBB2/viewtopic.php?t=47284&view=next&sid=02299fb0db7b7129cafcb6513cad58d7`.

[19] J. Gettys. Bufferbloat: Dark buffers in the internet. *Internet Computing, IEEE*, 15 (3):96–96, May 2011. ISSN 1089-7801. doi: 10.1109/MIC.2011.56.

[20] E. Gundegjerde. Smart i lyse - hva naa?? (smart at lyse - what now?). 03 2014.

[21] V. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. Hancke. A survey on smart grid potential applications and communication requirements. *IEEE Transactions on Industrial Informatics*, 9(1):28–42, 2013.

[22] gwyneth. Installing optware on tomatousb, May 26. 2014. URL `http://gwyneth.blogspot.no/2012/03/installing-optware-on-tomatousb.html`.

[23] S. Ha, I. Rhee, and L. Xu. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008. ISSN 0163-5980. doi: 10.1145/1400097.1400105. URL `http://doi.acm.org/10.1145/1400097.1400105`.

[24] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong. On designing improved controllers for aqm routers supporting tcp flows. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1726–1734 vol.3, 2001. doi: 10.1109/INFCOM.2001.916670.

[25] E. Hossain, Z. Han, and H. V. Poor, editors. *Smart Grid Communications and Networking*. Cambridge University Press, 2012.

[26] V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM. ISBN 0-89791-279-9. doi: 10.1145/52324.52356. URL `http://doi.acm.org/10.1145/52324.52356`.

[27] John. Installing optware in tomato usb shibby, May 26. 2014. URL http://uk.nyclee.net/2012/04/28/installing-optware-in-tomato-usb-shibby/.

[28] K. Liyanage, M. A. M. Manaz, A. Yokoyama, Y. Ota, H. Taniguchi, and T. Nakajima. Impact of communication over a tcp/ip network on the performance of a coordinated control scheme to reduce power fluctuation due to distributed renewable energy generation. In *Industrial and Information Systems (ICIIS), 2011 6th IEEE International Conference on*, pages 198–203, 2011. doi: 10.1109/ICIINFS.2011.6038066.

[29] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 90–102, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-771-4. doi: 10.1145/1644893.1644904. URL http://doi.acm.org/10.1145/1644893.1644904.

[30] J. Nagle. On packet switches with infinite storage. *Communications, IEEE Transactions on*, 35(4):435–438, Apr 1987. ISSN 0090-6778. doi: 10.1109/TCOM.1987.1096782.

[31] K. Nichols and V. Jacobson. Controlling queue delay. *Queue*, 10(5):20:20–20:34, May 2012. ISSN 1542-7730. doi: 10.1145/2208917.2209336. URL http://doi.acm.org/10.1145/2208917.2209336.

[32] D. Niyato, Q. Dong, P. Wang, and E. Hossain. Optimizations of power consumption and supply in the smart grid: Analysis of the impact of data communication reliability. *Smart Grid, IEEE Transactions on*, 4(1):21–35, 2013. ISSN 1949-3053. doi: 10.1109/TSG.2012.2224677.

[33] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *High Performance Switching and Routing (HPSR), 2013 IEEE 14th International Conference on*, pages 148–155, July 2013. doi: 10.1109/HPSR.2013.6602305.

[34] M. Rupental. http://tomato.groov.pl/.

[35] T. Sauter and M. Lobashov. End-to-end communication architecture for smart grids. *IEEE Transactions on Industrial Electronics*, 58(4):1218 –1228, april 2011. ISSN 0278-0046. doi: 10.1109/TIE.2010.2070771.

[36] SSB. Tabell: 06992: Internettabonnement, etter husholdningstype og husholdningsinntekt (prosent) (table: 06992: Internet subscriptions, by household type and household income (percent)), May 26. 2014. URL https://www.ssb.no/statistikkbanken/SelectVarVal/Define.asp?MainTable=IKTHtypeHinnt&KortNavnWeb=ikthus&PLanguage=0&checked=true.

[37] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband internet performance: A view from the gateway. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 134–145, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0797-0. doi: 10.1145/2018436.2018452. URL http://doi.acm.org/10.1145/2018436.2018452.

[38] C. Villamizar and C. Song. High performance tcp in ansnet. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, Oct. 1994. ISSN 0146-4833. doi: 10.1145/205511.205520. URL http://doi.acm.org/10.1145/205511.205520.

[39] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524 vol.4, March 2004. doi: 10.1109/INFCOM.2004.1354672.

Appendices

A DVD and Contents listing

Table 22: Contents of DVD

Path	**Description**
Final Masters thesis.pdf	A digital copy of this thesis.
Matlab model/	Simulink models portrayed in figures 53 and 54.
Software/DelayMonitor/	Delay Monitor source code and makefile.
Software/TrafficMonitor/	Traffic Monitor source code and makefile.
Software/Scripts/	Matlab scripts to read and plot `.csv` files from Delay Monitor and Traffic Monitor.
Statistics/	Measurements used in results.
Tomato firmware and config/	Firmware used on the routers and config files.
Tomato firmware and config/Packages/	Optware packages used.

B Hardware

A short description of hardware used in the experiments performed in this thesis.

B.1 NETGEAR WNR3500Lv1

Table 23: Hardware information about router used by Sundaresan et al. [37].

Model:	Netgear WNR3500L v1
Chipset:	Broadcom BCM4718A1
CPU Freq:	480 MHz
Flash Size:	8 MB
RAM Size:	64 MB

B.2 NETGEAR WNR3500Lv2

Table 24: Hardware information about router used in this characterization experiments.

Model:	Netgear WNR3500L v2
Chipset:	Broadcom BCM5357 chip rev 2 pkg 10
CPU Freq:	480 MHz
Flash Size:	128 MB
RAM Size:	128 MB

B.3 NOX Box

Table 25: Hardware information about NOX Box.

Model:	ALIX 2D13
Processor:	AMD Geode
CPU Freq:	500 MHz
Flash Size:	2 GB
RAM Size:	256 MB
OS:	Debian Linux

B.4 DELL optiplex 9010

Table 26: Hardware on office computer at NTNU Gløshaugen.

CPU:	INTEL CORE i5-3470 CPU @ 3.20 GHz
RAM:	8.00 GB
OS:	Windows 7 Enterprise (64-bit), Service Pack 1

C Software and Software Tools

This appendix presents some information on the software and the software tools used in the work with this report and provides some guidance in using them.

C.1 TomatoUSB

TomatoUSB is an open-source router firmware developed by several contributors [34]. It is based on the Linux kernel. The firmware runs on several consumer routers or gateways and tutorials for several uses are widespread. One of the features provided by TomatoUSB is the installation of optware. Optware is designed to work on small embedded devices and is a way to distribute packages on such devices. The use of the packet manager `ipkg`, much like `apt` and `yum`, makes it a easy and well known way to add extra features to the firmware.

The firmware used in this report is configured and compiled by Michal Rupental, also known as shibby [34]. He has taken up the torch from those that have contributed heavily in the development of the tomato firmware, and has done a great deal of development work on the firmware.

The current firmware actually being run on the two routers in this report is build `build5x-116-EN` from Michal Rupental's website [34]. An image of the firmware and settings configuration file is included on the DVD in appendix A.

C.1.1 Development on Tomato USB firmware

There are two ways of developing and compiling software for the tomato router firmware. One is to cross-compile the software in a development environment. Another way is to use optware, see appendix C.1.2, to install a buildroot and `gcc` on the router itself and build the application there.

Optware makes development on the tomato firmware easy. Once Optware is installed, all that is needed to build and compile software on the router is to install two packages, `optware-devel` and `buildroot`.

With the build provided by shibby installation of optware is made easy by providing a install script, `optware-install.sh`. The steps necessary to have a working set of optware, compiler and buildroot are:

1. Creating a USB flash drive with a partition readable by TomatoUSB. At first ext2 was chosen due to less I/O required per write to the USB drive, but because of a lot of problems with data becoming corrupt ext3 was used at the end. An easy way of creating a ext2 partition on a usb flash drive is to run the command:
   ```
   # mkfs.ext2 -L optware /dev/sda1
   ```

2. The USB drive must be mounted under `/opt`. If the script below is added to the "Run after mounting" box under USB support a partition with the name `optware` will be mounted at `/opt`.
   ```
   if [ -d /mnt/optware ]; then
   ```

```
mount -o bind /mnt/optware /opt
fi
```

3. With the USB drive mounted at the correct location, the shell script `optware-install.sh` located at `/usr/sbin/optware-install.sh` can be run.

4. Now the `ipkg` commands can be executed.
   To update package list:
   ```
   # ipkg update
   ```
   List all available packages:
   ```
   # ipkg list
   ```
   Install PACKAGE:
   ```
   # ipkg install PACKAGE
   ```
   Remove PACKAGE:
   ```
   # ipkg remove PACKAGE
   ```

5. To be able to build own software on the router the packages `optware-devel` and `buildroot` must be installed.
   ```
   # ipkg install optware-devel buildroot
   ```

The steps above are compiled from [22, 27].
Problems that may arise:

1. `gcc: can't resolve symbol _obstack_begin` when running make. To fix [18]:
   ```
   # unset LD_LIBRARY_PATH
   ```

2. Sometimes wget fails [5]

C.1.2 optware

Optware was originially developed to aid those who wanted a lightweight package manager for use in the lightweight `Linux` distro for the `Linksys NSLU2`, a low cost network storage device [2]. Since it has been adopted by several lighweight `Linux` distributions including TomatoUSB, DD-WRT, Open-WRT, and webOS. Optware is a lightweight package manager.

C.2 Implemented Software

This appendix contains a description of the software developed in this thesis, and some directions use.

C.2.1 Traffic Monitor

The traffic monitor records the traffic at given intervals in a file. The traffic is read from the statistics variables provided by the `Linux` kernel for a specified network interface.

The statistics can be found at: `/sys/class/net/NIC_NAME/statistics/`
The statistics provided in the `TomatoUSB` firmware include:

- `rx_bytes`, number of bytes received over this network interface.
- `tx_bytes`, number of bytes sent over this network interface.

Known Bugs

1. The time stamp is recorded is not always recorded with 3 digit accuracy of milliseconds, 5 ms is recorded as `yyyy-mm-dd HH:MM:SS:5` and 14 ms is recorded as `yyyy-mm-dd HH:MM:SS:14`

C.2.2 Delay Monitor

The delay monitor sends a UDP packet with an incrementing number to a server and records the reply from the server, and calculates the RTT for the packet.

The following statistics are stored for all successful packets:

- Packet number
- Send time
- Receive time
- RTT in seconds
- RTT in milliseconds
- A moving average of 10 RTTs

Packets time out after 10 seconds and are then stored with packet number and send time. Packet-loss for the entire run time of the program is stored in a log file.

Known Bugs

1. The time out thread sleeps too much.
2. The pinger sends packets even if the buffer to hold them is full, and they can not be recorded. This results in some packets getting old timestamps when they are received from the server.
3. The time stamp is recorded is not always recorded with 3 digit accuracy of milliseconds, 5 ms is recorded as `yyyy-mm-dd HH:MM:SS:5` and 14 ms is recorded as `yyyy-mm-dd HH:MM:SS:14`

D Code

This section list some of the code of the traffic monitor and the delay monitor, and matlab scripts to extract the measurements from the files created by the traffic monitor and the delay monitor follows.

D.1 Traffic Monitor

Below is the main loop of the traffic monitor.

```
1  while(keepRunning){
2    txFile=fopen(txFileName, "rb");
3    rxFile=fopen(rxFileName, "rb");
4    if((txFile!=NULL)&&(rxFile!=NULL)){
5      fread(txData, 1, sizeOfByteFiles, txFile);
6      fread(rxData, 1, sizeOfByteFiles, rxFile);
7      fclose(txFile);
8      fclose(rxFile);
9      txConvert=getNumberFromString(txData);
10     rxConvert=getNumberFromString(rxData);
11     if((txConvert>=0)&&(rxConvert>=0)){
12       gettimeofday(&now,NULL);
13       timeval_subtract(&delta,now,last);
14       deltat=delta.tv_sec;
15       deltat2=(double)delta.tv_usec/1000000;
16       deltat=deltat+deltat2;
17       if(txConvert<txBytes){
18         //Handle wrap around
19       }else{
20         txSpeed=(txConvert-txBytes)*8/deltat/statsPrefixCalc;
21       }
22       if(rxConvert<rxBytes){
23         //Handle wrap around
24       }else{
25         rxSpeed=(rxConvert-rxBytes)*8/deltat/statsPrefixCalc;
26       }
27       if(!firstIteration){
28         if(!silentFlag){
29           printf("%.6f\t%.2f %sbit/s\t\t%.2f %sbits/s\t",deltat,txSpeed,
                 statsPrefix,rxSpeed,statsPrefix);
30         }
31         pthread_mutex_lock(&statFileMutex);
32         fprintf(statFile,"%.6f,%.4f,%.4f,",deltat,txSpeed,rxSpeed);
33         pthread_mutex_unlock(&statFileMutex);
34         char currentTime[84] = "";
35         getTimeWithMilliseconds(&now, currentTime);
36         if(!silentFlag){
37           printf("%s \n", currentTime);
38         }
39         pthread_mutex_lock(&statFileMutex);
40         fprintf(statFile,"%s\n", currentTime);
41         fflush(statFile);
42         pthread_mutex_unlock(&statFileMutex);
```

```
43        }else{
44          firstIteration=0;
45        }
46      }
47      txBytes=txConvert;
48      rxBytes=rxConvert;
49      copy_timeval(&last,&now);
50    }else{
51      if(!silentFlag){
52        printf("Error opening TX and RX files\n");
53      }
54      fclose(txFile);
55      fclose(rxFile);
56    }
57    usleep(msInterval*1000);
58  }
```

D.2 Delay Monitor

Pinger thread of delay monitor:

```
1  void *pinger(void *threadid){
2    struct timeval sent;
3    while(keepRunning){
4      sprintf(buffer,"%d",PacketsSent);
5      send( serverSocket, buffer, strlen( buffer ), 0 );
6      gettimeofday(&sent,NULL);
7      addNewPacket(PacketsSent,&sent);
8      usleep(msInterval*1000);
9      PacketsSent++;
10   }
11 }
```

Reader thread of the delay monitor:

```
1  void *reader(void *threadid){
2    struct timeval sent;
3    struct timeval received;
4    struct timeval delta;
5    double deltat=0;
6    double deltat2=0;
7    while(keepRunning){
8      if(numberOfPacketsInTransit>0){
9        bytesReceived = recv( serverSocket, buffer, BUF_SIZE, 0 );
10       gettimeofday(&received,NULL);
11       if(getAndRemovePacketInfo(atoi(buffer),&sent)){
12         timeval_subtract(&delta,received,sent);
13         deltat=delta.tv_sec;
14         deltat2=(double)delta.tv_usec/1000000;
15         deltat=deltat+deltat2;
16         char currentTime[84] = "";
17         getTimeWithMilliseconds(&received, currentTime);
18         char sentTime[84] = "";
```

```
19          getTimeWithMilliseconds(&sent, sentTime);
20          if(!silentFlag){
21            printf("%d\t\t%s\t\t%s\t\t%.6f\t%d\t%d\n",atoi(buffer),sentTime,
                  currentTime,deltat,(int)(deltat*1000),(int)(movingAverage
                  *1000));
22          }
23          addDelayToAverage(deltat);
24          pthread_mutex_lock(&statFileMutex);
25          fprintf(statFile,"%d,%s,%s,%.6f,%d,%d\n",atoi(buffer),sentTime,
                currentTime,deltat,(int)(deltat*1000),(int)(movingAverage*1000)
                );
26          fflush(statFile);
27          pthread_mutex_unlock(&statFileMutex);
28        }else{
29          printf("Error getting packet\n");
30          keepRunning=0;
31        }
32        usleep(1000);
33      }
34      usleep(1000);
35    }
36  }
```

Time out thread of delay monitor:

```
1  void *checkOldestForTimeout(void *threadid){
2    struct timeval now;
3    struct timeval delta;
4    int tempCounter;
5    char sentTime[84] = "";
6    while(keepRunning){
7      if(numberOfPacketsInTransit>0){
8        gettimeofday(&now,NULL);
9        timeval_subtract(&delta,now,packetTimes[startOfPackets]);
10       if((delta.tv_sec>=PACKET_TIMEOUT)){
11         droppedPackets++;
12         tempCounter=counterNumber[startOfPackets];
13         getAndRemovePacketInfo(counterNumber[startOfPackets],&now);
14         getTimeWithMilliseconds(&now, sentTime);
15         if(!silentFlag){
16           printf("%d\t\t%s\t\t-1\t\t\t\t-1\t\t-1\n",tempCounter,sentTime);
17         }
18         pthread_mutex_lock(&droppedFileMutex);
19         fprintf(droppedFile,"%d,%s,-1,-1,-1,\n",tempCounter,sentTime);
20         fflush(droppedFile);
21         pthread_mutex_unlock(&droppedFileMutex);
22       }
23       usleep(1000);
24     }
25     sleep(1);
26   }
27 }
```

D.3 Scripts

D.3.1 Load Delay and Traffic Data

Below is a matlab script to read files with recorded delay measurements from the delay monitor. This can additionally be used to read the files with dropped packages from the delay monitor.

```
for i=1:length(delayFiles),
    fileID = fopen(['Rindadalsvegen/' delayFiles{i}],'r');
    dataArray = textscan(fileID, '%f%s%s%f%f%f%[^\n\r]', 'Delimiter', ',',
        'EmptyValue' ,NaN,'HeaderLines' ,1, 'ReturnOnError', false);
    fclose(fileID);
    for j=1:length(dataArray{:, 2}),
        if length(dataArray{:, 2}{j}) <= 21
            string=regexp(dataArray{:, 2}{j},':','split');
            string=[string{1} ':' string{2} ':' string{3} ':0' string{4}];
            dataArray{:, 2}{j}=string;
       end
        if length(dataArray{:, 2}{j}) <= 22
            string=regexp(dataArray{:, 2}{j},':','split');
            string=[string{1} ':' string{2} ':' string{3} ':0' string{4}];
            dataArray{:, 2}{j}=string;
       end
    end
    for j=1:length(dataArray{:, 3}),
        if length(dataArray{:, 2}{j}) <= 21
            string=regexp(dataArray{:, 2}{j},':','split');
            string=[string{1} ':' string{2} ':' string{3} ':0' string{4}];
            dataArray{:, 2}{j}=string;
       end
       if length(dataArray{:, 3}{j}) <= 22
            string=regexp(dataArray{:, 3}{j},':','split');
            string=[string{1} ':' string{2} ':' string{3} ':0' string{4}];
            dataArray{:, 3}{j}=string;
       end
    end
    dataArray{2} = datenum(dataArray{2}, 'yyyy-mm-dd HH:MM:SS:FFF');
    dataArray{3} = datenum(dataArray{3}, 'yyyy-mm-dd HH:MM:SS:FFF');
    Number = [Number; dataArray{:, 1}];
    SentTime = [SentTime; dataArray{:, 2}];
    ReceivedTime = [ReceivedTime; dataArray{:, 3}];
    RTT = [RTT; dataArray{:, 5}];
end
```

Below is a matlab script to read files with recorded traffic measurements, from the traffic monitor.

```
for i=1:length(trafficFiles),
  fileID = fopen(['Rindadalsvegen/' trafficFiles{i}],'r');
  dataArray = textscan(fileID, '%f%f%f%s%[^\n\r]', 'Delimiter', ',', '
      EmptyValue',NaN,'HeaderLines',1, 'ReturnOnError', false);
  fclose(fileID);
    for j=1:length(dataArray{:, 4}),
       if length(dataArray{:, 4}{j}) <= 21
           string=regexp(dataArray{:, 4}{j},':','split');
           string=[string{1} ':' string{4} ':' string{3} ':0' string{4}];
           dataArray{:, 4}{j}=string;
       end
       if length(dataArray{:, 4}{j}) <= 22
           string=regexp(dataArray{:, 4}{j},':','split');
           string=[string{1} ':' string{4} ':' string{3} ':0' string{4}];
           dataArray{:, 4}{j}=string;
       end
    end
  dataArray{4} = datenum(dataArray{4}, 'yyyy-mm-dd HH:MM:SS:FFF');
  timeSinceLastSample = [timeSinceLastSample; dataArray{:, 1}];
  TXbitss = [TXbitss; dataArray{:, 2}];
  RXbitss = [RXbitss; dataArray{:, 3}];
  Time = [Time; dataArray{:, 4}];
end
```