Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso - Chile

Testing, Detection and Possible Solutions for the Bufferbloat Phenomenon on Networks.

Thesis Degree

JUAN S. CATALAN OLMOS

Definición de Tema de Memoria
para optar al Título de:
INGENIERO CIVIL INFORMATICO

Referent Professor: Horst H. von Brand Skopnik
Coreferent Professor: Raúl Monge Anwandter

October 19, 2014

Glossary

Bandwidth Delay Product (BDP) An amount of data measured in bits. It is equivalent to the maximum amount of data on the network circuit at any given time. Commonly measured by the RTT*bandwidth.

Bottleneck bandwidth The smallest bandwidth along the path. Packets cannot arrive at the destination any faster than it takes to send a packet at the bottleneck rate.

Bufferbloat Refers to excess buffering inside a network, resulting in high latency and reduced throughput.

Congestion Window (cwin) One of the factors that determines the bytes that can be outstanding at any time. It prevents the overload of the link with too much traffic. The size is calculated by estimating the congestion between the two places.

Explicit Congestion Notification (ECN) Defined in RFC 3168. ECN allows end-to-end notification of network congestion without dropping packets. ECN is an optional feature that may be used between two ECN-enabled endpoints when the underlying network infrastructure also supports it.

Maximum Transmission Unit (MTU) MTU of a layer is the largest protocol data unit that the layer can pass onwards. In ethernet the MTU is 1500 bytes.

Slow start threshold (ssthrsh) It determines whether the TCP should do Slow Start or Congestion Avoidance. It is initialized to a large value, and after a congestion is signaled, cwin is divided in half and ssthrsh is set to cwin.

Sojourn A period of time when you stay in place as a traveler or guest. Example: Sojourned for a month at the mountains.

System Throughput Corresponds to the fastest rate at which the count of packets transmitted to the destination by the network is equal to the number of packets sent into the network.

Throughput Measure of the efficiency of a network expressed as the data(bits or packets) transfer rate of useful and non-redundant information. It depends on factors such as bandwidth, line congestion, error correction, etc.

Contents

1 Introduction **5**

2 The Transport Control Protocol **8**
2.1 Fundamentals of TCP . . . 8
2.2 TCP's Phases . . . 10
2.2.1 Slow start and Congestion avoidance . . . 11
2.2.2 Congestion control algorithms . . . 12
2.2.3 Fast retransmit and fast recovery . . . 15

3 Router's Buffers effects **16**
3.1 Full Buffers . . . 17
3.2 High Latency . . . 17
3.3 Sizing Router's Buffers . . . 18

4 Active Queue Management **21**
4.1 RED . . . 22
4.2 BLUE . . . 24
4.3 CoDel . . . 25

5 Hidden Flaws and Bufferbloat **27**

6 Experimental Work **29**
6.1 The Test Setup . . . 29
6.1.1 Hardware Characterization . . . 30
6.2 Tools Definition . . . 31
6.2.1 Speedtest . . . 32
6.2.2 Netalyzer . . . 32
6.2.3 Iperf . . . 33
6.2.4 Page Benchmarker . . . 34
6.2.5 Smokeping . . . 34
6.3 Test Description . . . 35
6.3.1 Speed test . . . 35
6.3.2 Signs of trouble . . . 36
6.3.3 Collapse test . . . 37
6.3.4 Load benchmark test . . . 38
6.3.5 Smoke the path . . . 38

7 Results **40**
7.1 Speed test . . . 40
7.2 Netalyzr test . . . 42

7.3 Iperf test 45
7.4 Benchmark test 49
7.5 Smokeping Test 52
7.6 Summarising Results 56

8 Conclusions and Further Work **58**

A Speed Test Details results **67**

B Page Benchmark results **68**

List of Figures

1 The Chronology of a Slow-Start. 11
2 The window growth function of CUBIC 13
3 CUBIC's Congestion Window.[i] 13
4 Compound TCP sending window.[ii] 14
5 RED's Flowchart 23
6 BLUE's algorithm 24
7 CoDel droptime interval 26
8 Result Summary example from Netalyzr 33
9 Speed and Pings based on Appendix A, Table 7 40
10 Iperf: RTT graphs for a fiber network 47
11 Iperf: RTT graphs for a network with minor issues 48
12 Iperf: RTT graphs for a network with bad performance 48
13 Iperf: RTT graphs for a network with DNS issues 49
14 Page Benchmark: Total Load Means 50
15 Smokeping: Ping test to National servers 53
16 Smokeping: Ping test to International servers 54
17 Smokeping: Web requests to National servers 55

List of Tables

1 Speed Test: Variation ratio of offered vs measured speed 41
2 Netalyzr Test:DNS resolution time, Buffer time and Performance 43
3 Netalyzr Test: Bandwidth 44
4 Netalyzr Test: Summary of errors and warnings in Netalyzr 46
5 Page Benchmark: Minimum, Maximum and variation ratio. 51
6 Page Benchmark: Ratio over own iteration 52
7 Speed Test: Speeds and Ping measured 67
8 Speed Test: Measured and Contracted Speed 68

9 Page Benchmark: Total Load Means. 69

1 Introduction

According to the reports published by SUBTEL to the first half of 2013, *12.8* per 100 inhabitants have access to broadband Internet [28]. This steady growth is largely due to the emergence of new applications such as streaming video and music, online gaming and bandwidth intensive applications.

Not long ago, the links had a much more limited bandwidth than they have today. With the evolution of electronics and telecommunications, speeds have increased dramatically. With the current level of data flowing through the network, it is important to control congestion that might occur. This is one of the tasks of the routers.

It is important to remember that the traffic in a network is inherently bursty, the role of the buffers in the router is to smooth the flow of traffic. Without any buffering, to allocate the bandwidth evenly would be impossible. But there are some problems with current algorithms; they use tail-drop based queue management that has two big drawbacks: 1.- lockout 2.- full queue that impact with a high queue delay.

Current low hardware prices make memory cheap, and with the "more is better" mentality have led to the inflation and proliferation of buffers everywhere [22]. When a router joins two networks with different bandwidths, each packet is squeezed down in bandwidth, it must stretch out in time since its size stays constant. When the queue starts to grow, more and more memory is deployed leading to massive standing queues. It turns out that this is a recipe for Bufferbloat. Evidence of Bufferbloat has been accumulating over the past decade, but its existence has not yet become a widespread cause for concern.

Bufferbloat creates large delays but no improvement in throughput. It is not a phenomenon treated by queueing or traffic theory, which unfortunately results in it being almost universally missclassified as congestion. The *Bufferbloat* problem, making the window match the pipe size, is hard to address. Window sizes are chosen by senders while

queues manifest at bottleneck gateways.

As mentioned by Jim Gettys and Kathleen Nichols; *Today's networks are suffering from unnecessary latency and poor system performance. The culprit is Bufferbloat, the existence of excessively large and frequently full buffers inside the network. Large buffers have been inserted all over the Internet without sufficient thought or testing. They damage or defeat the fundamental congestion-avoidance algorithms of the Internet's most common transport protocol. Long delays from Bufferbloat are frequently attributed incorrectly to network congestion, and this misinterpretation of the problem leads to the wrong solutions being proposed* [12].

The objective of this thesis work is checking the effects of Bufferbloat phenomenon, test the impact that it has on different networks and to propose solutions. To accomplish this, it first requires to address the following general objectives:

- To define the *Bufferbloat* phenomenon, and explain the impact that it could have on latency and Throughput(related to System Throughput) in Internet.
- To detect its presence by measurements of the latency and throughput in a TCP/IP Network.
- To propose solutions in the implementation of a network where the existence of excessively large and frequently buffers are detected

In order to archieve theses objectives:

- Develop appropriate tests to be able to prove the existence of *Bufferbloat*
- To test and differentiate the possible causes of the excessive latency and throughput reduction in a TCP/IP LAN and check how much is generated by *Bufferbloat* or by a miss-configuration

- To propose configuration of the TCP parameters in a Linux based machine or an algorithm that can help to minimize the phenomenon.

Chapter 2 explains the basis on which TCP was developed and the fundamentals for the current operation of the Internet. The conservation principle upon which all communication protocols are based will be explained. It also describes each of the four phases of TCP.

In Chapter 3 we will review one of the most important components for the communication and the main place where the packages are stored: the Routers. We will analyze how their queues have a destructive size for communication, why they are stalled at full capacity and cause excessive latency. We also consider how to define the appropriate buffer size in a router.

Chapter 4 presents a technique to deal more efficiently with active congestion that could generate not only on endpoints but also on routers. This is Active Queue Management (AQM). It describes the first algorithms, such as RED and BLUE, and mentions CODEL, which aims to solve the Bufferbloat phenomenon. Failure points in both TCP and AQM, are described and analyzed. All this is presented in Chapter 5.

Chapter 6 defines the methodology and tools used to perform each of the tests to determine the existence and the effects of Bufferbloat. Each test is defined by its purpose, starting with some general tests to check the behavior and sanity of the network and moving to a more user-related effects of the Bufferbloat. All the tools used are Open Source, available for all commercial operating systems. In Chapter 7 the results of all tests are presented, leading to the conclusions of this work in chapter 8.

2 The Transport Control Protocol

The Transport Control Protocol or TCP is the main transport protocol of Internet, providing reliable host to host communication over unreliable transport media [23]. The advantages of its design, flexibility and robustness provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer, but for that, one of the costs is the need of careful design so it provides a good service under heavy loads. Many modifications have been done since it was originally defined in 1981.

The root idea of any algorithm related to transport connections must be based on the "*packet conservation principle*". This principle claims:

Definition 2.1. A new packet isn't put into the network until an old packet leaves.

2.1 Fundamentals of TCP

TCP operates basically making two hosts exchange segments of data. The connection between these two hosts is identified uniquely by the network addresses and a 16 bit port number at both hosts. The communication between the hosts is initiated by a three way handshake between them, where the sequence number is synchronized between the participants. The sequence number is a 32-bit number and it is the basis for reliable data transport through an unreliable network. This is, starting from the initial sequence number, each data byte sent as part of the connection has a corresponding sequence number; and only after having being acknowledged by the receiver is the data considered to be transmitted successfully.

TCP makes use of the idea of pipe size and the assumption there was reasonable buffering along the data path to send a window of packets at a time. To control the amount of data that flows through the network path, the receiver sent the information of

how much data it can receive at once, so the network resource is used efficiently. Window size represents how much data a device can handle from its peer at one time before it is passed to the application process. All excess data will just be dropped. This window is also a constraint to the sender, the sender is not allowed to transmit more data than the window before acknowledgment of the data sent.

To manage the data sent and the acknowledgment received for those packets, TCP uses a cumulative scheme. After a packet is in flight from the server with its corresponding sequence number, sent data goes to a retransmission queue where it is held until the corresponding acknowledgement from the other end has come in, or to be resent if not acknowledged within a timeout. When the acknowledgment of a sequence number is received, the sender discards all data with sequence numbers bellow the sequence number in the acknowledgment that has arrived. For retransmission, TCP uses an adaptive scheme. The timeout is automatically set from the measured round trip time of the connection, taking into account the variance of the measured values [16]. This helps avoid retransmitting potentially lost segments too quickly or too slowly.

Because today's networks are dynamic and in different configurations, both topologically speaking as a bandwidth, TCP must handle these changes and still be able to maintain communication between the two hosts. In case of loss of one packet means subsequent packets cannot be acknowledged until the lost packet is retransmitted. This can lead to excessive retransmission and unnecessary load. TCP extensions have been developed that allow the receiver to send selective acknowledgments of block of received data with sequence numbers that are not cumulative with the data acknowledged in the traditional way [20].

Since networks are shared and conditions change along the path, the algorithms continually probe the network and adapt the number of packets in flight. Often there are

decreases in the bandwidth along the way. Such spots are the bottlenecks[iii] and they are important because the performance or capacity of the entire connection (connection as a state between the two hosts) is limited by the resources that this trace has. With this, controlling the optimal rate of data transmission is hard work, and the receiver window as communicated by TCP is not a necessarily a very good indicator. To fix this issue, TCP received an addition to its specification: *congestion window*. The congestion window plays a crucial role in estimating the available bandwidth between the hosts. After this modification, the minimum between the receiver window and the congestion window is used as the transmit limit. All the TCP additions attempt to keep the network operating near the inflection point where throughput is maximized, delay is minimized, and little loss occurs.

2.2 TCP's Phases

To provide a good service, it's needed that TCP flows respond to orders given by the hosts machines that controls the connection during congestion. This characteristic of flows is called "*responsiveness*" and tells when flows must back-off during congestion. In the second half of the 80's, TCP began to experience a strange phenomenon that manifested itself through a dramatically diminish throughput. After a deep analysis, Van Jacobson showed that TCP needed a mechanism to limit transmission speed in the face of congestion [16]. This lead to the development of a better congestion control mechanism for TCP, which was added as a requirement for hosts connected to Internet [3]. Since then, the congestion controls methods implemented in TCP has been updated several times, adding two new algorithms, the *slow start* and *congestion avoidance* were designed to keep in "equilibrium" the data that is in exchange. Later a third where added, the *fast retransmit and fast recovery*.

[iii] Or the Bottleneck bandwidth points

2.2.1 Slow start and Congestion avoidance

The idea behind bandwidth occupancy in TCP is to use as much as it is allow to use. With this in mind, slow start proves the network by increasing exponentially the rate of how many packets sends; the Congestion Window (cwin) is increases by one packet and adds another for each packet acknowledged. This behavior continues until the Slow start threshold (ssthrsh) is reached or congestion is signaled. If congestion is detected, the slow start threshold is reset to half of the amount of congestion window, at the time that congestion window is reset to a size of one segment, and slow start is restarted. In case that the slow start threshold is reached, the sender switches to congestion avoidance mode, which is employed to maintain the transmission rate, increasing by one segment each RTT[iv].

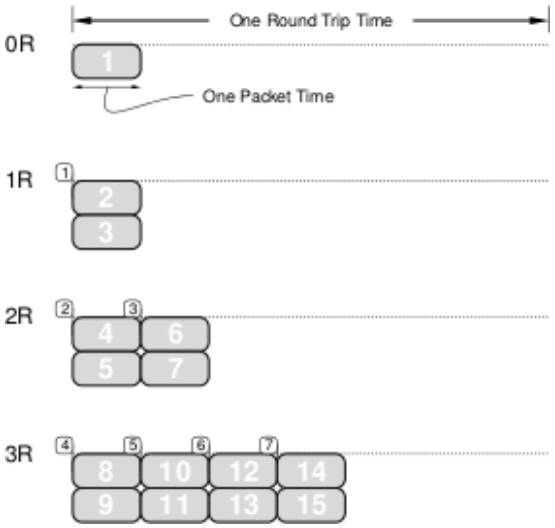


Figure 1: The Chronology of a Slow-Start. [16]

TCP must be smart enough to signal to the hosts when path is experiencing congestion, so it must have policies that decrease the network utilization or increase it if congestion signals are received or not. Also, based on the afore mentioned principle ("*packet conservation principle*"), congestion collapse would become the exception rather than a rule. Thus congestion control involves finding place that violate conservation and fixing them. It is important to detect early this misbehavior, since new packets are added exponentially

[iv]Each ack increases the congestion window by $ss * ss/cwnd$. This results in a linear increase of the congestion window; that means in 1 RTT, congestion window will be increased by 1 [16] [24] [6]

also the congestion grows exponentially, so if is detected early, only small adjustments will cure it.

Typically, when the sender is signaled that packets are been dropped, TCP/IP networks understand that as congestion, but typical effects include queuing delay, packet loss or the blocking of new connections. New implementations define a explicit congestion notification methods [25], that allows notification of congestion between the end points without any packet loss. TCP specification mandates initially setting the congestion window to between 2 and 4 segments of data depending on the segment size [1].

2.2.2 Congestion control algorithms

One of the biggest problems that TCP had in the past years for its development has been the achieve of optimal utilization preventing congestion associated with the differences of bandwidth existing in the medium through which communication is propagated. That is why lately there have been various implementations for proper handling of the original algorithm deployed in TCP for attaining prevent or lessen the effects of this problem. Among the algorithms that currently exist, the two predominant in modern operating systems will be mentioned: CUBIC and Compound TCP.

2.2.2.1 CUBIC TCP is the default implementation for congestion control in the Linux kernel since 2.6.19. *Scaling the window growth rate to match large Bandwidth Delay Product (BDP) is rather straightforward, tackling the fairness issues of new protocols has remained as a major challenge. Although BDP implies the network capacity by packet count(or window size), packet count is not adequate to characterize TCP performance because growth rate of TCP depends on RTT* [14].

Because the basis of their growth is a cubic function (hence its name), has a less aggressive behavior than their predecessors, while retaining excellent scalability, fairness

and stability. As can be seen in Figure 2, the cubic function is compound of three parts: a concave part where the window grows rapidly over small time values, a plateau set at the previous maximal congestion window size, and the end portion where the increasing bandwidth above the maximum which exhibits a convex shape.

As can be better seen in the formula at Figure 3, the congestion window ain't dependent of its previous acknowledged packets; instead, the congestion window is computed at each step from a calibrated function. With this calibration, the plateau is at the previous maximal congestion window size. This feature allow the algorithm to respond faster to changes in available bandwidth. Since in previous algorithms throughput is defined by the packet loss rate as well as RTT, the throughput in CUBIC is defined by only the packet loss rate.

Another feature that CUBIC has with respect to its predecessor in the Linux kernel, it also changes the behavior of the Slow Start algorithm, replacing it with

$$W_{cubic} = C(t-K)^3 + W_{max}$$

Figure 3: CUBIC's Congestion Window.[v]

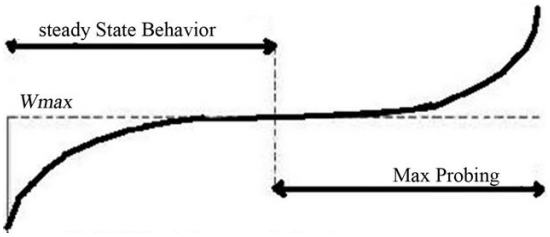


Figure 2: The window growth function of CUBIC

one called Hybrid Slow Start (or HyStart) [13]. The idea behind this algorithm is to prevent the extra drop of packets caused by the overshoot of bandwidth done by the regular slow start. This extra amount of data cause unnecessary delay on both ends hosts

[v] C is a scaling factor, t is the elapsed time from the last window reduction. W_{max} is the window size before the last window reduction. $K = sqrt[3]W_{max}beta/C$. And $beta$ is a constant multiplication decrease factor applied for window reduction at the time of loss event

and network, and taking long time to recover from, that can be worst if the packet drop is made at a late stage of the slow start[vi].

2.2.2.2 Compound TCP (CTCP) is the modification to TCP stack algorithm promoted for Microsoft and implemented in their systems since Windows Vista and Server 2008. This algorithm aims to achieve a significant improvement in resource utilization by TCP, presenting an fair mix between delay-based approach and loss-based approach.

CTCP is based in the idea that if the link is under-utilized, the protocol should increase the sending rate in a aggressively manner so it can obtain available bandwidth more quickly. But when the link is nearly full utilization, keep that aggressiveness is unproductive because at this stage it is expected that the protocol is more accurate in achieving complete the remaining bandwidth. So CTCP incorporates a scalable delay-based component into the standard TCP congestion avoidance algorithm [34].

As can be seen in figure 4, the congestion window or cwnd which manages the loss-based component is untouched, and a delay window - dwnd - is added to manage the delay-based component. The awnd is the advertised window from the receiver. This algorithm has shown that in periods of under- utilization, it has an aggressive and scalable increase rule, and gradually it reduce the sending rate accordingly when the network is sensed to be fully utilized. But thanks to the packet loss component, it also reacts to packet loss reducing the sending rate to keep the conservative property of TCP. Also, Tan et al [33] has reveal that CTCP has a good performance over current high BDP networks too.

$$win = min(cwnd + dwnd, awnd)$$

Figure 4: Compound TCP sending window.[vii]

[vi] since slow start adds one extra packet for each acknowledged

2.2.3 Fast retransmit and fast recovery

Timers are an important part of this self-regulated protocol. TCP maintains four timers[viii]. One of them, the retransmission timer, causes the sender to retransmit the segment over which at the expiry of this timer(which is a function of the estimated RTT), has not yet received their respective acknowledge, assuming the segment is lost in the network. When this action is unleashed, the server will interpret it as a sign of congestion, but data still flowing.

As already seen in **2.2.2.1**, slow start can do a lot of damage to our networks, but fast retransmit helps to recover from lost without hurting the network that much. It is triggered after a specified number of acknowledgments, usually is at the third, with the same acknowledge number are received by the sender. This makes the sender reasonably confident that the next higher sequence number was dropped and needs to be retransmitted; all this without the need for the action triggered by the timeout.

Fast recovery manages the continuous transmitting of new data on subsequent duplicate acknowledgments, until the retransmitted packet is received and after a non-duplicated acknowledge is received, the congestion window is reset back to the value it had before the fast retransmit was initiated, and the fast recovery mode is ended, going back to congestion avoidance mode.

[viii] The timers that TCP has for each connection are: (1)Persist Timer: Ensures that window size information is transmitted even if no data is transmitted. (2)Keepalive Timer: Detects crashes on the other end of the connection. (3)2MSL Timer: Measures the time that a connection has been in the TIME_WAIT state. (4)Retransmission Timer: The timer is started during a transmission. A timeout causes a retransmission

3 Router's Buffers effects

One main function of routers is to absorb bursts of traffic coming from the hosts, and ensure that links are used to their maximum capacity. A network that does not have buffers does not have the ability to store packets that are waiting to be transmitted, so any packet that is over the capacity of the link will be dropped. In order to operate without buffers, arrivals must be constant and predictable; synchronization of the entire network would be necessary to avoid any loss. Such networks are complex in design, expensive to implement and particularly restrictive.

The throughput of each network is limited by the capacity of the slowest link causing a bottleneck; no matter how much more packets are introduced to the network, the transfer will not be faster than that determined by this link. That is why the major location of buffers in areas where bottlenecks occur is essential for proper functioning of the network, but this transition between fast-to-slow networks is different on each route, and even for the reverse path.

The presence of buffers is necessary to help reduce data loss. In the past the high cost of memory kept the buffers quite small, causing them to fill pretty quickly after the network became saturated, signaling the presence of congestion and thus the need for compensating adjustments.

Unfortunately the network transport protocols, in order to operate at full capacity, require that the hosts are notified in a timely manner when they should back up and thus able to adapt its transmission rate to the available capacity. The absence of such timely notification triggers the presence of full buffers and increased communication latency.

3.1 Full Buffers

Packet networks require to absorb bursty arrivals. When received, a packet is immediately validated, but not necessarily immediately processed and transferred out without having spent some time in a buffer. That means that if the rate at which packets are received is greater than the delay to process and remove a package from the buffer, the buffer fills up[ix] and stays congested, contributing to excessive traffic delay and losing the ability to perform their intended function of absorbing bursts.

This standing queue is the essence of Bufferbloat and is the result of the difference between the window and the bandwidth of the link, only creating long delays in communication, but no benefit in overall throughput. *It is not a phenomenon treated by queue or traffic theory, which, unfortunately, results in it being almost universally missclassified as congestion (a completely different and much rarer pathology)*[26].

When the packet reaches a bottleneck queue, it is squeezed down in bandwidth but must stretch out in time since its size stays constant. This stretch is the cause, at the buffer level, of the delay of the next packet in the queue.

3.2 High Latency

The latency a packet experiences in a network is made up of transmission delay (the time it takes to send it across communications links), processing delay (the time each network element spends handling the packet) and queuing delay (the time spent waiting to be processed or retransmitted). But large buffers only increase latency, and this only causes conflict with the needs of nowadays applications.

Once packets in-fly reach a bottleneck, they begin to pool. Because the characteristics already explained, more and more packets are coming, and this queue continues to increase,

[ix] When this occurs, the receiving device may need to adjust window size to prevent the buffer from being overloaded.

which leads that each new arriving packet spending more time in the queue than the predecessor packet, which means an increase in the latency. Eventually, packets start to be dropped, notifying the hosts of the presence of congestion on the path.

As stated in [8], it is quite common to find these high- latency queues in the last mile. The causes can be many, but among the most common are both link quality at homes, different implementations of traffic shaping methods implemented by ISPs, or massive buffers set by the latter to avoid loss of data, which can add a delay of up to several milliseconds. Bottlenecks at the Internet's edge can easily move between the wireless access (when its bandwidth is slow) and the provider's up-link, both of which can have highly variable bandwidths.

For networks that use coaxial cable, multiple clients multiplex their upward flows in a single transmission, resulting in a burst with a large volume of data, which can led to a high fluctuation in latency. This concatenation can also generate jitter time, which can be produce a missinterpretation for some protocols of incipient congestion and cause entering congestion control avoidance too early. The other type of highly used networks are the DSL, in which distance to the supplier reduces transmission rate. That is why it needs advanced signal processing and error correction algorithms which can lead to high packet propagation delays.

3.3 Sizing Router's Buffers

Unmanaged buffers are more critical today since buffers sizes are larger, delay-sensitive applications are more prevalent, and large downloads common. Overly large, unmanaged, and uncoordinated buffers create excessive delays that frustrate and baffle end users.

Correct buffer sizing is not an easy problem. Undersizing – making buffers smaller than the traditional BDP – is fraught with problems. Today's links vary in bandwidth,

and individual connections vary in RTT. This makes it impossible to properly pick a static size for most edge links. A simple, robust algorithm that can manage buffer delay regardless of buffer size and link bandwidth without a negative impact on utilization can make over-sized buffers irrelevant.

A widely used rule-of-thumb recommends the buffers of size $B = \overline{RTT} \times C$, where $\overline{RTT}$ is the average round trip time experienced by connections utilizing the link, and C is the data rate of the link. Many buffers have been inserted without sufficient care due to the complexity of establishing the appropriate size and test on real environments[40], helping to determine where further complicate occurs Bufferbloat.

The rule-of-thumb comes from the idea of maintaining the medium as busy as possible, and to maximize the throughput of the medium. But due to the characteristics of TCP, no matter how large the buffer is, it will always be saturated.

The main condition to choose the proper size of a buffer is the ability to keep sending data in the periods in which the sender is stopped, preventing possible downtimes in order to maximize utilization of the link and kept high throughput at all times. The buffer will avoid to idle if the first packet from the sender shows up at the buffer just as it hits empty.

After a lost is detected, the *cwnd* is set to half of is last value, so if we denote as $(W_{max}/2)/C$ the amount of time that packets are sent in congestion phase, and as B/C the time that takes a buffer with size B to drain, the size of a buffer B needed is $B \leq (W_{max}/2)$ [2].

The problem occurs when the required size buffers are implemented, and the size is exceeded producing overbuffering. Overbuffering hurts anytime a link is saturated causing extra delay, and destroying many uses of modern Internet. As stated in [12], *a TCP connection must react quickly to changes in demand at the bottleneck link, but TCP reaction time is quadratic to the amount of overbuffering.*

Unfortunately this error comes from the fact that manufacturers began to think that adding larger buffers to their products should bring positive consequences, especially considering the current volumes of data, in addition to providing more equitable access to available bandwidth handled.

4 Active Queue Management

Ensure communication only checking the endpoints has revealed that it is not enough and that it is also necessary to check the status of in-flight data transmission. With the passage of time and with increasingly high speed networks it has become more and more important to have mechanisms that keep throughput high but average queue size low. The changes implemented in TCP for congestion control have proven not to maximize the capacity of the medium along with a performance degradation. Among the most common problems are: connections still experience multiple packet loss, low link utilization and congestion collapse. It is important to keep in mind that when a packet is dropped before it reaches its destination, all of the resources it has consumed in transit are wasted.

This is how the role of routers extends beyond joining and interconnecting local networks and the Internet together. The role of routers becomes important because they are able to better manage congestion that may occur in networks due the knowledge they have about other routers in the network and can choose the most efficient path for the data to follow, or as already seen, send message to the ends to fall back or even drop data. They must allocate the available bandwidth fairly between all flows.

Typically the queues that routers maintain are designed to smooth and accommodate bursts of data delivered by the hosts and transient congestion, but as the queues start to fill, the routers drop packets using a FIFO based drop-tail management. This discipline has two mayor drawbacks:

1. Lock Out: a small number of flows tends to monopolize the usage of the buffer capacity[4] .

2. Full Queue: these buffers tends to be always full, leading to high latency.

Also, as explained in [10], when a router use tail-drop, *the more bursty the traffic from*

a particular connection, the more likely it is that the gateway queue will overflow when packets from that connection arrive at the gateway/router.

All this led to develop and implement more aggressive or active strategies for congestion control on the Internet called Active Queue Management or AQM. AQM is a group of FIFO based queue management mechanisms to support end-to-end congestion control in the Internet. The goals behind AQM are to reduce the average queue length and with that decreasing the end-to-end delay. Also to reduce packet losses that reflect as a more efficient resource allocation.

AQM maintains the network in a region of low delay and high throughput by dropping packets before queues become full and can reliably distinguish between propagation delay and persistent queuing delay. Also, because the router can monitor the size of the queue over time, the gateway is the appropriate agent to detect incipient congestion. The most common AQM algorithms are: RED, SRED, BLUE, SFB, CoDel.

4.1 RED

Random Early Detection or RED is the algorithm introduced in [10]. RED monitors the average queue size and when it exceeds a preset threshold, the router drops the arriving packet with a certain probability, where the exact probability is a function of the average queue size. If Explicit Congestion Notification (ECN) is active, the packets are chosen randomly and marked. When a connection uses a larger share of available bandwidth, it is more likely to be marked. The idle periods (periods when the queue is empty) are also taken into account for the calculations of the average queue size.

The objectives behind the implementation of RED are:

1. Minimize packet loss and queue delay.

2. Avoid global synchronization of sources by randomizing its marking decision and by

spacing out its marking.

3. Maintain high link utilization.

4. Remove biases against bursty sources by limiting the queue occupancy so that there is always room left in the queue to buffer transient.

Because RED can control the average queue size while accommodating transient congestion, implementing RED in routers can provide high throughput and low average delay in high-speed networks with TCP connections that have large windows. Also, in routers where RED and ECN are implemented the bursty traffic is well handled and the global synchronization between flows is avoided by decreasing their window at the same time.

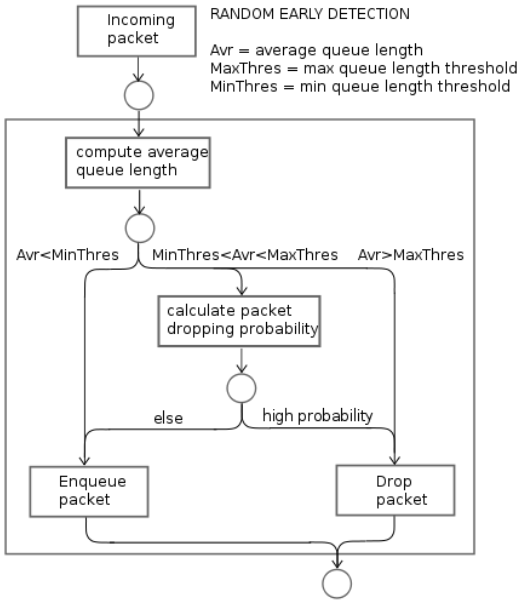


Figure 5: RED's Flowchart

The most common criticism to RED is its complexity to properly configure its parameters. It is because of this that after their official publication, it was necessary to perform a second publication [17], which aims to explain a few more details on how to configure it properly. Since RED relies on queue length as index of congestion, it can reflect the presence of congestion but not the severity nor the numbers of competing connections sharing the link. Since long ago networks ceased to have a static configuration, RED requires a wide range of parameters to operate correctly under different congestion scenarios. Unfortunately, this failure leads to deterministically mark packets and suffer

from poor link utilization. *While ECN timeouts allows RED to avoid packet loss, the deterministic marking eventually causes underutilization at the bottleneck link*[9].

As stated in [22], *RED was simple and can be effective at reducing persistent queues, little guidance was available to set its configuration parameters and it functioned poorly in a number of cases. This led to a general reluctance to use it.*

Also to archieve the ideal operating point, it can only do so when it has a sufficient amount of buffer space, which leads by default induce an extra latency.

4.2 BLUE

Introduced by Wu-chang Feng et al in [9], BLUE aims to manage congestion by using packet loss and link utilization history. The concept behind BLUE's development is to avoid drawbacks of RED like parameter tuning problems or to determine the fluctuations of actual queue length. The key idea is to perform queue management based directly on packet loss and link utilization rather than on the instantaneous or average queue lengths.

```
Upon packet loss
    if (now - last_update >freeze_t)
        Pm = pm + d1
        last_update = now
Upon link idle
    if (now - last_update >freeze_t)$
        Pm = pm - d2
        last_update = now
```

Figure 6: BLUE's algorithm

Through the use of a single probabilistic indicator called p_m, it simplifies the process of dropping or marking packets when they are enqueued. Against a continuous dropping of packets due to overflow, the probability increases, increasing the rate at which it sends back congestion notification. Also, to control how this parameter changes over time, the $freeze\_time$ is used to determine the

minimum time interval between updates. The value on which to increase or decrease, if the link is idle, is determined by δ_1 and δ_2.

The effects of BLUE, as proved in [9], are the reduction of the buffer size, which also reduces the end-to-end delay and helps to improve the effectiveness of the congestion control algorithm. BLUE has also shown that it has a better performance when marking incoming packets, leading to congestion notifications causes periods of sustained packet loss nor periods of continual underutilization.

4.3 CoDel

After several years using algorithms based on the average queue size, Nichols and Van Jacobson realized that this approach although they have information of congestion, no hard data on how serious is this. Jacobson et al, determined that it may submit two types of queues: one that smooths bursts of arriving packets turning it into a stable and continuous sent and tends to last less than one RTT (called good queues); and those that only tend to create excessive delays and lasting several RTTs (called bad queues). With this, they were able to determine that the core of Bufferbloat detection problem is separating good from bad queues. Bearing in mind the above is that an algorithm called *Control delay* or CoDel by its acronym was developed.

Presented in [22], CoDel is based on the idea that it is sufficient to maintain a threshold above which the tail can not exceed rather than keeping a window of values to compute the minimum. Rather than measuring queue size in bytes or packets, it is used the packet-sojourn time through the queue. The time each packet spends in the queue is independent of the transmission rate, the Sojourn is more valuable information about the behavior of the buffer, in addition to being much more related to the effects the user may experience. The sojourn is based on a timestamp corresponding to when the packet arrives at the

queue, at which this mark is added to the package information. The minimum packet sojourn can be decreased only when a packet is dequeued.

Another difference that occurs is that CoDel accepts the existence of queues, but it is unacceptable to drop packets when you have a buffer with less than Maximum Transmission Unit (MTU)'s bytes.

But when the queue delay has exceeded target for at least interval, a packet is dropped and a control law sets the next drop time accordingly to Figure 7. The ways in which CoDel stops packet drop is either when the delay queue is less than the target value or when the buffer contains less than MTU's bytes. Furthermore the algorithm has additional logic that controls reentering to dropping state too soon.

$$f(n) = \frac{100}{\sqrt{n}} * f(n-1)$$

$$\text{f(0)} = 1$$

Figure 7: CoDel droptime interval, n=iteration

Beside the minimum packet sojourn, the other two parameters that CoDel needs are the target and interval, which are almost self-explanatory. Target is the acceptable standing queue delay and interval works as the time on the order of a worst-case RTT of connections through the bottleneck. The recommended target value is 5ms since tests reveal that with this value, the utilization of the links is optimal. Interval is loosely related to RTT since it is chosen to give endpoints time to react without being so long that response time suffers. A setting of 100ms works well. *CoDel's efficient implementation and lack of configuration are unique features that make it suitable for managing modern packet buffers*[22].

5 Hidden Flaws and Bufferbloat

Because much of the basis for how TCP was designed in the early days of the Internet, where speeds were low, the burden on the networks were only a few kbps, and the networks interconnecting were separated by a couple milliseconds, the use was ideal. But it is not hard to see that in the past few decades, Internet growth has exceeded the expectations of even the most experienced analysts and dreamers.

Intensive-consumption applications like on-line games, audio and video streaming for music or on-line calls, torrent or P2P applications and sites that have a high level of resource utilization like Youtube, Facebook or Netflix, have made networks of today quite different from those that formerly existed, but still running the same protocols[x]. If that was not enough, the creation and incorporation of new devices such as smart phones, tablets and even smart-watches with the ability to transmit data over the Internet, makes it more necessary to handle data traffic in an efficient way.

TCP relies on timely notification to adjust its transmission rate to the available bandwidth, which is commonly signaled by packet loss. But since a couple of years, and helped by lower prices of hardware along with the thought that more is always better, the manufacturers have decided to prevent the loss of data as much as possible with the addition of larger buffers on their devices. This simple action is slowly bringing a new collapse not only TCP but data transmission in general, as can be seen in [26]

With the increase in the size of buffers, the packages spent more time "on the fly" in big buffers instead of dropped, signaling to TCP to reduce the sending rate. When finally a large amount of data is dropped, it causes TCP to sharply drop in transmission rate, freeing up bandwidth. Unfortunately, due to the size of the buffers being static, when the new TCP slow-start phase starts, again due to the lack of timely signals, TCP will

[x]While many have been updated, the basics remain the same

work with reduced performance. This problem is cyclical, resulting in exponential TCP back-off, throughput degradation and very high latencies.

But the paths on Internet are shared by multiple TCP streams so the buffers, and this back-off behavior, has a tendency to synchronize flows [2]. This causes all the flows to throttle back their transmission rates simultaneously, amplifying the effect. This decrease in latency and reduced throughput are the effects of Bufferbloat phenomenon.

As stated in [12], *long delay from Bufferbloat are frequently misattributed to insufficient bandwidth and this misinterpretation of the problem leads to the wrong solutions being proposed*. As seen in the past sections, the packet loss for TCP is not in itself a problem, conversely it is essential for its functioning. The real problem is the excessive and consecutive packet losses that occurs when the buffers kept persistently full. This effects may cause that modern applications timeout their connections or in others may experience some delay. This is the real effects and problems of Bufferbloat.

6 Experimental Work

The goal of the experiments outlined in this section is to examine different residential and public networks with the objective to prove the existence of the phenomenon under an uncontrolled scenario. This will be done by running a set of tests with different tools, first to define and characterize the network, then to measure and compare how the network behaves with and without load. More specifically, the factor to be tested is the latency under load and analyzed to identify if the latency that occurs is due to an excess of buffers or due to some other problem.

This section aims to explain the setup used to perform the tests, the tools selected for each of these tests and what is to be achieved and expected from each.

6.1 The Test Setup

The tests were ran under a pseudo controlled environment, using one physical machine and a second virtual machine hosted on the first machine. These systems runs under a regular OS without any modifications. Also, for some tests two other devices will be added, one acting as an Iperf Server and a regular Android Tablet that will be used to add some extra load to the network when the Ethernet cable is used as medium.

All of these tests will be carried out in a real-world scenario, where no packet prioritization is done by the server against our flows, the routes can vary between each iteration of the same test, and many different flows will collide with other flows from different sizes and types. Nor there is more information about how the flows are treated by the queue manager algorithms or about how they are configured.

6.1.1 Hardware Characterization

Physical Machine :

The physical machine runs as host OS, Windows 7 SP1, that works with an Intel(R) Core(TM) i7-2670QM CPU 2.20GHz with 8GB of available RAM. This machine will always be connected through its wireless adapter, a *Broadcom Corp. BCM4313 802.11b/g/n Wireless LAN Controller (rev 01)*. The Ethernet controller is a *Realtek Semiconductor Co., Ltd. RTL8111/8168 PCI Express Gigabit Ethernet controller (rev 06)* adapter, and will be bridged to the virtual machine for some tests.

Virtual Machine :

The virtual machine is hosted using VMWare Player 6.0.1, with 4 processors assigned for use plus 4GB of RAM, and with the Ethernet adapter connected only for certain tests. The OS selected is a Debian based OS called Kali Linux, and using the kernel release identified as Debian 3.12.6-2kali.

The wireless adapter is a AIR-802 USB adapter with Zydas chipset and a TP-link 8dbi antenna. This adapter is directly connected to the virtual machine and hooked to the physical machine without the USB extension, this way any extra signal loss is avoided.

Iperf Server :

This is a VPS hosted by Digital Ocean [xi] with 512MB Ram, 20GB SSD Disk, and located in New York data center. This machine runs Ubuntu 12.04.3 x64 under KVM software using as a processor an Intel Hex-Core 3 GHz.

The idea of using an external device to connect the virtual machine to the network and not using a bridged configuration provided by software, is mainly because with an USB

[xi] `https://digitalocean.com`

device the machine will take care of all the management and administration of the device, avoiding any possibility that the host machine modifies or manages any flow. Also, by using a second machine to overload the uplink avoids overflow the queue on the machine that is performing the tests. With this, it is expected to minimize the possibility that our testing machine is causing extra latency, either by the saturation of the wireless channel, or by the queue in the traffic control subsystem into the kernel.

For the tests, the Bufferbloat community[37] has created a set of best practices[30] to follow so the results are consistent and repeatable, but in this case, computers and routers will not be modified as it will attempt to analyze what an every-day-user experience. Those practices will be taken in consideration if it is the QoS present in routers and deactivate as recommended.

6.2 Tools Definition

The tools used for the benchmark were selected by the capability to determine the presence of the Bufferbloat phenomenon in the network of study. So, with this as a main consideration, the tools selected for the benchmark will be chosen by the complexity and accuracy to measure and determine the RTT into an IP/TCP connection, and how hard is to consistently replicate the results under similar contexts. Other tools were selected to help determine, characterize, and prove the capability of the network to cause Bufferbloat.

The tools selected to be used in this tests are the following:

- Speedtest by Ookla
- Netalyzr by ICSI
- Iperf Tool with Tcptrace/Xplot.org
- Page Benchmarker extension for Google Chrome
- Smokeping Latency Tool

6.2.1 Speedtest

Developed by Ookla[xii], this tool is used for most of the ISPs and many users in Chile to test their broadband's connections globally. It can be used not only in their website *www.speedtest.net* but also on Android, iOS or Windows Phone. A command line interface developed in Python can be used too for testing Internet bandwidth.

The server selected to perform every test, either has or not the fastest ping, is the one hosted by the Pontificia Universidad Católica de Valparaíso (this host is the default selected most of times by the site also).

6.2.2 Netalyzer

For end-users, little is revealed about how ISPs manage their networks. That is why since 2009 ICSI has developed Netalyzr, a "two click " tool developed to test networks that runs in web browser as a Java applet. Once downloaded, the applet contacts the back-end server using a range of protocols and mechanisms employed as part of the testing, and then conducts a series of tests. Once completed, it uploads its findings to the back-end where they are distilled into a detailed report breaking down the findings into correctly

[xii] `https://www.ookla.com/about`

operating aspects, those that show potential signs of trouble, and those that are downright broken. Figure 8 is an example of the output.

The primary goal in developing Netalyzr's tests was to provide a new kind of diagnostic tool, *one that particularly illuminates under what sort of restrictions a user's Internet connection operates, like both forms of filtering (blocking) and proxying imposed by the user's ISP, and performance issues that arise from the nature of the user's Internet access setup*[19]. Among the performance considerations, Netalyzr's measures are packet loss, latency, bandwidth. Also, it computes in-path forwarding device buffer size by comparing small-packet latencies under idle and loaded states in networks (the perfect time for Bufferbloat to happen). Other tests like general TCP and UDP service reachability are performed.

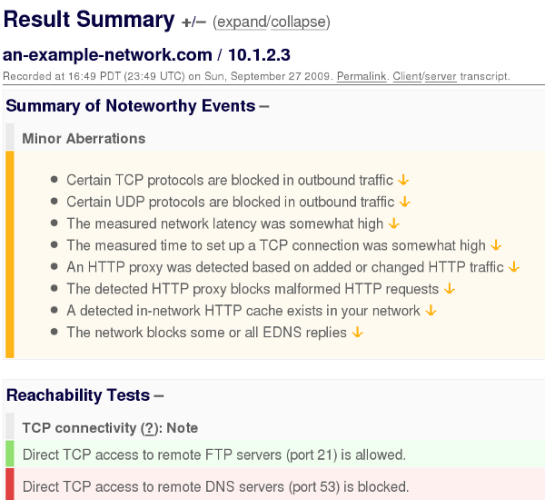


Figure 8: Result Summary example from Netalyzr. Source: `www.dslreports.com`

6.2.3 Iperf

Iperf is a well known and commonly used network testing tool. It can create TCP and UDP data streams and measure the bandwidth and the quality of a network link. It can perform multiple tests like Latency, Jitter or Datagram Loss.

Iperf basically tries to send as much information down a connection as quickly as possible reporting on the throughput achieved. This tool is especially useful in determining

the volume of data that links between two machines can supply. This two machines define the network, one acting like a server and the second as the client. For this scenario, the server will be the VPS that only will receive the Iperf connections (also will be running ssh but without further interaction). The VM Linux machine will work as an Iperf Client.

As mentioned in Iperf users mailing list "*When one runs TCP tests, there are 2 things that block Iperf from having clear view of real throughput: buffering on sender's side (TCP/IP stack) and TCP behavior itself (acking). What Iperf can measure is the pace with which it sends data to TCP/IP stack; TCP/IP stack will only accept data from application when buffers are not full. If the buffer is huge, Iperf will see high throughput initially, then it will drop. If there's congestion or retrasmission going on, Iperf will see it as lower throughput*"[18], but the data generated by Iperf won't be further analyzed because the idea behind using this tool is a TCP's packet generator. This means that the packets generated by Iperf will captured and analyzed with tcpdump, tcptrace and xplot.org.

6.2.4 Page Benchmarker

Page Benchmarker is a Google Chrome extension that tests page load time performance within Chrome. It measures time-to-first-paint, overall page load time, KB read/written, and several other metrics, and with its capability to clear the cache and existing connections between each page load, makes this tool one of the main sources of income to analysis.

6.2.5 Smokeping

SmokePing is a latency logging and graphing tool that consists of a running daemon which organizes the latency measurements and a CGI which presents the graphs. SmokePing gives us the ability to measure latency and packet loss in the current network, and with

RRDtool, it is capable to maintain a long term data store and to draw different graphs with the giving up-to-the-minute information on the state of each network connection.

Smokeping can be configured to perform a wide range of latency measurement probes each one directed to an independent target or over a set of targets selected for each proof.

6.3 Test Description

To test the existence of the Bufferbloat phenomenon, five tests are conducted as described below. Each test will be repeated under the following contexts:

1. Twice on the same day in one network to determine if does exist a considerable variance in latency for different times in a day.
2. Select different public and private networks with different *"speeds"*.
3. Use the Ethernet cable in order to compare the results with those previously obtained using Wireless.

Because it is intended to prove the existence of the phenomenon under circumstances experienced by everyday users, no kernel parameters will be amended nor changed, and only the ability to perform QOS on routers that have this feature is disabled. Furthermore, the overall question these tests seek to answer is the following:

Theorem. *"The networks that we use every day, have the necessary characteristics to generate the Bufferbloat phenomenon whether under low loads and if does exists, the how serious are the effects ?"*

6.3.1 Speed test

The idea under this test, is to set a baseline by comparing the speed offered by the ISP and the one at the moment of testing. To find the speed provided, the tool used is the

Speedtest in the web site of Ookla.

The benefits of using this web site is that not only it will reveal only the available national uplink and downlink, also it will set the baseline for the ping. The expected pings, independently of the connection bandwidth, should be around $\sim 12ms$, based on the data presented in [39] and [36] by the two most used ISP in Chile[xiii].

As data, it is estimated that the ratio of the average speed as compared to the rated speed of the service offered for domestic bonds has a variation to 85%, while for international links decreases drastically to 35%[27]. Due this factor, none of the tests will be performed on servers located outside our country, as it is believed that the data will not very representative regarding actual service.

6.3.2 Signs of trouble

After characterize our network based on the speed, it is needed to try to define the state of the service based on the quality in which our network operates, in example: what kind of traffic is passing through our network, and also get a little more detail on the average rates of delay and buffers with which the traffic can find along the path. To collect all this information will be used Netalyzr.

While laws in Chile assure consumers networks free of traffic shaping barriers and filters, ISPs apply traffic management measures[35][38] to deliver an optimum experience of the service's use and thus, achieve an efficient use of the network, thus further to protect the safety of users while maintaining network stability.

This is why it is expected that some ports or services are partially or completely blocked, but the most important is that this test will first light on the existence of the phenomenon under study.

[xiii] Telefónica has a 39.2% market share while VTR owns 38.8% respectively. [29]

6.3.3 Collapse test

Already having a clear idea of the state of the studied network, the next will be try to check empirically that the results obtained by the previous test above described are valid. For this, Iperf will be used to generate as much traffic as possible for 5 minutes between servers. But the main goal is not to measure the throughput of the network; instead to capture the packets that Iperf produces and study them. So before run Iperf, tcpdump will capture this information. Subsequently, the tcp's trace is taken and extracted with tcptrace tool and generate the RTT graph.

This exercise will be conducted three times as part of the test, running the first time as the sole source of network load. For the second and third time, after 50 seconds after Iperf was started, from the Windows machine will be performed an upload to a Dropbox account with a file big enough to the upload take more than the time defined to this test. This upload intends to saturate the upstream link, leading to an overload of existing buffer (indistinct whether the server is national or not, there are several levels where the route is common). The upload will be stopped 50 seconds before the time limit.

The time gap established before loading and the end of the test is to let Iperf frames reach a steady state flow and thus to generate a basis on which to analyze the time when the traffic is maximum.

The expected RTT are around $\sim 12ms$ and $\sim 100ms$ without load, for national and international servers respectively. About the expected value under load, it is hard to estimate, further that their expected behavior is to be stable round a certain value. In case of no stable behavior, the possible causes will be analyzed and tracked.

6.3.4 Load benchmark test

With the effects of excess buffers already determined, this tests seeks to show how this phenomenon affects users who surfs through a web browser. This is why the extension of Google Chrome Page Benchmarker comes in and it will be used to do 5 iterations of 10 loads to a website and analyze the behavior of these.

As in the previous experiment, the first will be without any load on the network. Then, again a file will be uploaded to Dropbox from the second machine and after 30 seconds from the start of the load, and with the load active, will proceed to a second iteration. The third will begin the after a minute, canceling the file upload and after waiting 30 seconds after cancel the upload, the benchmark will be measured again. The fourth and fifth will be a minute and a half of a minute since the previous iteration have finished. The site chosen is `http://www.usm.cl`.

As like Jim Gettys showed in his video demonstration[11], the benchmark increases between 10 to 15 times with load against the original times. For this test, the expected proportions are lower, this mainly because the kernel machine that will run the tests has implemented already some improvements to mitigate this problem.

6.3.5 Smoke the path

To verify that the effect of Bufferbloat, if it is the case, does not just happen on a single server or with a single TCP flow, the Smokeping tool will be configured to perform two types of tests: Fping and echopinghttp. These tests will be directed to different types of servers (i.e., physical and VPS machines) which have different connection settings and/or type of services (some servers have dedicated links), and located in both Chile and the United States.

The tool will be left a couple of minutes to track and save the state of the connection

under no extra load (only the test machine will be generating data). After that, a upload of a big file will be performed from a second machine until it finish or the behavior has become stable and then re-enter to a phase without load for couple of more minutes and repeat the upload.

With this test, in addition to verifying the validity of the previously captured data is expected to determine the presence of other factors that can contribute to the degradation of the quality of service on the network, whether factors such as quality of service provided by the host, channel/path problems, problems related to the way handling packets by the router or the machine, and/or any other that may arise.

7 Results

This section shows and summarizes the results obtained in the different tests detailing each specific case, and then collect and summarize all.

7.1 Speed test

As expected, in all cases the networks were asymmetric, the download bandwidth (the downlink) being significantly higher than the uploading or uplink. This also explains the relationship between the uplink and ping, presenting a major ping in the networks with lower uplink. The relationship can be better be seen in the Figure 9.

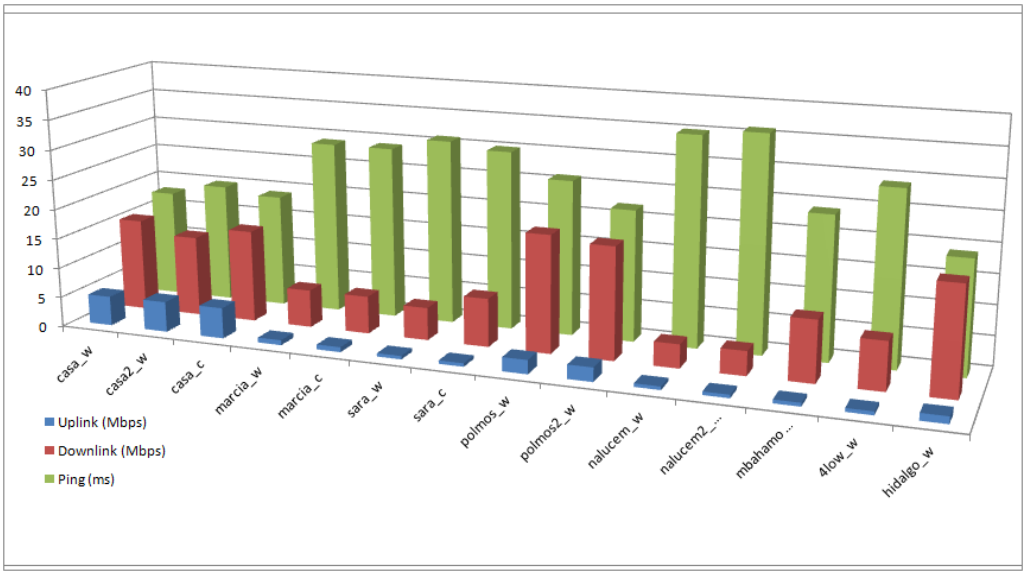


Figure 9: Speed and Pings based on Appendix A, Table 7

The Figure 9 also is a proof of the common misunderstanding about the how the ISP's promote high download speeds but with low uplink. Even though most of the time home users download content, in order to keep a good ratio the data needs to flow both ways, due to the way TCP works, you need good communication in both directions. While in

theory this works without major problems, unfortunately in reality users download content from different sources at once, and if we add new requirements like real-time or on-demand applications and online games, we see that not only a high-speed downlink is important.

Another important point is the growing need for higher bandwidth. Unfortunately it is common to associate bandwidth with speed. Instead, bandwidth is a measure of capacity, not how fast the network responds. So, more capacity is only better if the existing latency is low. This may explain why the tests performed in network tagged as *"casa"* can have low pings even when its capacity is close to the mean, but this network is through fiber[xiv].

	Ratio (%)	
Location	Uplink	Downlink
casa_w	99,00	103,20
casa2_w	101,80	89,73
casa_c	103,00	102,93
marcia_w	166,00	63,00
marcia_c	166,00	62,90
sara_w	112,00	54,50
sara_c	114,00	80,80
polmos_w	116,00	49,03
polmos2_w	117,00	46,80
nalucem_w	114,00	99,25
nalucem2_w	112,00	101,50
mbahamon_w	120,00	102,10
4low_w	120,00	80,30
hidalgo_w	119,00	89,80

Table 1: Variation ratio of offered vs measured speed.

The $\sim 12ms$ ping were also not obtained, but the values are still within the acceptable range around the $\sim 26ms$. These values may be due to factors such as congestion, conditions on the server, or the processing time information. The variation of the ratio between the measured and the bandwidth contracted is of 80%, which is only 6% less than

[xiv] In practice, fiber service have a propagation delay 10 times faster than ASDL service

the spected value[xv].

Further analysis is needed to determinate how the bandwidth is related with the Bufferbloat, but it is clear that less the bandwidth, higher the ping, but it can not be stated the opposite, higher the bandwidth, less the ping because it also depends on how fast the network is and the effects on the latency [5].

7.2 Netalyzr test

After having a clear idea about the capacity of the network, it is necessary to know how it speed and behavior works. This requires analyzing how latency behaves. As stated, latency is the time it takes a message to travel from one computer to a sever, and has a huge impact on how the user experiences the network.

From all the information that Netalyzr provides, only the most relevant information to determining whether Bufferbloat is present on our networks is shown. For this, the data that will be take into consideration is:

DNS resolver Time: This test measures how quickly the DNS resolver is able to resolve the mnemonic name. Slow resolvers may make the network seem "slow" even if the network itself is fast.

Network Buffers: The most revealing fact of the existence of the phenomenon studied. While it is a fact that is commonly overlooked, also it is crucial to the quality of the network. If the buffer is too small, network protocols such as TCP are unable to send as fast as the network allows. If the buffer is too large, a single transfer will fill up the buffer, delaying all other traffic.

Network Performance: To determine the performance of the network, Netalyzr measures latency by sending a series of small messages to the server and then seeing how

[xv] Details of the provider and bandwidths can be found on Appendix A, Table 8

long it takes for the messages to return. Since the communication is from outside US, the latency should naturally be higher, but how much higher?

The results in Table 2 are much more clarifying than those obtained in the previous test. In the first data set related with DNS, the times are relatively stable and the time that took to resolve the requests were almost indiscernible. While it is common to find cases with shorter at $\sim 15ms$ for local name resolution[xvi], the average resolution time is acceptable within the metrics for international queries. A special case is presented on the network tagged as *"4Low"* were the time taken to effectively resolve queries to any website is extremely high but after the resolution finishes, the network had no major problems. Also no mayor relation can be seen between the bandwidth and the DNS resolve time.

		Buffer		Performance	
Location	DNS (ms)	Uplink (ms)	Downlink (ms)	Latency (ms)	Loss (%)
casa_w	190	290	190	140	00,00
casa2_w	180	280	180	140	00,00
casa_c	180	280	190	140	00,00
marcia_w	190	1200	860	190	0,50
marcia_c	180	990	2100	190	00,00
sara_w	220	5100	470	160	4,00
sara_c	200	5100	459	160	4,00
polmos_w	220	360	160	180	00,00
polmos2_w	180	370	160	190	00,00
nalucem_w	210	5100	1800	160	1,50
nalucem2_w	210	5100	1800	210	0,20
mbahamon_w	200	2900	290	200	1,50
4low_w	1300	2900	590	180	0,50
hidalgo_w	180	260	100	200	1,50

Table 2: DNS resolution time, Buffer time and Performance

The next four lines are crucial to prove existence of the phenomenon. The buffer

[xvi] As example can be a site hosted in the same country and tested with dig - `http://linux.die.net/man/1/dig`

section can reveal how much time a packet spent in the existing buffers along the way or, in other words, into the link between the network and the server. Unfortunately, in the uplink times are about $\sim 300ms$, which according by ICSI are times that in some cases may present a degraded performance (as in online games or real time conference). For the downlink, the networks that already are marked with high buffering time in the uplink are the same for the opposite route. While the high measured times in the uplink could it be justified by the diminished capacity to put new data in the network against the capacity related with the downlink (related with the asymmetric bandwidth capacity) there are cases in which the time is excessive and Netalyzr alerts the presence of excessive buffers possibly generated by Bufferbloat.

	Bandwidth (Mbps)	
Location	Uplink	Downlink
casa_w	5,00	14,00
casa2_w	5,00	14,00
casa_c	5,00	15,00
marcia_w	0,47	1,50
marcia_c	0,54	6,20
sara_w	0,57	6,30
sara_c	0,57	6,30
polmos_w	2,10	9,60
polmos2_w	2,10	9,60
nalucem_w	0,57	4,00
nalucem2_w	0,57	4,00
mbahamon_w	0,54	7,80
4low_w	0,54	6,60
hidalgo_w	1,00	11,00

Table 3: Bandwidths measured with Netalyzr

The main characteristic of Bufferbloat is the high latency, with some effect on the packets loss, since packets spends most of the time in buffers along the route. Netalyzr actually reveals that in at least two networks, *marcia* and *sara*, this phenomenon occurs

because although the latency is low under normal conditions there is packet loss and buffer times higher than normal. The packet loss can also be due to the conditions of the experiments, and because due its characteristics, it is more common higher packet lost using wireless but for these experiments there were no obstacles between the router and computer located less than a meter from each other. Also it is important to remember that the way of buffering time and latency are calculated is through two different experiments with low and high load on the network.

Among the characteristics of Netalyzr is the capacity to produce a summary table showing the different problems that arise in the networks and the severity thereof. In Table 4 presents the summary after all networks.

This summary shows that the networks actually have problems with their buffers, specifically troubled networks are: *marcia, sara, nalucem, mbahamon* y *4low*. Coincidentally, lower bandwidth networks are the same with larger buffers. In addition, as mentioned, the network *4low* also has serious issues with name resolution.

7.3 Iperf test

Since the networks are already characterized and with enough knowledge to define which networks the phenomenon studied is mostly likely to occur theoretically, the following are the results of the first part of the practical experimentation.

Iperf revealed that indeed, overloading a network average RTT time in direct proportion to the load in-fly (relation with more data to negotiate, more information over the network, higher RTTs), with which to demand it even more, like including other connections can reach to the collapse making even to load a basic website will become a task that takes a couple of minutes.

The following graphs show results in three networks characterized in ascending seri-

Panel A: Summary or errors

Location	Alerts
casa_w	1,2,8
casa2_w	1,2,8
casa_c	1,2,8
marcia_w	1, 3, 6, 9
marcia_c	1, 3, 6, 8, 9
sara_w	1, 2, 4, 5, 6, 8
sara_c	1, 2, 4, 6, 8
polmos_w	1,8
polmos2_w	1, 3, 6, 8
nalucem_w	1, 2, 6, 8
nalucem2_w	1, 2, 6, 8, 10
mbahamon_w	1, 3, 6, 8, 10
4low_w	A, 1, 3, 6, 7, 8
hidalgo_w	1, 8

Panel B: Description of errors

Alerts	
A	ISP's DNS is slow to lookup names
1	Certain TCP protocols are bloqued in outbound traffic
2	The network does not reply when it needs fragmented traffic
3	Fragmented UDP traffic if bloqued
4	The packet loss was somewhat high
5	The time to set up a TCP connection was somewhat high
6	Network packet buffering is excessive
7	DNS resolver may have probelms with DNSSEC
8	Only some root servers returned DNSSEC information
9	Not all DNS types were correctly processed
10	The network indicated bursts of packet loss

Table 4: Summary of errors and warnings in Netalyzr

ousness of the problem. Each figure is the result of three iterations performed in each network, without load and the two iterations performed with extra load.

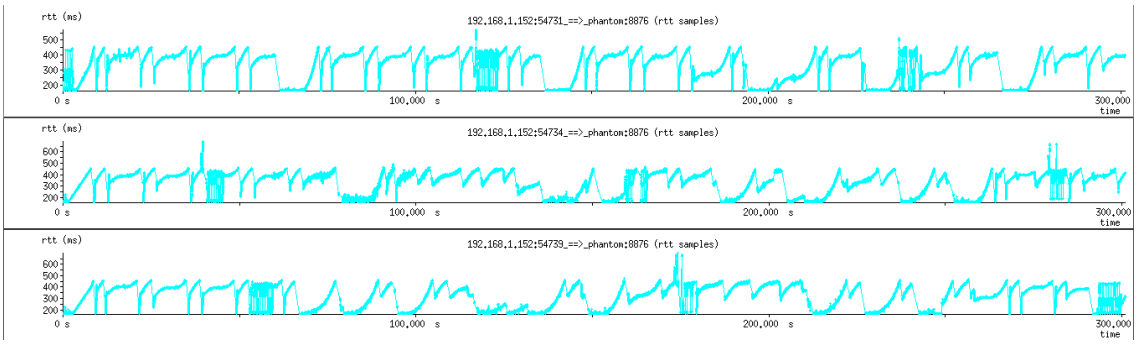


Figure 10: RTT graphs for the three test in a fiber network

Figure 10 shows the behavior of the network which, although it has a bandwidth in the mid-high average range, but with its transmission rate higher because it is a fiber optic network (*casa_w*). It can be appreciated that in a state where only one application uses the entire bandwidth, there is an increased RTT time, but not significant. In fact the only difference compared to the following two graphs sharing network resources applications is that there are some higher peaks, but these are not constant. Nor is there a greater retransmission (denoted by the thinner lines). So, with at least two data sources generating traffic, there is no significant increase in RTT time and should not imply that the increase in the loading time for a browsing client, or some type of problem seen in real time applications.

For Figure 11, at the beginning of the fist test (first $\sim$ 10 seconds) the RTT times are really small, but according to previous cases, is about the time it takes to saturate the buffers. This behavior proves that is what is occurring, specially that after this period of time RTT rise almost doubling, presenting ranges between 400 and 700 ms, but with valleys that are round to 300 ms or less. For the second and third graph in this network, the valleys are reduced and with higher values, especially in periods when the second application is

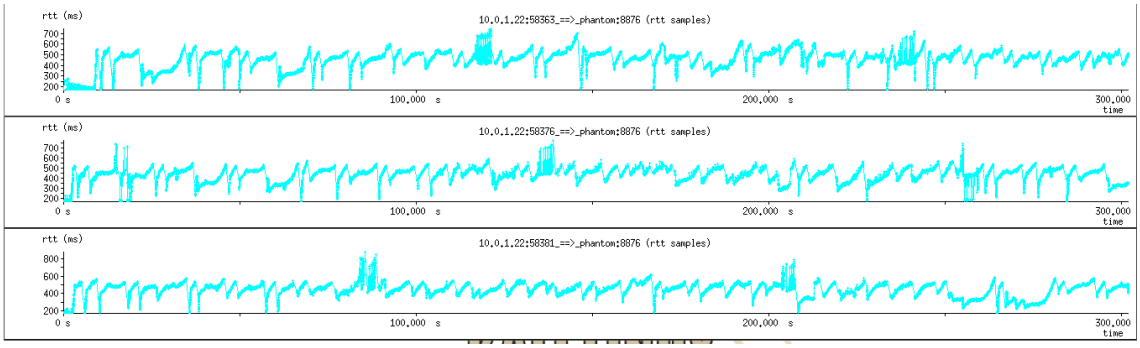


Figure 11: RTT graphs for the three tests in a network with minor issues

making use of the resource (between about 50 and 250 seconds), being noticed particularly few times in the third chart which also shows an increase in the maximum upper 800 ms. This network corresponds to *polmos_w*.

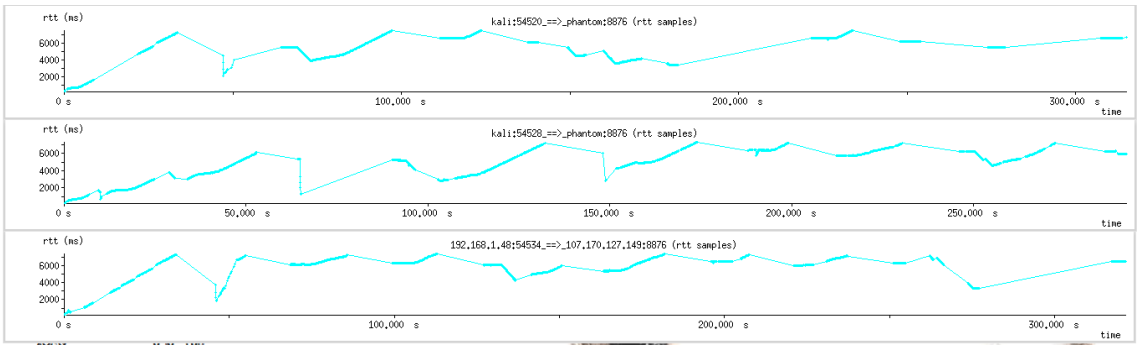


Figure 12: RTT graphs for the three tests in a network with bad performance

In Figure 12 the things got to the extreme. Even for the base case, where the times should be low, for this scenario the peaks are around 6 seconds, with a always upward trend. Also it is a constant presence of long periods where no samples are among the packets in transit which are not retransmitted packets (thinner lines). Current websites delayed an average of 2 seconds to load the full site, finding in the range between 1 and 7 seconds, for the full negotiation; but here, only one package is taken almost the same that takes a full site to load.

At first glance it seems that the case two is better than the base case, but it can be noticed that the relay times (may be caused by loss), are up most of time, perhaps they produced a massive drop of packets (approximately the second 70), so having a valley at a point less than in the base case. The third graph shows no far difference with what already found only proving the fact that there are some serious problems in the network that makes RTTs times increase to three or four times the normal behavior. With this times, it is almost impossible to maintain a steady stream of data with one application, so the use of real-time applications such as online games or web conference while web surfing would be almost impossible, and must sacrifice any.

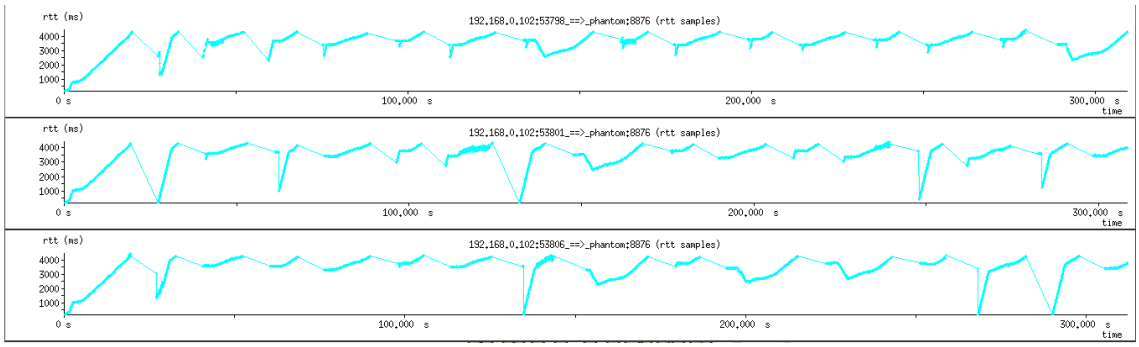


Figure 13: RTT graphs for the three tests in a network with DNS issues

It is interesting that for the network *4low*, which presents problems with DNS resolution time was needed more than twice tries in order to perform this tests, as the website used to saturate as a second stream of data could not be loaded throwing error connection time (timeout). After the website completes its full load, having to load it before start iperf for the two cases, it could continue with the normal course of the test.

7.4 Benchmark test

While ordinary Internet users request mainly optimized web sites to make the most of each user connection, it is natural that the traffic generated by several requests of web

sites simultaneously generates some kind of significant load to the network. Or so it is what one should naturally think.

Unfortunately with the results obtained with this Page Benchmark it is clear it is not so. By analyzing Figure 14, the networks that already were categorized as *with problems*, follow that trend and even with all the available bandwidth, take longer to complete the request. As example, the network *marcia_w* without other sources of load took a little over 9 seconds to complete the site request [xvii]. This results is way above expectations since under normal conditions, the normal average is between 1-7 seconds with a tendency to be close to 2 seconds). An important study case is the network *mbahamon_w* which was one of the lowest overall in cases without overloading the network. While on a later date after these tests, it was necessary to change the router used as access point, which generated a lot of noise and interference for testing between tests seeing themselves in a bad state at that time.

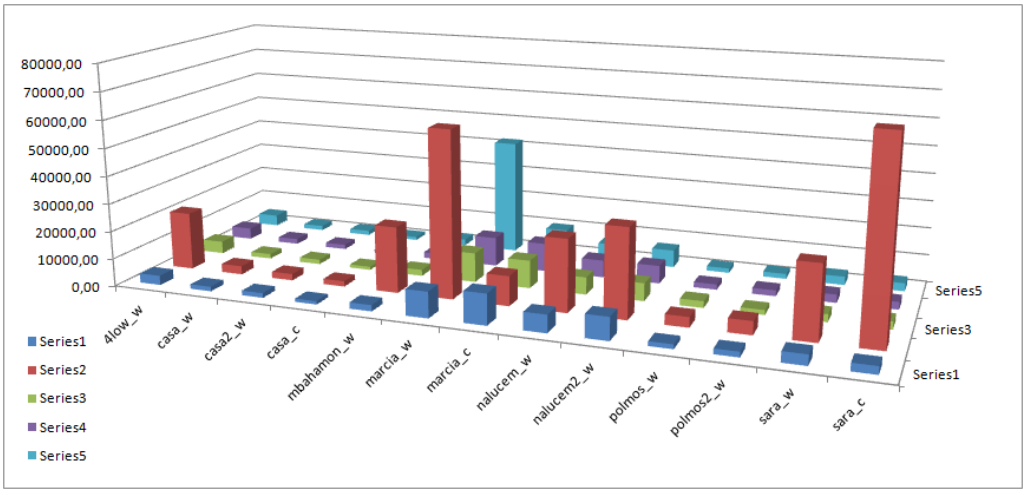


Figure 14: Total Load means (in ms) based on B, Table 9

Before realizing that the router was the flaw in the network *mbahamon_w*, under con-

[xvii] Again, the website is `http://www.usm.cl`

ditions of stress, this network was one of the most affected networks delaying average time 23 seconds. The networks that are in worst situation are those mentioned *marcia_w* and *sara_c*. Due to the presence of large buffers which store packets of uncontrolled manner and for longer periods, when the network's load decreases, this resource is still under utilization. This is reflected in the observations from from 3 onwards.

Name	min	max	ratio
4low_w	3415,1	20932,3	612,93374
casa_w	1654,8	3053,6	184,52985
casa2_w	1646,6	2354,4	142,98555
casa_c	1165,3	1944,1	166,83258
mbahamon_w	2182,6	23868,5	1093,58105
marcia_w	9401,6	60119,0	639,45499
marcia_c	10028,3	11140,0	111,08563
nalucem_w	6280,8	26048,6	414,73379
nalucem2_w	6349,4	32151,5	506,37068
polmos_w	1623,4	3701,4	228,00296
polmos2_w	1738,5	4861,8	279,65487
sara_w	2808,6	26561,5	945,72029
sara_c	2757,6	70423,3	2553,78953

Table 5: Minimum, Maximum and variation ratio.

In Table 5, shows the comparison between the minimum and maximum average load value across over all iterations on the same network and the ratio between these values. While the ratio can not be compared directly between two different networks, it does compare how many times the maximum value was respect to its original value [xviii]. Here, the network *marcia_c*, and in the Table 9, accomplish times that were close to 11 seconds for all five iterations. With this values, the ratio is low (11%) but the real values are relatively high for normal conditions (around 11 seconds).

Again, the networks *marcia_w*, both cases for *sara*, stand here because the high level

[xviii] a network may have a low ratio with times that have a low variance and high average time

of variation around 6 to 20 times the lowest average time; highlighting *mbahamon_w* and *sara_c*. Contrary to what was expected, in the case of the wired iteration over sara, *sara_c*, the maximum variation was significantly higher in this iterations than for those observed using wireless technology.

	Ratio (%)				
Name	1	2	3	4	5
4low_w	155,55556	707,62342	196,64754	232,87492	3302,32392
casa_w	108,35366	299,26429	116,08775	356,49786	118,85296
casa2_w	113,99878	281,65249	127,18041	103,10219	117,71533
casa_c	112,57589	304,73684	103,63636	103,57766	104,58478
mbahamon_w	163,95912	183,86356	153,77847	139,55813	319,52128
marcia_w	309,24808	6298,01469	267,57064	198,10685	205,48759
marcia_c	310,36182	187,34044	146,11202	205,08497	138,25129
nalucem_w	261,95410	254,73473	109,85803	106,96041	105,23376
nalucem2_w	980,66263	315,95548	254,12946	284,86443	110,70346
polmos_w	106,36537	263,10680	3902,47219	8443,23607	4150,06459
polmos2_w	159,75460	1991,19249	8206,53009	417,45121	8226,29199
sara_w	402,38095	240,75420	394,93299	976,89312	395,52042
sara_c	111,04952	281,07638	109,03226	107,01366	109,54898

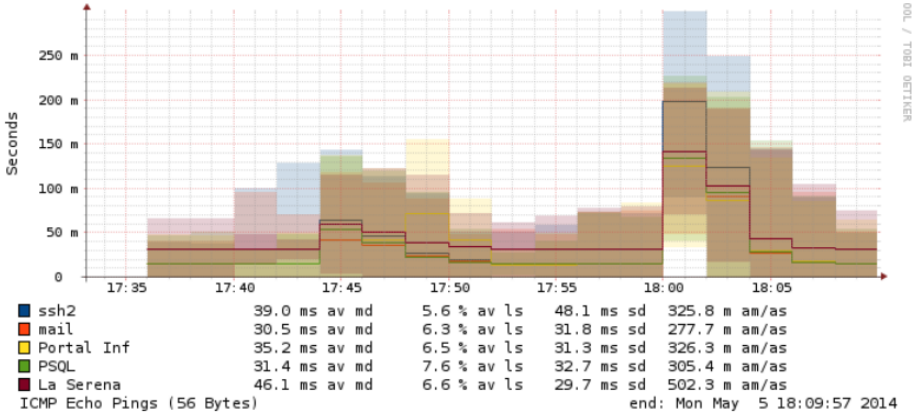
Table 6: Ratio over own iteration.

Also, for some cases adding another source was not decisive to obtain quite high times. As can be seen in 6, for *polmos_w*, which in its latest iteration had a minimum of 127.343 ms [xix] and a maximum of 1548ms. Grounds for these outlier measurements may be due to many factors ranging from the connection setup to server problems.

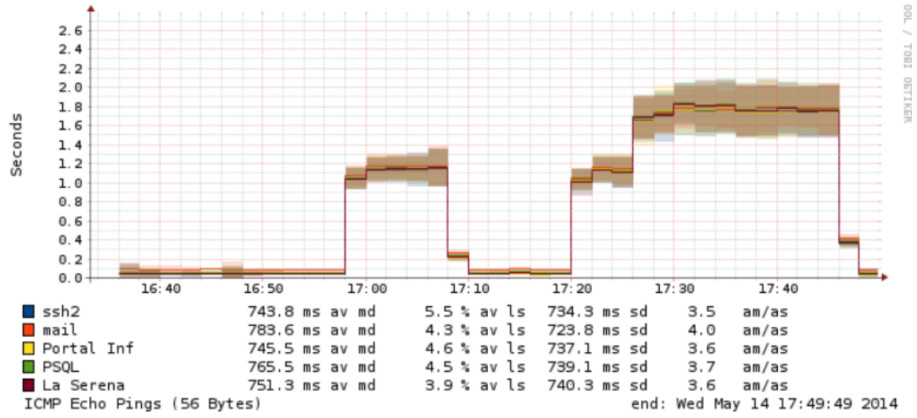
7.5 Smokeping Test

It is already clearly established the presence of Bufferbloat at least three networks, resulting in a decrease in the performance and usability of the networks, resulting in catastrophic

[xix] The detailed values are available with CD



(a) Good performance Network



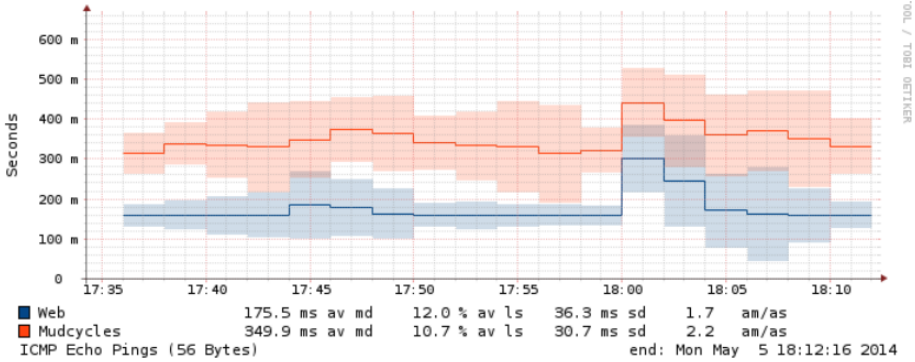
(b) Bad performance Network

Figure 15: Ping test to National servers

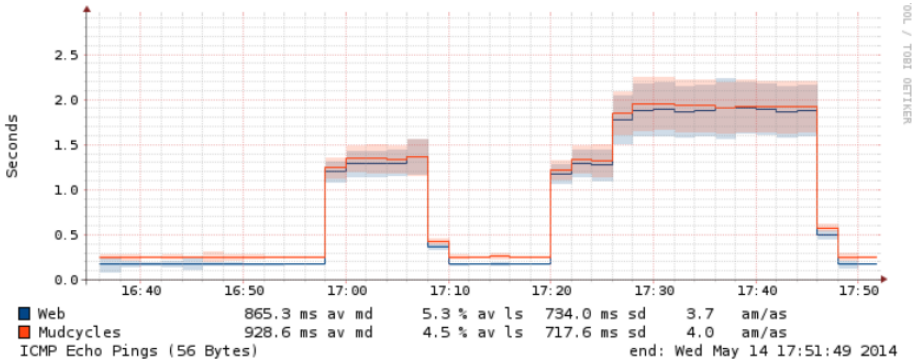
times of answer for anyone who wants to have a smooth session with any other service that requires capacity of the network.

Trying to assimilate otherwise these results is that in Figure 15 can see the contrast of the two opposite poles.

Figure 15a shows the behavior of the network *casa_w* which has faster response times with no load around 50ms and to superor 200ms with maximum use of the network. While can be appreciate greater amount of *"smoke"* (bar with same color as the line), this is because by the time range that is driving (very low), tends generates this small variation.



(a) Good performance network



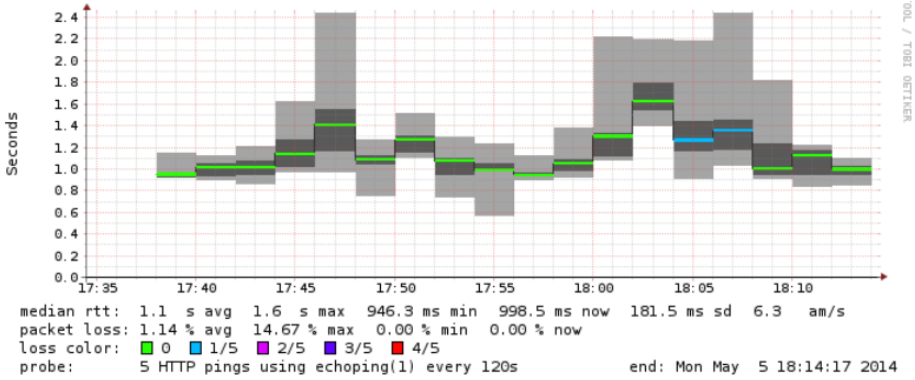
(b) Bad performance network

Figure 16: Ping test to International servers

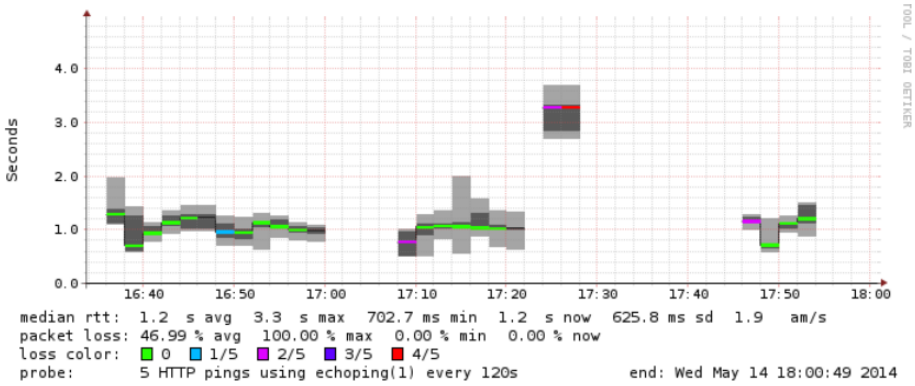
In contrast, in Figure 15b, the amount of smoke is much lower, but the variation in time is much higher, resulting in lower time to 100ms before loading, then vary drastically reaching over two seconds. While this case was less packet loss, test took almost the double because of the tendency to vary dramatically in the second case.

For international servers, it can be appreciated that the tendency remains very strong. The times related to Figure 16b increases to 8 times its minimum value, ranging from 250ms to about 2 seconds. Again, the loss in Figure 16a is almost double than in Figure 16b, but the latter spent a longer period without load, which reflect that in average have lower loss.

It is also interesting to note that while both servers are in the U.S., the requests to



(a) Good performance network



(b) Bad performance network

Figure 17: Web requests to National servers

the server noticed by the orange line took almost twice the time against the blue-line-host (above 300ms). This may be due to the characteristics of the service provided or the resolution time by the server because after a little more investigation, it was determined that was hosted on a VPS with characteristics similar to Iperf's server connection used (same hosting service).

To conclude, Figure 17 reveal the results of the HTTP requests. In the case of Figure 17a, load average achieved was 1.1 seconds with a loss of approximately 1.14%. In contrast, in Figure 17b, right after the first overload (close at 17:00 hrs) of the network, communication is affected, only been able to resume the communication upon completion

of the test. In the second iteration, becomes not only intermittent communication, but also when it generate some traffic, it takes about 3 seconds and has a loss of about 70%. This not only affirms our theory about the behavior and relationship to loss over time that lasted the test, but also shows how terrible it would be for anyone trying to load any website.

7.6 Summarising Results

The results are synthesized in order to analyze them comprehensively. The first test gave us the data to define the bandwidth of each network with which subsequently went to work. Overall the upload link (uplink) behaved as expected, performing in some cases gains of up to 50% or more than contracted. Obviously this could be due to generally low values ($\sim 1Mbps$) are therefore at a small variance is expressed as a high variation ratio. For networks with greater uplink, between 1 and 5 Mbps also showed good performance. On the other hand, the downlink reveal more variation between the contracted and obtained. The circumstantial differences between a network with good behavior to those with certain problems got reflected at this point. A special case was the network *polmos*. Strangelym after being tested with two different Operating Systems (but on the same computer) results near the contracted value were never obtained; not with other devices, where exhibit values around the 40Mbps. The range values for the ping was not as expected, but they still accepted values without being detrimental to smooth navigation with a use that does not cause an excessive stress to the system.

With Netalyzr it is possible to obtain a clearer idea of the fundamentals and the inner workings of networks. Also most revealing information was obtained from the theoretical perspective, the existence of Bufferbloat obtaining approximate times of the duration of the buffers present in these networks. This results was the most important for the continuity of

the study, mainly because it proceeded to rule out or find problems associated with other factors, for example a defective router (as in the case of *mbahamon*) or environmental interference.

Iperf revealed the behavior of the networks trying to use their full capacity. Also checked the behavior of the protocols and the different phases that they have (slow start and congestion avoidance periods). While this was not within the objectives of the study, witness the behavior helped to verify that the operating level protocols, were working as expected. Along with this, it helped to ensure that the presence of buffers is lethal when trying to make a fluid exchange data with a high level of network utilization.

The simulation of the experience gained by a user when trying to load a website under periods of low and high loads also can enable to understand better the effects of Bufferbloat. While the trend was already well defined by the above tests, individual results included in the study were obtained, but defined as outliers, such as occurred in the network *polmos_w* where a maximal value was obtained in the last iteration, which were expected to be more alike to the first iteration.

If it is believed that due to Bufferbloat the times were increased, thanks to smokeping got demonstrated that not only increase considerably, but almost impossible to web surf or almost very difficult to think in online games with good performance. The resulting loss and increased latency are considerable reaching about 12 times the initial value, and growing to 100% loss communication.

8 Conclusions and Further Work

By analyzing all of the work done for this thesis, it is possible to conclude that the objective of it was achieved considering the same frame of reference as the criteria defined at the beginning of it being the *objective of checking the effects of Bufferbloat phenomenon, test the impact that it has in on different networks and to propose solutions.* To accomplish this, it first required to address the following general objectives:

- To explain the *Bufferbloat* phenomenon, and explain the impact that it could have over latency and throughput in Internet.
- To detect the presence by empirical measure of the latency and throughput in a TCP/IP Network.
- To propose possible solutions in the implementation of a network where the existence of excessively large and frequently buffers are detected.

Thanks to the various tests it was possible to demonstrate the presence and feel the effects of *Bufferbloat* in some networks. While the common factor in them at first sight is their low bandwidth, you can not directly assign this as a cause, as also contrary cases that had a bad performance determined by physical effects such as routers were in poor condition or environmental interference.

As proven in this thesis and corroborated in [21], in general, a low latency network is wanted in order to exchange messages between a server and a client. Since low latency make us feel a faster web surfing and enables better performance in online games and VoIP technology.

It is clear that the farther away a clients is from the server, the latency will be higher, but what if we put more and more load to the network, how much the latency can go

up to? Having 12 times the latency when the network overload is not normal and as mentioned by *Jim Gettys* several times, the culprit is Bufferbloat.

The effects on latency and navigation are catastrophic growing to full charging time in minutes or even generate timeout sites. If we take today's world where it is more and more common to see applications that make use of high bandwidth, makes it almost impossible navigating multiple users with these characteristics. Then, two users on the same network using such applications should take turns to ensure a smooth operation.

Also from [2] and [40], we can see that the rule-of-thumb of sizing routers $B = (\overline{RTT}xC)$ doesn't longer apply. With routes which have sizes much larger buffers than required, all it does is slow down and hinder the passage of trading floors. The larger the buffer size, the longer it takes for a packet to go through it, not adding any value in the packet transfer and only adding additional latency.

In the past days networks had much lower performance than they are today, so as *Bufferbloat* alike phenomena were more difficult to detect because their effects were not as visible. Today with the advance of technology and the use of tools that feed us with information almost instantaneously, any change or modification at any point within the communication channel is evident, and why not say is magnified even to make some systems collapse.

The complexity of deploying new algorithms into production environments is such that no matter how many tests are performed over new algorithms, the conditions under development environments will hardly match. Also doing testing in production environments is too hard and too complex[40], and if tests are not well designed can affect users in a hard way.

Thanks to Codel, and Bufferbloat community is possible to propose an effective solution to the phenomenon of Bufferbloat. Although while this phenomenon has been

studied for a couple of years, the solutions are not yet fully proven, but these solutions are designed for modern systems (e.g.. Codel requires BQL recommended for Linux kernel 3.5 onwards), so there will always be some systems vulnerable to its effects. Apply QoS and bandwidth limiting for some applications are the most common and easy to implement recommendations for any user, regardless of the operating system on which they are working.

It is also important to keep a regular check of all the components in our networks, not only to our computer. Effects of routers in evil, or using technologies like wireless with the boom of recent years in homes fact that interference between emitters is much more common than years ago, can be determining factors in achieving this long-awaited win the last game shooter, or close the business of our lives through a video conference.

Much is still to be done in this field. As example, the analysis in specific of how the three different flows (Elephant, mice and ant)[13, 4] are affected by big buffers and prioritize smaller flows can help fix some of the issues like DNS resolution, but how they react when the whole network is under the effects of *Bufferbloat*? The effects change if this occurs at a last-mile-router or at a back-bone?

Thanks to studies as done by Hoiland-Jorgensen[15], is that a better understanding of how the Linux Kernel reacts to *Bufferbloat*, but this follows up to all Linux based OS? With the advance of technology, new devices are getting cheaper and most people can have access to them. Well known is the case of Open-Wrt[7], that is a Linux distribution for embedded devices. Projects like Cero-WRT[32], can help to the community to fight against *Bufferbloat* in every piece that is involved on the communication. But will this help in all devices? As example, Arduino Yun is a microcontroller board, with an Atheros processor that supports a Linux distribution based on OpenWrt named OpenWrt-Yun. The board has built-in Ethernet and WiFi support. So, will this board behave well under high

consumption of bandwidth too? Also the effects of *Bufferbloat* under Windows machine are not well known. Beside the recommendations of the Bufferbloat community[31] not much information can be found about it. Study the effects under laying the use of CTCP[33][34] will be a good starting point.

References

[1] Allman, M., Floyd, S., and Partridge, C. Increasing TCP's Initial Window. RFC 3390 (Proposed Standard), Oct. 2002.

[2] Appenzeller, G., Keslassy, I., and McKeown, N. Sizing router buffers. *SIGCOMM Comput. Commun. Rev. 34*, 4 (Aug. 2004), 281–292.

[3] Braden, R. Requirements for Internet Hosts - Communication Layers. RFC 1122 (INTERNET STANDARD), Oct. 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864.

[4] Burguers, B. Evolvements of tcp-, udp-, short-lived tcp- and long-lived tcp- flows. *6th Twente Student Conference on IT* (2006).

[5] Cheshire, S. It's the latency, stupid. `http://rescomp.stanford.edu/~cheshire/rants/Latency.html`, may 1996. [Online, accessed 20-Dic-2011].

[6] Deering, S., and Hinden, R. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871, 6437.

[7] developer Team, O. Openwrt project. `http://www.openwrt.org`.

[8] Dischinger, M., Haeberlen, A., Gummadi, K. P., and Saroiu, S. Characterizing residential broadband networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2007), IMC '07, ACM, pp. 43–56.

[9] Feng, W.-c., Shin, K. G., Kandlur, D. D., and Saha, D. The blue active queue management algorithms. *IEEE/ACM Trans. Netw. 10*, 4 (Aug. 2002), 513–528.

[10] Floyd, S., and Jacobson, V. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw. 1*, 4 (Aug. 1993), 397–413.

[11] Gettys, J. Bufferbloat: "dark" buffers in the internet - demonstrations only. `https://www.youtube.com/watch?v=npiG7EBzHOU`, January 2012.

[12] Gettys, J., and Nichols, K. Bufferbloat: Dark buffers in the internet. *Commun. ACM 55*, 1 (Jan. 2012), 57–65.

[13] Ha, S., and Rhee, I. Taming the elephants: New tcp slow start. *Comput. Netw. 55*, 9 (June 2011), 2092–2110.

[14] Ha, S., Rhee, I., and Xu, L. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev. 42*, 5 (July 2008), 64–74.

[15] Hoiland-Jorgensen, T. Battiling bufferbloat: An experimental comparision of four approaches to queue management in linux.

[16] Jacobson, V. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev. 18*, 4 (Aug. 1988), 314–329.

[17] Jacobson, V. Notes on using red for queue management and congestion avoidance. talk at NANOG 13, June 1998.

[18] Kozelj, M. Re: [iperf-users] how iperf works? `https://www.mail-archive.com/iperf-users\spacefactor\@mlists.sourceforge.net/msg00147.html`, nov 2009.

[19] Kreibich, C., Weaver, N., Nechaev, B., and Paxson, V. Netalyzr: Illuminating the edge network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2010), IMC '10, ACM, pp. 246–259.

[20] Mathis, M., Mahdavi, J., Floyd, S., and Romanow, A. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), Oct. 1996.

[21] Mathis, M., Semke, J., Mahdavi, J., and Ott, T. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev. 27*, 3 (July 1997), 67–82.

[22] Nichols, K., and Jacobson, V. Controlling queue delay. *Queue 10*, 5 (May 2012), 20:20–20:34.

[23] Postel, J. Transmission Control Protocol. RFC 793 (Standard), Sept. 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[24] Postel, J. TCP maximum segment size and related topics. RFC 879, Nov. 1983.

[25] Ramakrishnan, K., Floyd, S., and Black, D. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), Sept. 2001. Updated by RFCs 4301, 6040.

[26] Staff, C. Bufferbloat: what's wrong with the internet? *Commun. ACM 55*, 2 (Feb. 2012), 40–47.

[27] SUBTEL. Modelo de competencia por calidad de servicio. `http://www.subtel.gob.cl/images/stories/apoyo_articulos/notas_prensa/modelo_competencia_qos_subtel_23enero2012.pdf`, January 2013.

[28] SUBTEL. Chile se mantiene por anõ consecutivo por sobre el promedio ocde en crecimiento de banda ancha. `http://www.subtel.gob.cl/noticias/133-banda-ancha/5230-chile-se-mantiene-por-tercer-ano-consecutivo-por-sobre-el-promedio-ocde-en-cr` jan 2014.

[29] SUBTEL. Información estadística, serie conexiones internet fija. `http://www.subtel.gob.cl/images/stories/apoyo_articulos/informacion_estadistica/series_estadisticas/06032014/1_SERIES_CONEXIONES_INTERNET_FIJA_DIC13_050214.xlsx`, March 2014.

[30] Taht, D., and Gettys, J. Best practices for benchmarking codel and fq codel. `https://www.bufferbloat.net/projects/codel/wiki/Best_practices_for_benchmarking_Codel_and_FQ_Codel`, March 2013. Wiki page on bufferbloat.net web site.

[31] Taht, D., and *BufferBloat Community*. Bloat: Windows tips. `http://www.bufferbloat.net/projects/bloat/wiki/Windows_Tips`, February 2011.

[32] Taht, D., and *BufferBloat Community*. Cerowrt project. `http://www.bufferbloat.net/projects/cerowrt`, 2014.

[33] Tan, K., and Song, J. Compound tcp: A scalable and tcp-friendly congestion control for high-speed networks. In *in 4th International workshop on Protocols for Fast Long-Distance Networks (PFLDNet), 2006* (2006).

[34] Tan, K., Song, J., Zhang, Q., and Sridharan, M. A compound tcp approach for high-speed and long distance networks. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings* (April 2006), pp. 1–12.

[35] Telefónica Chile, S. Medidas o acción para la gestión del tráfico y administración de red servicio banda ancha fijo. `http://www.movistar.cl/PortalMovistarWeb/ShowDoc/WLP+Repository/Portlets/P030_Generico/Recursivo/acordeones/acordeon_banda_ancha/pdf/RED_servicio_BandaAnchaFijo.pdf`, 2013.

[36] Telefónica Chile, S. Neutralidad en la red. `http://www.movistar.cl/PortalMovistarWeb/neutralidad-de-la-red`, 2013.

[37] *Bufferbloat.net*. Community web site. `http://www.bufferbloat.net/`, 2012.

[38] VTR Banda Ancha Chile, S. Políticas de administración de red. `http://vtr.com/neutralidad/b4.php`, 2013.

[39] VTR Banda Ancha Chile, S. Vtr - reglamento de neutralidad de red. `http://vtr.com/neutralidad/b3.php`, 2013.

[40] Vu-Brugier, G., Stanojevic, R. S., Leith, D. J., and Shorten, R. N. A critique of recently proposed buffer-sizing strategies. *SIGCOMM Comput. Commun. Rev. 37*, 1 (Jan. 2007), 43–48.

A Speed Test Details results

Location	Medium	Uplink (Mbps)	Downlink (Mbps)	Ping (ms)
casa_w	Wireless	4,95	15,48	18
casa2_w	Wireless	5,09	13,46	20
casa2_c	Cable	5,15	15,44	19
marcia_w	Wireless	0,83	6,3	29
marcia_c	Cable	0,83	6,29	29
sara_w	Wireless	0,56	5,45	31
sara_c	Cable	0,57	8,08	30
polmos_w	Wireless	2,32	19,61	26
polmos2_w	Wireless	2,34	18,72	22
nalucem_w	Wireless	0,57	3,97	35
nalucem2_w	Wireless	0,56	4,06	36
mbahamon_w	Wireless	0,6	10,21	24
4low_w	Wireless	0,6	8,03	29
hidalgo_2	Wireless	1,19	17,96	19

Table 7: Speeds and Pings measured.

		Measured (Mbps)		Contracted (Mbps)	
Location	Provider	Uplink	Downlink	Uplink	Downlink
casa_w	Movistar	4,95	15,48	5	15
casa2_w	Movistar	5,09	13,46	5	15
casa_c	Movistar	5,15	15,44	5	15
marcia_w	VTR	0,83	6,3	0,5	10
marcia_c	VTR	0,83	6,29	0,5	10
sara_w	Movistar	0,56	5,45	0,5	10
sara_c	Movistar	0,57	8,08	0,5	10
polmos_w	VTR	2,32	19,61	2	40
polmos2_w	VTR	2,34	18,72	2	40
nalucem_w	Movistar	0,57	3,97	0,5	4
nalucem2_w	Movistar	0,56	4,06	0,5	4
mbahamon_w	VTR	0,6	10,21	0,5	10
4low_w	VTR	0,6	8,03	0,5	10
hidalgo_w	VTR	1,19	17,96	1	20

Table 8: Measured and Contracted Speed.

B Page Benchmark results

	Total Load Means (ms)				
Name	1	2	3	4	5
4low_w	3415,10	20932,30	4490,50	4170,10	4087,10
casa_w	1654,80	3053,60	1718,00	1877,40	1666,80
casa2_w	1682,90	2354,40	1682,30	1667,50	1646,60
casa_c	1191,00	1944,10	1167,10	1165,30	1180,90
mbahamon_w	2248,40	23868,50	2248,00	2182,60	2223,80
marcia_w	9401,60	60119,00	10692,40	10646,70	42022,80
marcia_c	11140,00	10764,30	10158,80	10386,30	10028,30
nalucem_w	6421,40	26048,60	6280,80	6335,80	6306,40
nalucem2_w	8199,00	32151,50	6479,10	6537,10	6349,40
polmos_w	1623,40	3701,40	2420,60	1947,00	1659,30
polmos2_w	1912,40	4861,80	1738,50	2136,10	1827,50
sara_w	3781,30	26561,50	2808,60	3005,30	3101,00
sara_c	2806,50	70423,30	2865,90	2757,60	2964,50

Table 9: Total Load Means.