UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Universidad Técnica Federico Santa María
Departamento de Informática

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

April 16, 2014
Valparaíso, Chile

Contents

List of Figures

List of Tables

1 Introduction

If a little salt makes food taste better, then a lot must make it taste great, right?. What happens if you apply the same statement to a network domain? It keeps been as good as it was? It improves the performance or makes it worse?.

Lets think of a network as a road system where everyone drives at the maximum speed. When the road gets full, there are only two choices: crash into other cars, or get off the road and wait until things get better. The former isn't as disastrous on a network as it would be in real life: losing packets in the middle of a communication session isn't a big deal. But making a packet wait for a short time is usually better than "dropping" it and having to wait for a re-transmission.[24]

At this point, the role of the router becomes important. It has to control the congestion effectively in networks. It is important to remember that the traffic in a network is inherently bursty, so 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.

These problems are fixed with the creation of a group of FIFO based queue management mechanisms to support end-to-end congestion control in the internet. That procedure is called Active Queue Management (AQM). With AQM the loss of package and the average queue length is reduced; this impacts in a decreasing end-to-end delay by drooping packages before buffer comes full, using the exponential weighted average queue length as a congestion indicator. For the proper use of AQM, it has to be widely enabled and consistently configured the router.

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.[8]

The existence of cheap memory and a misguided desire to avoid packet loss has led to larger and larger buffers being deployed in the hosts, routers, and switches that make up the Internet. 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.

2 The Bufferbloat Foundations

2.1 The TCP Protocol [13][19]

It is not hard to see that in the past few decades, the growth of internet and the ways that we uses it has exceed any expectation. With that, the problems related with it also as increase. For example, is common to see internet gateways drop 10% of incoming packets because of local buffer overflows. As we will see through this paper, many are related not with the protocol themselves, instead, with the ways that these protocols are implemented. The obvious ways to implement a protocol, sometimes, can result in exactly opposite behavior. One example is the congestion collapse identified as a possible problem as far back as 1984 [12]. It was first observed on the early Internet in October 1986, when the NSFnet phase-I backbone dropped three orders of magnitude from its capacity of 32 kbit/s to 40 bit/s, and this continued to occur until end nodes started implementing Van Jacobson's congestion control between 1987 and 1988.

The root idea of any algorithm related to transport connections must be based into the "*packet conservation principle*". This principle claims that **a new packet isn't put into the network until an old packet leaves**. From physics, a conservative flow means that for any given time, the integral of the packet density around the sender-receiver-sender loop is a constant, and should be robust to the face of congestion[i] If this principle is obeyed, congestion collapse would become the exception rather than a rule, so the only three ways for packet conservation to fail are:

1. The connection doesn't get to equilibrium,
2. A sender injects a new packet before an old packet has exited,
3. The equilibrium can't be reached because of resource limits along the path.

The first failure has to be from a connection that is either starting or restarting after a packet as loss. The second and third are addressed once the data is flowing reliably.

The Transmission Control Protocol is one of the core protocols of the Internet Protocol Suite, based on a connection less end-to-end packet service. The advantages of its connectionless 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, the cost are: the needed of careful design so it provides a good service under heavy loads, or the result can be another "Melt Down" like in the '86.

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 a flows must "back off" during a congestion.

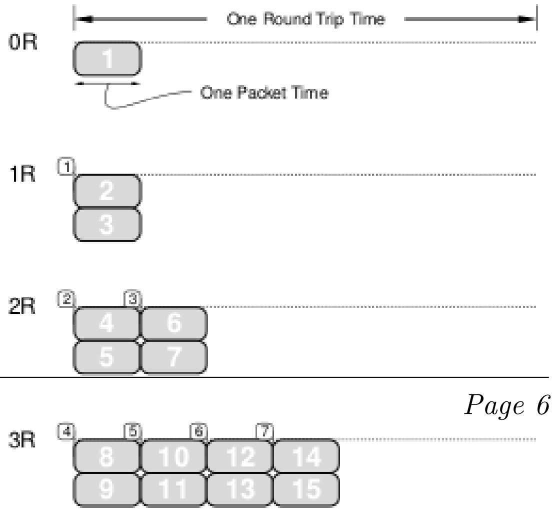
[i] The proof of this property is out of the scope, but if is needed it will be analyzed in the final paper.

The technique that TCP uses, requires the receiver to respond with an acknowledgment message as it receives a packet of data. This technique could be use as a clock to adapt to the “conservation property”. Since the receiver can generate ACK no faster than data packets can get through the network, the protocol is “self clocking” to when it has to put a new packet into the line, and by that, the system can be stable. But this same property causes a redundant problem: in order to get data flowing, there must be data flowing that tell when to put new packets into the system. For this reason, TCP posses two algorithms, the *slow start* and *congestion avoidance.* This algorithms were designed to keep in “equilibrium” the data that is flowing through the system.

2.1.1 Slow-Strart Algorithm[14][5]

Old TCPs implementation, would start a connection with the sender injecting multiple segments into the network, up to the window size advertised by the receiver. This action has no implications if the two host are into the same LAN. While this is OK when the two hosts are on the same LAN; but in the Internet, this schema isn’t valid. As is know, between the two end points are routers and getaways, and the flow of packages is not constant, between the two end points some links could be slower than others, and some intermediate buffer’s queues could run out of space.

The algorithm to starts this “*clock*” and to avoid this congestion is called *slow start.* It is the responsible to gradually increase the amount of data that is in transit by observing that the a new package is injected into the network until the acknowledgment of a previous packages as arrives



from the other end. In other words, it is used to avoid sending more data than the network is capable of transmit.

Slow start adds a variable window to the sender's TCP: the congestion window, called "*cwnd*" to the per-connection state. When a new connection is established or restarting after a loss connection with a host, the congestion window is initialized to one segment, with the size of two times the maximum segment size (MMS)[ii] . Each time an ACK is received, the congestion window is increased by one segment (1 MMS). The sender can transmit up to the minimum of the congestion window and the advertised window. The congestion window is flow control imposed by the sender, while the advertised window is flow control imposed by the receiver. The former is based on the sender's assessment of perceived network congestion; the latter is related to the amount of available buffer space at the receiver for this connection.

The sender starts by transmitting one segment and waiting for its ACK. When that ACK is received, the congestion window is incremented from one to two, and two segments can be sent. When each of those two segments is acknowledged, the congestion window is increased to four as seen in figure 1. Here, the gray numbered boxes are packages and the white are the corresponding ACK. As each ACK arrives, two packages are generated, one for the ACK package that left the "*pipe*" and one because an ACK opens the congestion window by one. This provides an exponential growth, it takes time $Rlog_2W$[9], where R is the round trip time and W is the window size. Although it is not exactly exponential because the receiver may delay its ACKs,typically sending

[ii] The study of MMS is out of the scope of this paper, but more info could be found in [14] and [5]

one ACK for every two segments that it receives.

At some point the capacity of the internet can be reached, and an intermediate router will start discarding packets. This tells the sender that its congestion window has gotten too large. Early implementations performed slow start only if the other end was on a different network. Current implementations always perform slow start.

2.1.2 The Congestion Avoidance Algorithm [3][1]

In the "*slow start*" phase, if a when a lost occurs, half of the current window is saved as a Gresham a variable that is used to determine whether the slow start or congestion avoidance algorithm is used to control data transmission. After this, the *cwnd* is set again to 1 and start to grown until it reaches the ssthresh again. Now, TCP goes into congestion avoidance mode, where for each ACK increases the cwnd in 1/cwnd. A congestion can occur when data arrives at a router whose output capacity is less than the sum of the inputs. Congestion avoidance is a way to deal with lost packets.

This algorithm makes a fundamental assumption: *the packet loss caused by damage is very small (much less than 1%), therefore the loss of a packet signals congestion somewhere in the network between the source and destination.* Two are the indication of package lost: a timeout occurring and the receipt of duplicate ACKs.

A good congestion avoidance strategy, must have two components: 1.- The endpoints should know when a congestion is about to occur or occurring into the network, and 2.- If a signal that alert the congestion is received, the network's utilization must decrease and increases if the signal isn't received.

When a networks is getting congested, the queue lengths will start to increase exponentially (because of slow-start). The system will collapse if the network doesn't throttle back the traffic sources at leas as quick as the queues are growing. The way that the network announces via dropped packets when demand is excessive, but says nothing if a connection is using less than its fair share. This is also another problem because causes underbuffering, also causing that the resources are not fully under full use.

The implementation of congestion avoidance is as simple as slow start, but with a slight difference[9]. The steps are:

1. On any timeout, set *cwnd* to half the current window size. This produces a multiplicative decrease.

2. On each ack for new data, increase *cwnd* by 1/*cwnd*. Now cwnd has an additive increment so now the growth becomes linear.

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

A window of size cwnd packets will generate at most cwnd ACKs in one round trip time. Thus an increment of 1/cwnd per ACK will increase the window by at most one packet in one RTT. In TCP, windows and packet are in bytes so the increment translates to segsize*segsize/cwnd, where segsize is the segment size and cwnd is maintained in bytes.

Congestion avoidance and slow start are independent algorithms with different objectives. But when congestion occurs TCP must slow down its transmission rate of packets into the network, and then invoke slow start to get things going again. In practice they are implemented together. So, if cwnd is less than or equal to ssthresh, TCP is in slow start; otherwise TCP is performing congestion avoidance. Slow start continues until TCP is halfway to where it was when congestion occurred (since it recorded half of the window size that caused the problem), and then congestion avoidance takes over.

2.1.3 The Router's Congestion Avodance Complements

After the "congestion collapse" and with the growth in the last few decades of Internet, it has become clear that the TCP congestion avoidance mechanisms, while necessary and powerful, are not enough to provide a fully safe service, and the control that can be accomplished from the edges of the networks has proof that has a limit. So, in order to obtain a good service in all circumstances, some mechanisms in the routers to complement this endpoint congestion avoidance mechanisms are needed.

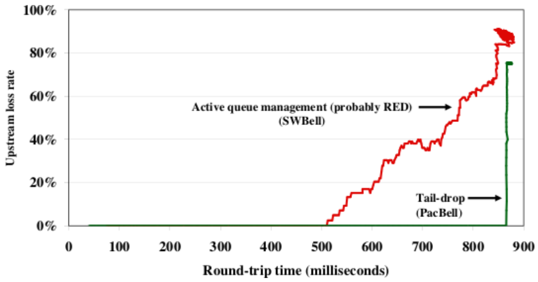


Figure 2: How Tail-drop management and RED AQM overflows[6]

Two classes of router algorithms related with congestion avoidance control can be distinguished: "queue management" and "scheduling" algorithms. While the first one manage the length of packet queues by dropping packets when necessary or appropriate, the second one determine which packet to send next, and are used to manage the allocation of bandwidth among flows. It is important to notice that while this two mechanisms are closely related, the performance that they address are rather different and should be seen as complementary, and not as replacements for each other.

1. **Managing the Routers Queue:** As we have seen in previous section, the traditional way to manage router queue is, after set a maximum length for the queue, accept packages until this length is reached, and then drop subsequent packages until a packet from the stack as been transmitted. This technique is called "tail

drop", and has served the Internet well enough for years, but it has two important drawbacks:

- It allows, in some situations, a single or few flows to monopolize the queue space, preventing other connections from getting room in the queue.
- The signaling is produced only when the queue is full, so it allows queue to maintain almost full status for long periods.

If queue is full or almost full, an arriving burst will cause multiple packets to be dropped, an this behavior can produce a global synchronization of flows throttling back followed by sustained period of lowered link utilization, which will impact in a reduce of overall throughput, and if a long flows arrives in that period, the lock-out of the queue.

Besides tail drop, another two techniques can be applied in these situations. The "random drop on full" will produce that the router drops a randomly selected packet from the full queue, which requires an $O(N)$ walk through the queue, when a new packet arrives. Under the "drop front on full", the router drops the packet at the front of the queue. Either of these solve the lock-out problem, but neither solves the full-queue.

2. **Active Queue Management and Random Early Detection** In the current Internet, dropped packets are used as a critical mechanism to notify a end node when a congestion is presented. So, if a router is capable to drop packets before the queue is full, and the end nodes take actions before the buffers overflow, the full-queue problem is solved. This proactive approach is know as "active queue management" (AQM) and allows routers to control when and how many packets to drop before buffers overflow.

For responsive flows, AQM can provide:

- reduce number of packets dropped in routers
- provide lower-delay interactive service
- avoid lock-out behavior

One AQM algorithm for routers is called "Random Early Detection" or RED. The algorithm drops arriving packets probabilistically, which increases as the estimated average queue size grows, so it approach is based on the "recent past" events.

The RED algorithm consists of two main parts:

- Estimation of the average queue size
- Packet drop decision

RED's particular algorithm for dropping is the culprit in the performance improvement.

2.2 Latency

When we move an amount of data, like a music file, it takes several minutes, or if we have lucky, several seconds. Smaller the file gets, less time is the duration of the transfer, but there is a limit. No matter how small the file becomes, we are stocked with a minimum time that we can never beat. That is call latency of the device. For an Ethernet network is $0.3ms$.

Maybe we don't notice the effect of this time, that when the amount of data is large enough, this time is too small compared with the time that takes the whole transfer. But, what happens with short-flows, like game streaming? Let's imagine that we want to stream audio over the net. 100ms may not sound very much, but it's enough to notice a delay and echo in voice. A better case can be found in [4], where the effects of latency in the transmissions and the buffers can be found.

There is no visible impact of varying the latency other than its direct effect of varying the bandwidth-delay product. Congestion can also be caused by deny of service attack that attempts to flood host or routers with large amount of network traffic.

3 Characterization of Buffers

buffers

3.1 Backbone Routers

As we already know, all internet routers contains buffers to hold packets during times of congestion. A widely used rule-of-thumb states that each link need a buffer of size $B = \overline{RTT}xC$, where $\overline{RTT}$ is the average round trip time of a flow passing across the link, and C is the data rate of the link. The main characteristic of bufferbloat is 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, so router buffers are the single biggest contributor to uncertainty in the Internet.

The rule-of-thumb come from a desire to keep the link as busy as possible so, the throughput of the network is always as big as possible. But, because the way that TCP works, no matter how big the buffer is at the bottleneck link, TCP will cause the buffer to overflow.

Overbuffering is a bad idea for two reasons:

1. It complicates the design of high-speed routers, leading to higher power consumption, more board space, and lower density.
2. It increases end-to-end delay in the presence of congestion

As seen in 2.2, large buffers only increases latency, and this only causes conflict with the needs of real time applications.

The most important fact of sizing a buffer is to make that sure that while the sender pauses, the router buffer doesn't go empty and force the bottleneck to go idle. Again, the idea is to keep as much throughput as possible so the use of the link is fully utilized.

The buffer will avoid to idle if the first packet from the sender shows up at the buffer just as it hits empty. In previous section, we define that 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 $B \leq (W_{max}/2)$.

Also from [2], we can see that the rule-of-thumb doesn't longer apply to backbone routers, and a better estimator of the size of a buffer with n flows would be no more than $B = (\overline{RTT}xC)/\sqrt{n}$. With the assumption that short-flows plays a very small effect, and that the buffer size is dictated by the number of long flows, this factor will be proof that routers are much longer than they need to be, possible by two order of magnitude.

3.2 Residential BroadBand Networks

It is well know that residential networks are often the bottleneck in the last mile access to the Internet Infrastructure. This could be because the ISP's of both of the most popular ways to access (DSL and cable networks) to internet today, use traffic shaping methods and ,as seen in previous sections, deploy massive queues that can delay packets for several hundred milliseconds.

Both shares the asymmetric bandwidths; they downstream bandwidth is higher than their upstream bandwidth, but in cable networks a single coaxial cable shares multiple customers, they can concatenate multiple upstream packets into a single transmission, which result in short bursts at high data rates, so the latency can heavily fluctuate. This concatenation can produce a jitter time, that under high network load can be higher than end-to-end jitter over the entire path under normal load, which can be produce a miss interpretation for some protocols of incipient congestion and cause to enter into congestion control avoidance too early.

In the other hand, in DSL networks, the maximum data transmission rate falls with increasing distance from the head, thus in order to boost the transmission rate, DSL relies on advanced signal processing and error correction algorithms which can lead to high packet propagation delays .

4 Experimental Work

The goal of the experiments outlined in this section is to examine different residential and public networks with the objective to proof 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 behave 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 that will be used to perform the tests, the selected tools for each of these tests and what is to be achieved and expected from each one of these tests.

4.1 The Test Setup

The tests will be ran under a pseudo controlled environment, using one physical machine and a second virtual machine hosted in first machine. These machines runs under a regular OS without any modification beside the ones that the own OS. Also, for some tests two other devices will be added, one acting as a 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 is there more information about how the flows are treated by the QM algorithms or about how they are configured.

4.1.1 Hardware Characterization

Physical Machine :

The physical machine runs as host OS, Windows 7 SP1, that works with a 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 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 machine will be used as a server for the Iperf test. This is a VPS hosted by Digital Ocean [iii] with 512MB Ram, and 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 a Intel Hex-Core 3 GHz.

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

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 device the machine will take care of all the management and administration of the device, avoiding any possibility that the host machine modify or manages any flow. Also, by using a second machine to overload the uplink, causes to avoid 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 has created a set of best practices[23] 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. Only will be taken in consideration if it is the QOS present in some routers and deactivate it.

4.2 Tools Definition

The tools used for the benchmark were selected by the capability to determinate the presence of the Bufferbloat phenomenon in the network of study by the analysis of its indicator, the latency or the round trip time[iv]. So, with this as a main consideration, the tool's selected for the benchmark will be chosen by the complexity and accuracy to measure and determine the RTT into a IP/TCP connection, and how hard is to consistently replicate the results under similar contexts. Other tools were selected to help to determine to characterize and proof the capability of the network to cause Bufferbloat.

[iv] To SUBTEL, organization responsible for control and supervision in the performance of telecommunications in Chile, RTT and Latency correspond to the same thing [20]

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

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

4.2.1 Speedtest

Developed by Ookla[v], 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 in Android, iOS or Windows Phone. A command line interface developed in Python can be used too for testing Internet bandwidth.

The server that will be 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).

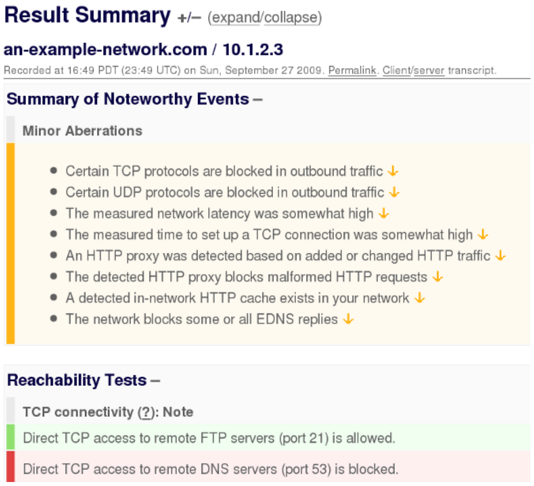
4.2.2 Netalyzer

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 operating aspects, those that show potential signs of trouble, and those

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

that are downright broken. Figure **??** 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*[11]. Among the performance considerations, Netalyzr's measure are packet loss, latency, bandwidth. Also, it compute in-path forwarding device buffer size by comparing small-packet latencies under idle and loaded states in networks (the perfect time to occur Bufferbloat). Other tests like general TCP and UDP service reachability.



Result Summary +/– (expand/collapse)

an-example-network.com / 10.1.2.3

Recorded at 16:49 PDT (23:49 UTC) on Sun, September 27 2009. Permalink. Client/server transcript.

Summary of Noteworthy Events –

Minor Aberrations

- Certain TCP protocols are blocked in outbound traffic ↓
- Certain UDP protocols are blocked in outbound traffic ↓
- The measured network latency was somewhat high ↓
- The measured time to set up a TCP connection was somewhat high ↓
- An HTTP proxy was detected based on added or changed HTTP traffic ↓
- The detected HTTP proxy blocks malformed HTTP requests ↓
- A detected in-network HTTP cache exists in your network ↓
- The network blocks some or all EDNS replies ↓

Reachability Tests –

TCP connectivity (?): Note

Direct TCP access to remote FTP servers (port 21) is allowed.

Direct TCP access to remote DNS servers (port 53) is blocked.

Figure 3: Result Summary example from Netalyzr. Source: `www.dslreports.com`

4.2.3 Iperf

Iperf is a well known and commonly used network testing tool. It can create a 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 or 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*[10], 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.

4.2.4 Page Benchmarker

Page Benchmarker is a Google Chrome extension that intent to test page load time performance within Chrome. 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 results.

4.2.5 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. With

SmokePing give us the ability to measure latency and packet loss in the current network, with RRDtool 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.

4.3 Test Description

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

1. Twice on the same day in one network to determine if does exists a considerable variance in latency for different times of 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 under circumstances experienced by everyday users, no kernel parameter 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 to generate the Bufferbloat phenomenon whether under low loads and if does exists, the how seriours are the effects ?"*

4.3.1 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 aOokla.

The benefits of using this web sites is that not only it will reveal not only the available the 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 [18] and [16] by the two most used ISP in Chile[vi].

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 of 85%, while for international links decreases drastically to 35%[21]. 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.

4.3.2 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 give 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 present in Chile assure consumers networks free of traffic shaping barriers and filters, ISPs apply traffic management measures[17][15] to deliver an optimum experience of the service's use and thus, achieve an efficient use of the network, thus

[vi] Telefónica has a 39.2% market share while VTR owns 38.8% respectively. [22]

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.

4.3.3 Collapse test

Already having a clear idea of the stat 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 (both national and international). 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 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 estimated, 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.

4.3.4 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 since the previous iteration have finished. The site chosen is `http://www.usm.cl`.

As like Jim Gettys showed in his video demonstration[7], 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.

4.3.5 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 (ie: physical and VPS machines) which have different connection settings and/or type of services (some servers has dedicated link), and located in both Chile and the United States.

The tool will be left couple of minutes to track and save the state of the connection under no load(except the required for testing). 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.

5 Results

Here goes a description of what will be shown in this section

5.1 Speed test

Here goes the information related to test speed getting the differences and results gathered by this tests.

Will be a table with the current speed and pings. Also will be difference table with the speed that must has.

Table 1: Speeds and Pings measured.

Location	Medium	Downlink (Mbps)	Uplink (Mbps)	Ping (ms)
4low	Wireless	0,6	8,03	29
mbahamon	Wireless	0,6	10,21	24
mhbrisas	Wireless	1,19	17,96	19
mgallard	Wireless	x	x	x
garriag	Wireless	x	x	x
casa	Wireless	4,95	15,48	18
casa2	Wireless	5,09	13,46	20
polmos	Wireless	2,32	19,61	26
polmos2	Wireless	2,34	18,72	22
nalucem	Wireless	0,57	3,97	35
nalucem2	Wireless	0,56	4,06	36
mcatala	Wireless	x	x	x
mcatala2	Wireless	x	x	x

5.2 Netalyzr test

Here will be a resume of what netalyrz give us

5.3 Iperf test

Graphs and what happend with the rtts

Table 2: Variation ratio of offered vs measured speed.

Location	Uplink ratio	Downlink ratio
4low	120,00	80,30
mbahamon	120,00	102,10
mhbrisas	119,00	89,80
mgallard	0,00	0,00
garriag	0,00	0,00
casa	99,00	103,20
casa2	101,80	89,73
polmos	116,00	49,03
polmos2	117,00	46,80
nalucem	114,00	99,25
nalucem2	112,00	101,50

5.4 Benchmark test

Page benchmark results with tables with differences and mean results

Table 3: Total load mean times.

Lugar	1	2	3	4	5
4low	3415,1	20932,3	4490,5	4170,1	4087,1
casa	1654,8	3053,6	1718	1877,4	1666,8
mbahamon	2248,4	23868,5	2248	2182,6	2223,8
polmos	1623,4	3701,4	2420,6	1947	1659,3
polmos2	1912,4	4861,8	1738,5	2136,1	1827,5
nalucem	6421,4	26048,6	6280,8	6335,8	6306,4
casa2	1682,9	2354,4	1682,3	1667,5	1646,6
nalucem2	8199	32151,5	6479,1	6537,1	6349,4

5.5 Smokeping Test

Graph results

5.6 Summarising Results

all test

Table 4: Variation ratio over own iteration.

Lugar	1	2	3	4	5
4low	55,55	607,623	96,647	132,874	3202,323
casa	8,353	199,264	16,087	256,497	18,852
mbahamon	63,959	83,863	53,778	39,558	219,521
polmos	6,365	163,106	3802,472	8343,236	4050,064
polmos2	59,754	1891,192	8106,53	317,45	8126,291
nalucem	161,954	154,734	9,858	6,96	5,233
casa2	13,998	181,652	27,18	3,102	17,715
nalucem2	880,662	215,955	154,129	184,86	10,703

Table 5: Variation ratio over all iterations.

Lugar	min	max	%
4low	3415,1	20932,3	612,933
casa	1654,8	3053,6	184,529
mbahamon	2182,6	23868,5	1093,581
polmos	1623,4	3701,4	228,002
polmos2	1738,5	4861,8	279,654
nalucem	6280,8	26048,6	414,733
casa2	1646,6	2354,4	142,985
nalucem2	6349,4	32151,5	506,37

6 Conclusions

conclusions

7 Further Work

further

References

[1] Allman, M., Paxson, V., and Stevens, W. TCP Congestion Control. RFC 2581 (Proposed Standard), Apr. 1999. Obsoleted by RFC 5681, updated by RFC 3390.

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

[3] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and Zhang, L. Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC 2309 (Informational), Apr. 1998.

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

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

[6] 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.

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

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

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

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

[11] 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.

[12] Nagle, J. Congestion Control in IP/TCP Internetworks. RFC 896, Jan. 1984.

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

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

[15] S.A., T. C. 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.

[16] S.A., T. C. Neutralidad en la red. `http://www.movistar.cl/PortalMovistarWeb/neutralidad-de-la-red`, 2013.

[17] S.A., V. B. A. C. Políticas de administración de red. `http://vtr.com/neutralidad/b4.php`, 2013.

[18] S.A., V. B. A. C. Vtr - reglamento de neutralidad de red. `http://vtr.com/neutralidad/b3.php`, 2013.

[19] Stevens, W. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001 (Proposed Standard), Jan. 1997. Obsoleted by RFC 2581.

[20] SUBTEL. Ref. fija indicadores de calidad de los enlaces de conexión para cursar el tráfico nacional de internet y sistema de publicidad de los mismos. `http://www.subtel.gob.cl/images/stories/articles/subtel/asocfile/res_698_trafico_internet.PDF`, June 2000.

[21] 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.

[22] 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.

[23] 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.

[24] van Beijnum, I. Understanding bufferbloat and the network buffer arms race. `http://arstechnica.com/tech-policy/news/2011/01/understanding-bufferbloat-and-the-network-buffer-arms-race.ars`, jan 2011. [Online, accessed 15-Dic-2011].