Universidad Técnica Federico Santa María
Departamento de Informática

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

State of Art

JUAN S. CATALAN OLMOS

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

Profesor Guía: Horst H. von Brand Skopnik
Profesor Coreferente: Raúl Monge Anwandter

March 28, 2014
Valparaíso, Chile

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.[1]

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.[2]

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 [3][4]

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 [5]. 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[1] 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.

[1] 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[6][7]

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.

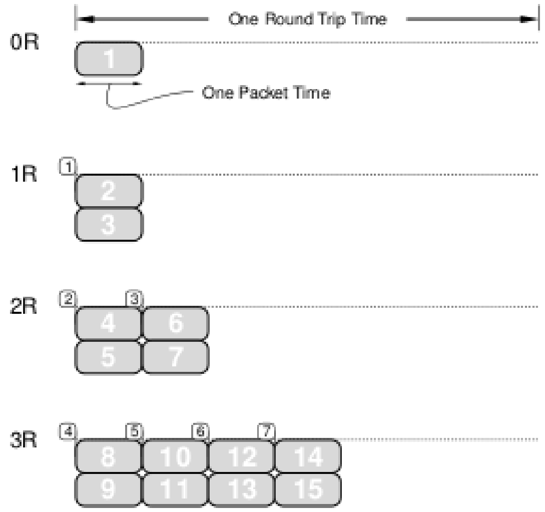

Figure 1: The Chronology of a Slow-Start.[8]

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)[2] . 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

[2] The study of MMS is out of the scope of this paper, but more info could be found in [6] and [7]

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 $Rlog_2W$[8], 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 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 [9][10]

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[8]. 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.

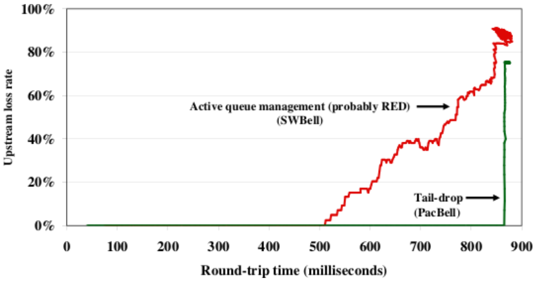

Figure 2: How Tail-drop management and RED AQM overflows[11]

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 [12], 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 Bufferbloat

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 [13], 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

experimental

4.1 The Test Setup

setup

define also the vps hosted by Digital Ocean with 512MB Ram, 20GB SSD Disk, New York 2, Ubuntu 12.04.3 x64 running under KVM sw using Intel Hex-Core CPU between 2 to 3 GHz

4.2 Tools Definition

The tools that will be selected are determinates by the capability to determinate the presece of the Bufferbloat phenomenon in network of study by the analysis of it indicator, the latency or the rount trip time. So, with this as main consideration for the tool's selection, the breanchmarking tools will be choosen by the complexity and accuracy to measure and determine the RTT into a IP/TCP connection, and the capability to replicate the results under similar contexts. Also, some other tools are selected that can help to determine the capability of the network to generate Bufferbloat.

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[3], this tool is used for most of the ISP's and many users in Chile to test their broaband's connections globally. It can be used not only in their website

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

www.speedtest.net but also can be used in Android, iOS or Windows Phone. Also is a command line interface developed in Python 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[14]

asdf

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 behaviour itself (acking). What iperf can measure is the pace with which it sends data to TCP/IP stack; TPC/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*[15], but the data generated by iperf won't be further analized becouse the objective to use this tool is as TCP's packet generator. Those packets will be generated to be captured and analized with tcpdump, tcptrace and xplot.org.

4.2.4 Page Benchmarker

Google Chrome extension

4.2.5 Smokeping

rtt test

4.3 Test Description

description

References

[1] Iljitsch van Beijnum. 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].

[2] Jim Gettys and Kathleen Nichols. Bufferbloat: dark buffers in the internet. *Commun. ACM*, 55(1):57–65, January 2012.

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

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

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

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

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

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

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

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

[11] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing residential broadband networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 43–56, New York, NY, USA, 2007. ACM.

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

[13] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. *SIGCOMM Comput. Commun. Rev.*, 34(4):281–292, August 2004.

[14] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: Illuminating the edge network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 246–259, New York, NY, USA, 2010. ACM.

[15] Metod Kozelj. Re: [iperf-users] how iperf works?, November 2009.