Detecting and quantifying bufferbloat in network paths

Danny Groenewegen
Harald Kleppe

danny.groenewegen@os3.nl
harald.kleppe@os3.nl

August 25, 2011

Abstract

Bufferbloat is the existence of excessively large buffers in network devices, which can result in poor network performance. Techniques such as Automatic Queue Management and Explicit Congestion Notification are available to prevent the issue in Layer 3 queues, but are not widely deployed. In this paper we describe a method to successfully detect the effects of bufferbloat and the responsible link in a network path. We also propose a scoring model called the Bufferbloat score to quantify the phenomenon. Measurements done both in a test environment and on the Internet show that our method is capable of detecting the bufferbloated link(s) in a path.

Contents

List of Figures

List of Tables

1 Introduction

Many network protocols and applications rely on timely arrival of correspondence, and some are designed with the assumption of low latency connections. Introduction of additional latency will first imply degraded performance of the network. When latency increases to several seconds, responses will be considered lost by the receiving party while they in fact are only delayed. Protocols in every layer of the OSI model are affected by this phenomenon: VoIP protocols, web browsing using HTTP, name lookups through DNS, TCP at the transport layer and ARP on the data link layer. In effect latency can have huge implications on network reliability and performance.

Buffers in packets switched networks, such as the Internet are a necessity to absorb short spikes in network traffic and prevent bursts of packet loss. The effects of poor management of oversized buffers have been documented for quite a while[11], but the subject has recently regained attention. In what seems like a race blindly focusing on throughput and minimizing packet loss, the plummeting memory prices have resulted in huge buffers in many network devices. Both in hardware devices such as switches and routers, but also in operating systems and applications. All this has led to a situation where latency is starting to become a major issue.

In 2010 Jim Gettys coined the term bufferbloat [6] to describe the phenomenon of dramatic increases in latency when network links are saturated. *"Bufferbloat is the existence of excessively large (bloated) buffers in systems, particularly network communication systems."* The large unmanaged buffers can exist in many devices, but the effects are only visible during network congestion. Packets being queued in oversized and poorly managed buffers causes extra delay. This phenomenon has major effects on the quality of real time Internet applications. The congestion avoidance mechanisms that are built into TCP implementations rely on packet loss to detect congestion. Common congestion avoidance algorithms in TCP do not take the round trip time, or its increase, into account. Jim Gettys was asked [1] by Vint Cerf to discuss the subject in his usual column in IEEE Internet Computing May/June 2011 issue[7].

1.1 Preventing bufferbloat

In addition to simply making buffers smaller there are multiple techniques to prevent the effects caused by bufferbloat. During network congestion, some packets either have to be dropped or marked with appropriate Explicit Congestion Notification (ECN) bits. ECN is an extension to the IP specification that allows signaling of congestion without packet loss. In the case of either packet drop or ECN marked packets congestion avoiding protocols such as TCP can at least be be prevented from filling buffers. A more thorough solution is Active Queue Management. AQM allows routers to manage their queue sizes to prevent extensive use of buffers, while still making use of them when needed. The main idea is simple: routers should start dropping packet when the queue sizes increase to notify the host about congestion in the network. An example implementation of this is Random Early Detection (RED) [5]. Unfortunately neither RED or ECN is widely deployed on the Internet today.

[1] http://www.bufferbloat.net/news/14

1.2 Project scope

During this project we will focus on methods to detect and quantify bufferbloat in network paths. We will put emphasis on solutions that can be applied on the Internet, and thereby detect buffers using methods on Layer 3 in the OSI model. The goal is a method to detect and quantify the effects of bufferbloat in a network path, requiring only software on one side of the path. Our main research question is:

> *Is it possible to determine which device in a path contributes to the effects of bufferbloat?*
> This question can be divided in several sub-questions:

- Is it possible to quantify the effects of bufferbloat?
- Can the same methods be applied on the Internet, where they are influenced by other unpredictable traffic?

Techniques for estimating characteristics of individual links in network paths already exist. We used *pathchar*, described in section 2, as a starting point in our approach. In section 3 we will describe our approach to extend this technique in order to detect the effects of bufferbloat.

2 Path Characteristics

Multiple techniques exist to gather characteristics of network paths. For example, *traceroute* can be used to enumerate hops in a source-destination path. Also more sophisticated methods exist: *pathchar*, written by Van Jacobson [9], is a tool that infers characteristics of links in a network path. *pathchar* estimates bandwidth, latency and queuing time by sending packets from a single host and measuring round trip times. To detect the effects of buffering in a path, we need the queue delay of each individual link in a path. The methodology used by *pathchar* is able to give estimates for this.

Downey [4] evaluated *pathchar* by looking at two example paths. He identified the circumstances where *pathchar* is likely to succeed and investigated techniques to improve the accuracy. The most interesting is a form of adaptive data collection that reduced the required number of measurements by more then 90% for some links. In the next section we will give an overview of the techniques behind *pathchar* and the adaptive data collection improvements as described by Downey in his paper "Using pathchar to estimate Internet link characteristics" [4].

2.1 Background of pathchar

pathchar extends on the concept used by traceroute: The time-to-live (TTL) field in IP packets is initialized by the original sender of a packet. Each intermediate router that forwards the packet must decrement the value by one. The packet has to be discarded by the router when the value of this field reaches zero. When this happens, the router will send a ICMP Time Exceeded Code 0 (In Transit) message back to the original sender [12]. The source address of the Time Exceeded message identifies the router that discarded the packet. By sending packets to a final destination with the TTL field set to n, the round trip time to the router n in the path can be measured.

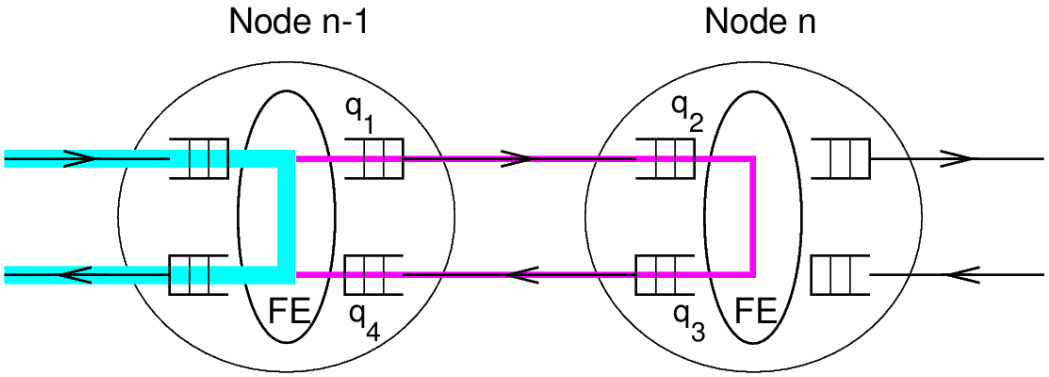


Figure 1: Network model, source: Downey [4]

pathchar continues on this concept by sending packets with varying time-to-live values and varying packet sizes. It measures the time between sending each probe and receiving the Time Exceeded message. The network model in Figure 1 is used to express the round trip time (RTT) from node $(n-1)$ to node n and back:

$$rtt_n = q1 + (latency_n + \frac{packet\ size}{bandwidth}) + q2 + \Delta process\_time + q3 + (latency_n + \frac{icmp\_size}{bandwidth}) + q4 \quad (1)$$

When a packet leaves node $(n-1)$ it waits in the transmit queue before going on the link. The transit time on the network is a linear function of the packet size: $(latency + \frac{packet\ size}{bandwidth})$. On arrival at node n, the packet traverses the receiving queue before being processed. The processing implies that the router at hop n will generate a Time Exceeded error message and put this in the transmitting queue to be transmitted back to the node $(n-1)$. The transit time on the network

back is again linear based on the size of the ICMP Time Exceeded packet. The q_i variables in equation (1) are expressing the unpredictable queue times.

Three assumptions are made to simplify the expression:

- $\frac{icmp\_size}{bandwidth}$ is negligible
- $\Delta process\_time$ is negligible
- Given enough measurements on a network path, it is likely that during at least one measurement a packet will traverse the complete path with near zero queue delays.

2.2 Infer characteristics

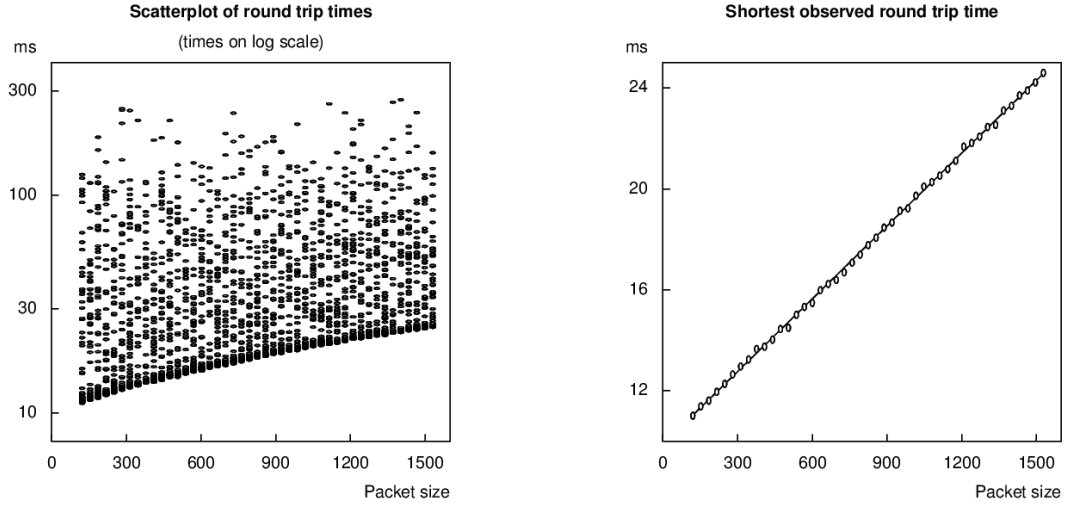


Figure 2: An example of RTT distribution and minimum observed RTT, source: Downey [4]

The characteristics of links distant in a network path can be derived from the information gathered by sending a number of carefully crafted packets and timing the arrival of their responses. *pathchar* sends probes of various sizes and time-to-live values to create a data set. A sample of round trip times for one hop is shown in the scatter plot in figure 2. The scatter plot shows that even though the probes are spread out on the logarithmic Y axis, many of them are near the minimum. Thus when sending enough probes, there is a reasonable chance that some packets will traverse each hop in the network path without notable queue delay. For each packet size the lowest round trip time is saved. A simple form of linear regression, the least squares fitting algorithm, is used to fit a linear line through the remaining set of probes (Figure 2). The intercept of the fitting function is $2 * latency$ and the slope of the function is the inverse of the bandwidth, $\frac{1}{bandwidth}$.

The obtained link parameters are cumulative: If we have the intercepts of the 8th and 9th hop, they can be subtracted to find the latency of the 9th link using equation (1). The same goes for the bandwidth, subtracting the slopes found at hop 8 and 9 results in the inverse of the bandwidth of link 9. The unpredictable queuing times described as q_i in equation (1) are also estimated by *pathchar*. It does this by subtracting the minimum RTT from all other probes of the same packet size to get an estimate of the queue delay distribution.

2.3 Adaptive data collection

Adaptive data collection was suggested to reduce the required amount of probes. This technique is based on the convergence of the interval estimates. The linear least squares fit gives error estimates for the parameters, but they have no meaning in the context of network characteristics. The least

squares fit only takes the minimum RTT of each packet size into account, not the probability of the other probes traversing the path without queue delays. Because there is no statistical model that uses data from the other probes, a technique from non-parametric statistics is used. The set of all probes is divided into smaller subsets and the variation of the estimates is used as an interval for the inferred characteristics.

The data collection starts with a small number of probes and uses subsets to calculate multiple estimates. The convergence criterion is the range of the estimates divided by the smallest estimate [4]. Additional probes are collected until the criterion is below 10%.

2.4 Accuracy & Difficulties

The first assumption, neglecting the size of the ICMP Time Exceeded packet, results in the latency estimates being slightly too high. But with the actual size of the ICMP packet, 56 bytes [13], it is possible to subtract $\frac{icmp\_size}{bandwidth}$ and adjust the latency.

Since the estimates are based on the difference in measurements between multiple links, the $\Delta process\_time$ or forwarding delay is eliminated. However, the variation in forwarding delay between routers will affect the estimates. But as shown by Govindan and Paxon [8], this is not significant: "tools such as pathchar and treno will not in practice suffer greatly".

Downey concluded that estimating the latencies of a link is relatively easy, because they are large in wide-area networks compared to the measurement errors. However, bandwidth estimates depend on the round trip time difference between the largest and smallest packet. The difference gets smaller with higher bandwidths. Thus the higher the bandwidth, the more difficult it is to estimate.

The main problem with estimates of the queue delay distribution are the changing conditions between measurements. Except for the first link, all estimated queue delays include the sum of all previous queue delays at that moment. A quote from Downey:

> *"If x is the total queue delay of the first $n-1$ links of a path, and y is the queue delay of the nth link, then clearly the total delay of the first n links is the sum of x and y, which we call z. If we knew x and z, therefore, it would be trivial to find y. But because of the way pathchar works, we can only measure x or z, never both for the same packet.*
>
> *Nevertheless, by making many measurements of x and z, we can estimate the distribution of each, and by deconvolution we can infer the distribution of y. The simplest form of deconvolution is the subtraction of moments. This is the technique pathchar uses. If we know F_X, the distribution of X, and F_Y, the distribution of Y, we can calculate the first three moments of F_Z, the distribution of the sum $Z = X + Y$, just by adding the moments of F_X and F_Y. This is true for all distributions and does not depend on the independence of X and Y. Using this property, pathchar estimates the moments of F_Y by subtracting the moments of F_X from the moments of F_Z. Since negative queue delays are impossible, the first three moments must increase monotonically from one link to the next. For example, the mean queue delay to the nth node cannot be less than the mean queue delay to the $(n-1)$th node."*

For some links the moments will increase monotonically, but that is not necessarily true for all links. The cause of this is a changing network environment (buffers filling up and emptying again) and the interval between hop measurements: *pathchar* collects all probes from one hop before the next hops. Given that the total run time of *pathchar* can be up to a few hours, the time interval between two hops can be 15-45 minutes. With this interval it is very likely that the queue distributions of previous hops has changed. Consider the following: We measure a total queue delay of x for the first $n-1$ links. While measuring z, the total queue delay for the first n links, the total queue delay for the first $n-1$ links lowers to $x-10$. This results in estimating a negative queue delay for link n, due to changed network environment at the previous links.

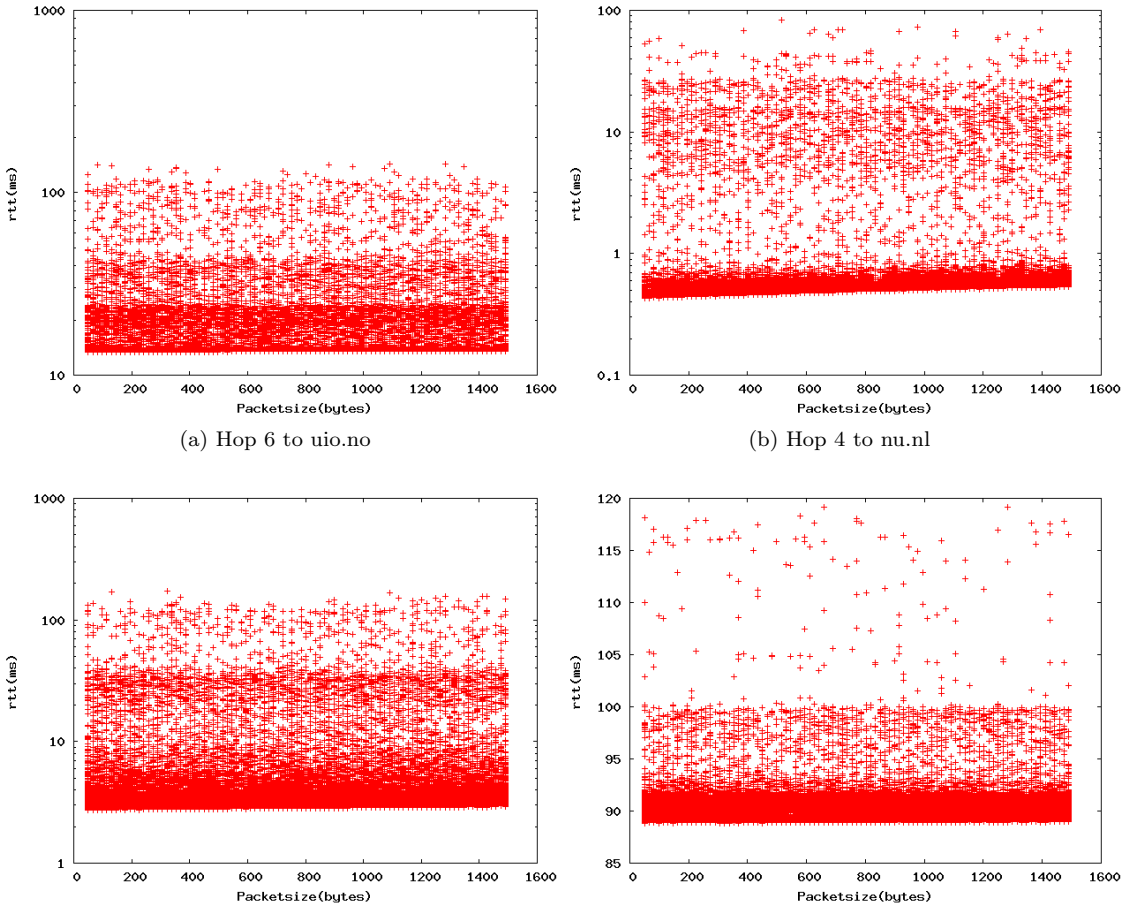
We tried to address this problem by extending the total time frame of the test and minimizing the time interval between measurements of each hop. In the next section we will propose our approach to this.

3 Approach

We tried to address the main issue of using *pathchar* for estimates of queue delay distribution, the changing network environment between hop measurements. The model used by *pathchar* is less suited to measure distribution of queue delays over time, as variations in the network will influence the measurements. We propose to change the model of *pathchar* by making some modifications and simplifications to better fit the task of measuring queue delays. We will describe our method in the following section and implement this in a tool named *Buffchar*[3].

3.1 Estimating queue delays

With today's high speed networks, the difference in RTT between small and large packets is small. Not only the RTT difference between packet sizes is negligible, also the queue distribution. Figure 3 shows the queue distribution among packet sizes at hops in different paths. Looking at the distribution in the scatter plots we see that there is no notable difference between packet sizes. Hence we can simplify our method to only send probes of one packet size for the measurements.



(a) Hop 6 to uio.no

(b) Hop 4 to nu.nl

(c) Hop 5 to utwente.nl

(d) Hop 7 to bufferbloat.net

Figure 3: Distribution of RTT at several hops from Science Park Amsterdam, NL

RTT(ms)	Empty path			TCP (Measured 9.44Mb/s)			UDP 9.569Mb/s			9.568 Mb/s		
Hop	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
1	0.118	0.121	0.130	0.120	0.125	0.138	0.116	0.127	0.144	0.116	0.128	0.138
2	0.176	0.185	0.209	0.176	0.189	0.307	0.186	0.199	0.307	0.186	0.200	0.226
3	0.423	0.431	0.437	196.566	641.396	982.504	36.439	210.879	453.482	0.766	23.791	53.173
4	20.501	20.513	20.542	20.693	592.169	871.846	55.860	230.139	472.434	21.384	43.923	74.290
5	20.580	20.590	20.609	20.749	595.813	901.131	56.623	230.413	473.244	21.034	44.037	73.995
6	20.663	20.670	20.678	20.669	606.569	910.085	56.637	230.758	473.327	21.075	44.206	74.061

Table 1: Minimum RTT on a congested path

3.1.1 Empty queue probability

The distribution of probes in figure 3 shows a clear minimum observed RTT. Even for links several hops away in a path, there seems to be a good probability that packets will traverse the path with a near zero queue delay in every hop. But how likely is it to observe the minimum possible RTT when links in a path are always saturated?

We have set up an experiment to show that this is a valid assumption. The experiment is based on a 6 hop path where the link between hop 2 and 3 is limited to 10Mb/s and the link between 3 and 4 has an additional RTT of 20ms. Without any other traffic, we measured the minimum observed RTT over a period of 15 minutes. During the next runs we used iperf in TCP mode to congest the slow link and saturate the buffers in front of it. A bandwidth of 9.41Mb/s was measured. We ran additional tests and configured iperf with a fixed bandwidth of 9.569Mb/s and 9.568Mb/s. The results can be found in table 1.

The results show that congesting the path and thus congesting the buffers increases the average RTT to 600ms. Even though the path is fully congested at the 10Mb/s link, a minimum RTT of 20.7ms is observed. This can be explained by the behavior of TCP. The changing send and receive windows result in changes in throughput, causing the buffers to be less full for very short moments. Measurements at exactly that moment are able to traverse the path with a near zero queue delay. Next we used iperf in UDP mode to saturate the path with a constant bandwidth. Transferring 9.569Mb/s turned out to be slightly more then the bottleneck link can handle, resulting in saturation of the buffers before the 10Mb/s link. During this test we saw the RTT slowly increasing. The minimum observed RTT (56.637ms) was measured at the beginning of the test, when the buffer started to fill up. After that the RTT only increases due to the increasing amount of packets in the buffer. Based on these results we expect that the minimum RTT can be observed when a link is congested slightly below its maximum bandwidth. The last results in table 1 show iperf in UDP mode configured with 0.001Mb/s less (9.568Mb/s). This is what the link could handle and RTT's very close to the minimum were observed.

Thus in a path where the buffers are always saturated, packets won't be able to traverse the path with near zero queue delays. However, only a short moment of decreased traffic just below each links capability is enough to observe the minimum RTT. Since Internet traffic usually has quiet periods[2], for example during the night or early morning, it is plausible to encounter a moment where the link is not fully congested. Therefor we will assume that the minimum RTT can be observed when probing over a long time period.

3.1.2 Deconvolution of queue delays

The queue delay, of a link n can be found using deconvolution of the total queue delay of the first n links and the first $n-1$ links. But using the earlier described method, those two values can never be measured at the same time. Therefore *pathchar* subtracts the distributions of those queue delays to get an estimate. The accuracy of this is low because the queue delay distribution to hop $n-1$ can be different when measuring the queue delay distribution to hop n. But as we have shown in section 3.1 we can lower the number of measurements by eliminating the variation in packet sizes.

Our approach is to send a probe to each hop in the path at (almost) the same time. This will increase the likelihood of a probe to hop n traversing the same queue delays as the probe to hop $n-1$. We repeat this to estimate the distribution and subtract the moments to estimate the queue delay distribution of each individual link. By probing with a single packet size these measurements

can be done in seconds instead of minutes. We expect that the influence of the changing network environment during this shorter time frame is negligible for our purposes. These measurements result in a value called *queue delay estimate* for a path at one given point in time.

The method as described in the previous paragraph will be repeated every 5 minutes to get an estimate of the queue delay per link over time. This is important because the symptoms of bufferbloat are not visible before links are saturated and the buffers are utilized. For example the buffers at an ISPs router could be congested by multiple other hosts generating traffic. There we need to measure continuously (with 5 minute intervals) to discover if links in a path are affected by bufferbloat.

3.2 Quantifying bufferbloat

Determining the actual size of a buffer is hard, but bufferbloat can be quantified on the basis of the largest measured value. In other words: its potential to be bloated. We propose a *Bufferbloat Score* that can be calculated from the *queue delay estimate* using equation 2. The *Bufferbloat Score* is based on the *queue delay estimate* in seconds. A Bufferbloat Score of 0 indicates that a path is not affected by bufferbloat. And a Bufferbloat Score of 10 or higher indicates that the path is clearly affected by bufferbloat effects. In many situations it could be enough to indicate the effects of bufferbloat with a value ranging from 0 to 10.

$$Bufferbloat\ Score = \frac{queue\ delay\ estimate}{0.025} \cdot 1.75 \tag{2}$$

The two constants in the equation are added to convert the *queue delay estimate* in ms to a *Bufferbloat Score* value where 0 means no bufferbloat effects and values of 10 or higher indicate a heavily affected path. These two constants are based on our observations of various paths during the project and appeared to give sensible values for the *Bufferbloat Score*.

The *Bufferbloat Score* of a path is simply the maximum *Bufferbloat Score* of all measurement intervals. This is analogue to how the bandwidth of a link is quantified. It has a defined maximum capacity, which defines it regardless of whether its full capacity is utilized at the time of discussion.

3.2.1 Why Packet loss is not taken into account

Loss of packets in a packet switched network is common and expected as communication reliability is added in a higher layer in the communication. Transport protocols such as TCP not only mask packet loss, but rely on packet loss as an indication of congestion. ECN, specified in RFC 3168 allows for congestion avoidance without packet loss through marking of packets using specific bits in the IP and transport layer protocol headers. In a situation where both end points and the underlying network are able and willing to use ECN, routers will mark IP packets using dedicated bits in the IP header to indicate impeding congestion. The receiving host will take note of these bits and may utilize fields in transport layer protocols such as TCP to communicate the situation of near congestion to the sender. In turn, the sender can can limit the rate of transmission to avoid congestion, packet loss and retransmissions.

We do not take packet loss into account in our model because packet loss, in absence of ECN, is essential for a healthy operation of the Internet [11]. Further, in the example of a real time application packets arriving delayed and out of order might be discarded anyway. In that case, dropping packets at a congested point could free up bandwidth that is otherwise used to transport doomed packets. Research conducted by B. Gijsbers and D. D. Akkoorath, "Performance simulation of buffer bloat in routers"[1], has shown that the best bandwidth and latency where achieved in situations with smaller buffers and high packet loss. There is of course a tradeoff between dropping packets and buffering to absorb short burst of traffic. Especially with TCP connection it is hard to judge whether packet loss is bad because of the required retransmissions or a good indication of network congestion. Therefore the amount of packet loss can't be directly used as a parameter in quantifying the link quality.

4 Lab environment

Our lab environment consists of 8 physical servers with dual on-board 1 Gigabit Ethernet network cards connected in a string as illustrated by figure 4. The hosts in each end of the string were considered end hosts, and were statically configured whilst configuration were changed as described in the next section on the remaining units. The probing was conducted from host # 1. We used *ethtool* to have individual network interfaces negotiate lower link speeds and *tc* to introduce latency as required by the tests. *Iperf* was used to generate traffic on the network, to saturate links and fill buffers to introduce latency. As the *Iperf* client would be limited to whatever speed an interface on the same host would be limited to is it always running on one host which is has no limitations on bandwidth. We also refrain from running *Iperf* on the same host which does the probing, as it would potentially fill up buffers anywhere in the host and thus affecting the probing results.

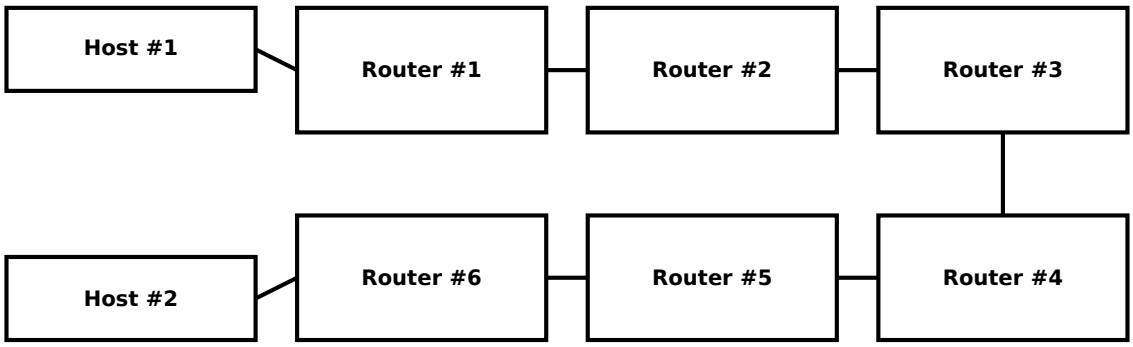


Figure 4: Configuration of lab environment

5 Results

The next sections show the results for each performed test. A table describing the details of the parameters applied in each test can be found in appendix A. We have performed tests both in our lab environment and on real world Internet links.

In the beginning of each test there is a short period of no traffic, where our tool is allowed to measure the minimum round trip time of the path. This period can be quite short in our lab environment, where we are confident that zero queue delay is observed, as there is no other traffic in the network.

5.1 Lab Environment

5.1.1 Test 1

Two equally throttled links

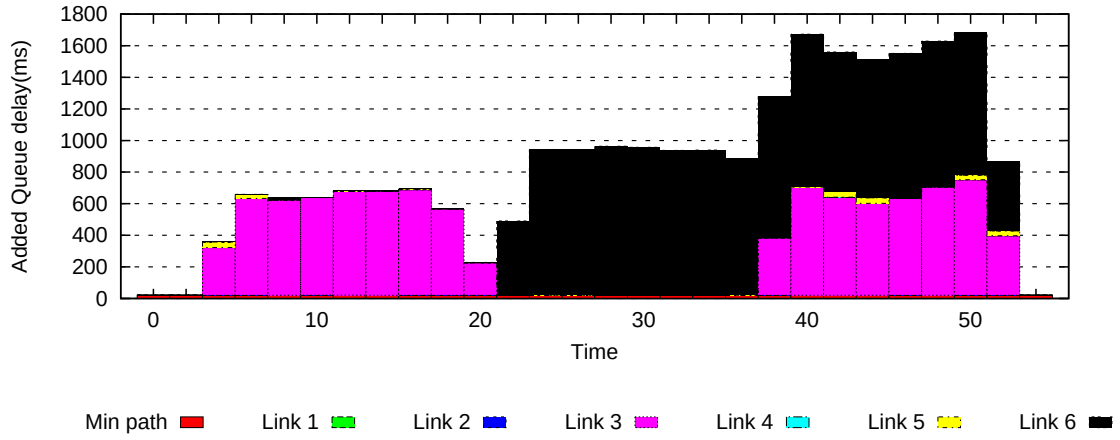


Figure 5: Results of test 1

In the first period of this test is the path between router 1 and router 3 congested. As a result is the queue delay on link 3 increasing to around 600 ms. From the 21st minute on, congesting only the constrained link on the far side of the high latency link results in an observed queue delay of 900ms. When both links are saturated and their buffers have been filled up, from the 39th minute on, a total of 1600ms queue delay is experienced.

Minor irregularities in queue delay at other hops, visible in figure 5 as Link 5, are caused by the variation in queue delays. Our tool is lead to blame the next hop for a small amount of the measured queuing delay when packets occasionally make it through the preceding hop queue with less delay. The overall variation as seen between time stamp 40 and 45, is caused by the variation in bandwidth used on the link as the TCP congestion avoidance algorithm on the sending host is varying transmission rate.

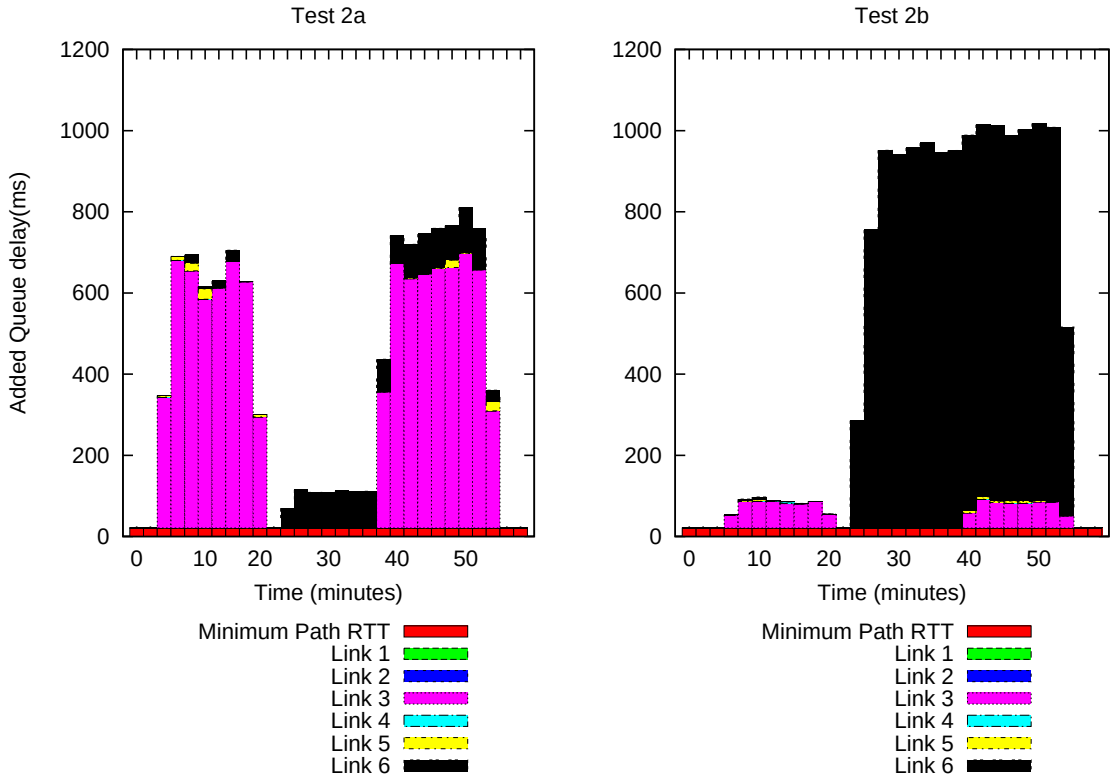


Figure 6: Results of test 2a and 2b

5.1.2 Test 2a and 2b

Two different throttled links, in varying order

Figure 6 shows the results of test 2a and 2b respectively. During this test first link 3 limited to 10Mbit and link 6 to 100Mbit, before they were configured the other way around in 2b. The following test scheme was conducted with both 2a and 2b and is divided into three sections, each of 10 minutes duration:

- Saturate first bottleneck only
- Saturate second bottleneck only
- Saturate both second and the full path simultaneously

This test scheme shred light on how capable our tool is of detecting queue delays in a link on the far side of an intermediate saturated bottleneck.

Test 2a reflects our expectations in form of a stable queue delay introduced by the first bottleneck. As mentioned in the discussion of test 1, some irregularities are caused by occasional, unexpected short queue delays. In both tests the queue delay introduced by the link limited to 100Mb/s is constant and in the order of 100ms. In test 2a, where this is the remote bottleneck, is the latency from this is simply added to the latency of the first. However, in test 2b where the slowest link is further away, the latency introduced by the slowest bottleneck is higher than what we observed with the same bottleneck in the preceding test. This behaviour is investigated in section 5.1.3.

5.1.3 Test 3

Varying in link latency to amplify bufferbloat effect

The goal of this test was to investigate the assumption that our tool falsely reported a too high added latency for a congested link behind an intermediate congested link. This effect was seen in the results described in section 5.1.1.

The approach was to gradually introduce more latency in one of the links in the congested path. The latency was introduced in link 4, and we expected the reported added queue delay for link 6 to increase. As figure 7 shows, this behaviour is not observed.

We will state this phenomenon as a proposal for further research, as it is likely to require a number of simulations with varying combinations of queue sizes and link configurations.

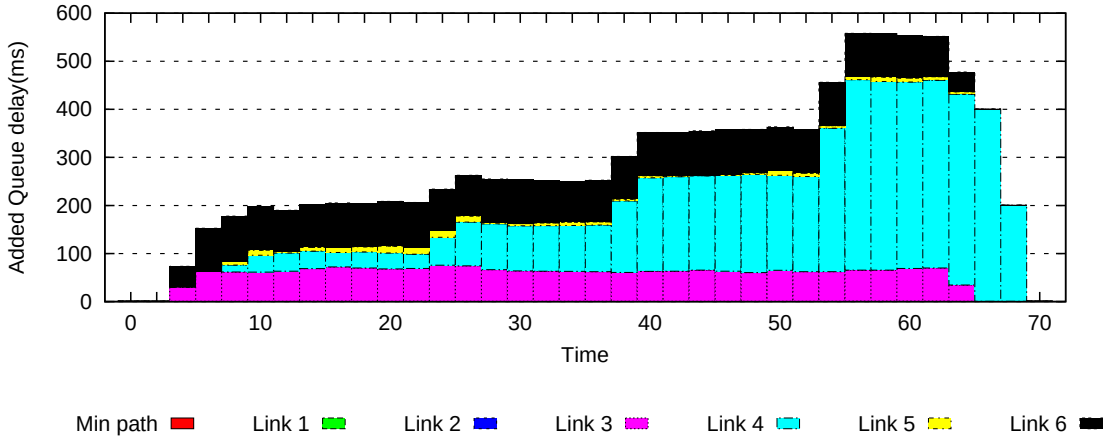


Figure 7: Results of test 3

5.2 Internet Links

Testing our tool on real world Internet links implies a number of new aspects compared to running tests in our controlled lab environment. Most notably, Internet paths are used by unknown and varying amount of cross traffic. Further is the topology of the network less transparent. Path characteristic tools could be used to shred light on the topology, but events like route changes etc could also happen and influence the test.

We have had our tool measuring several Internet links for a while. We measured various paths from servers located in the Netherlands, USA and Norway.

5.2.1 Consumer Grade Connection

Figure 8 shows measurements from a consumer grade 25 / 1.5 Mbit/s connection where we found a Bufferbloat Score of 10+. Around 09:30 one of the host behind the consumer ISP connection started downloading, saturating the full 25Mbit/s down-link. A latency increase up to 130ms is observed. Looking at the graph we see that the source of this increase is the second link in the path, the modem of this Internet connection. Around 11:30 the same host started uploading, saturating the full 1.5Mb/s up-link. An increased latency is observed again at the same link. The device on this link is assumed to have equal buffers on the incoming and outgoing interface. But the increased latency, up to 500ms, is higher because the time packets wait in the transmit queue is longer due to the lower upload bandwidth, 1.5 compared to 25Mb/s.

Another buffer problem is observed around 13:20. At that moment a large file was transferred between two wireless connected hosts. In this case the increased latency is observed at the first link. The file transfer in the local network is saturating the buffers in the combined router/access-point device, resulting into almost 200ms RTT for all connections to the Internet.

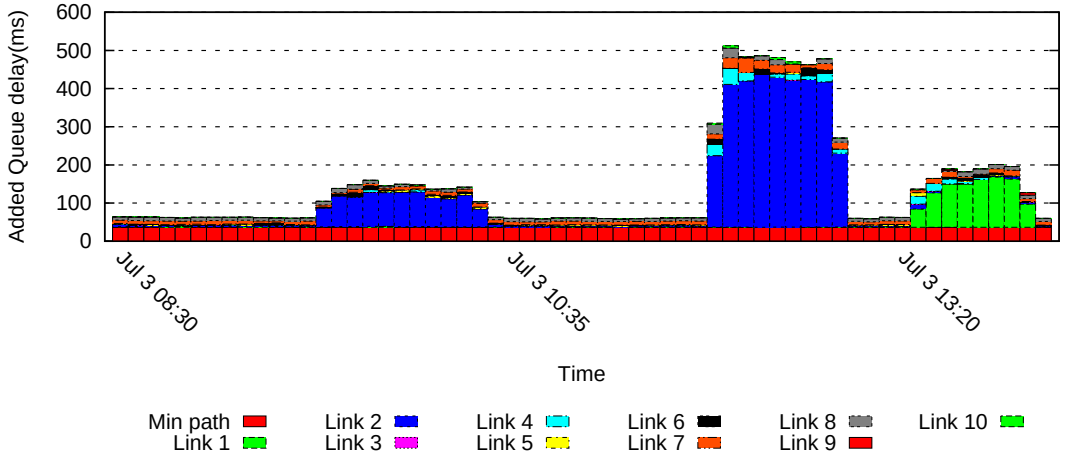


Figure 8: Probing from a consumer grade connection

5.2.2 Chicago, IL - Tokyo

The *queue delay estimate* per link shows that over 100ms of delay is added to the minimum observed round trip time. Latency of this magnitude can break real time applications and the path gets a Bufferbloat Score of 8.29.

Figure 9 shows measurements from a path which is clearly suffering from big, unmanaged buffers. Most of the time more than 100 ms of additional delay is added on top of the minimum round trip time of the path.

The minimum round trip time found during the first measurement was 69.6 ms, while the *queue delay estimate* for this measurement was 126.34 ms. The minimum round trip time observed after

probing with five minute intervals for a period of several days was 57.9 ms while it took no more than five hours to observe a round trip time of 59.3 ms. The *queue delay estimate* during the same period varied between 125 ms and 175 ms. This shows that it is in practice possible to make observations that make it through the entire path with near zero queue delays even in periods with a lot of congestion.

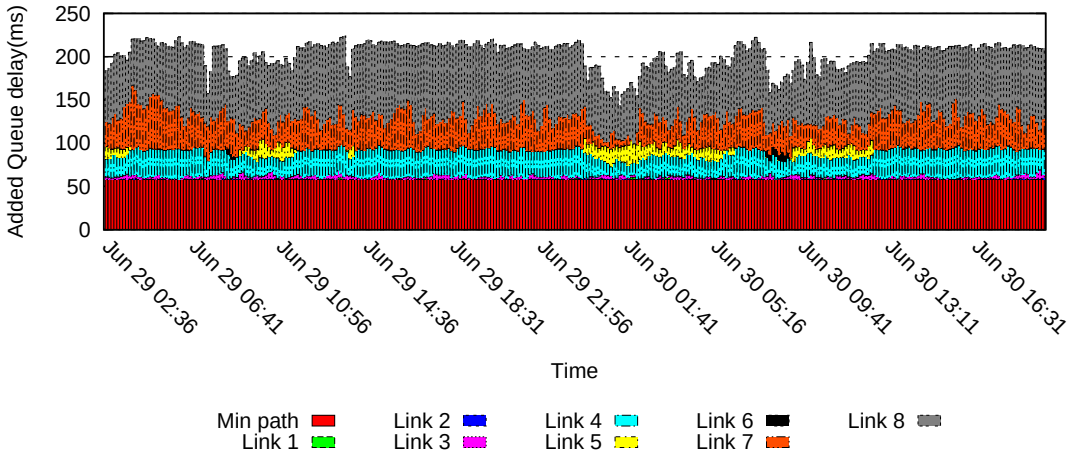


Figure 9: Probing of a path between Chicago and Tokyo

Looking at the queue delay estimates we see that a big part of the additional delay is introduced at link 7 and 8. If we combine this with the network model used in our method we can say something about the router that is causing the bufferbloat effects. Since added delay is detected on both the incoming and outgoing links of the router between link 7 and 8, it is safe to assume that that router is part of the problem in this path.

5.2.3 Other Internet Paths

We have collected measurements for a number of different Internet paths. Most of the links we have measured do not show the same symptoms as the path discussed in section 5.2.2. Added queue delay is observed on most paths, but to a smaller extent and not as constant as the link between Chicago and Tokyo.

As a side note of measuring IPv4 and IPv6 links it seems that the *queue delay estimates* of IPv6 links are slightly lower. We haven't collected enough data to completely explain this observation. One plausible explanation might be that dedicated IPv6 infrastructures are less congested, resulting in a slightly lower queue delay.

6 Conclusions

We came up with a method that can detect large, unmanaged buffers in distant links of network paths. We implemented this method in a tool that we named *Buffchar* [3]. We also proposed a model to quantify the effects. We have successfully applied these methods both in a controlled test environment and on uncontrolled Internet paths. Results from our test setups have shown that it is possible to detect the source of bufferbloat using our method. The different test setups show that it worked correctly using various network configurations.

The *Bufferbloat score* found in our experiments relates to our expectations in the respective scenarios. The values returned comply with the *queue delay estimates* show in the result graph.

6.1 Discussion

Added queue delay can be pinpointed down to individual links in a network path. Determining exactly which queue adds the delay is not possible with our method. Knowledge of the specific topology, configuration and traffic conditions combined with data from our methods can often give enough information to exactly pinpoint the device.

It is rather difficult to come up with a model to quantify a subjective phenomenon as bufferbloat. The model we presented in this paper only quantifies the added queue delay, while the minimum possible round trip time on a path is also important for the quality of a path. A model like this is not worth much before it is used, so we hope to see it adopted and its parameters tuned. *Buffchar* can help raising awareness about the subject and help identifying the source of the issue.

6.2 Future work

- Extend the model to detect layer 2 components

  Our method resides on layer 3 of the OSI model, so layer 2 devices such as switches and MPLS clouds can not be directly identified. Queue delay added by these devices will be detected, not in layer 2 but in the layer 3 devices surrounding them. As these devices are in widespread use on the Internet, it would be useful to be able to identify their queue delays. These delays will in our model show up in the total queue delay for the given link.

- Extend the model to cope with asynchronous links

  Asynchronous connections are common on the Internet and might affect the results of our model. By looking at the TTL value in the returning ICMP packet is it possible detect asymmetry in paths.

- Optimizing score model

  The scoring model we have proposed in this paper is based on a rather limited amount of compared links. Widening the data set on which comparisons are made could be used to fine tune the scoring model.

7 Acknowledgments

This paper is part of our second research project for the System and Network Engineering master at Universiteit van Amsterdam. Equipment available to us here was instrumental in our project.

We would like to thank Michiel Leenaars from NLnet, Freek Dijkstra from SARA and Bert Wijnen from RIPE NCC for their valuable feedback and input during our work with this project.

A Test details

Table 2 describes the different parameters that were applied to the hosts in our test environment. The value in the first column identifies the given router where each configuration change is done by the last two octets of the node's IP addresses in our test environment as described in section 4.

Test / Interface	Baseline			Test 1			Test 2a			Test 2b		
H1: 1.1	500	1000	0									
R1: 1.2	500	1000	0									
R1: 2.1	500	1000	0									
R2: 2.2	500	1000	0									
R2: 3.1	500	1000	0		10			10			100	
R3: 3.2	500	1000	0		10			10			100	
R3: 4.1	500	1000	10			10			10			10
R4: 4.2	500	1000	10			10			10			10
R4: 5.1	500	1000	0									
R5: 5.2	500	1000	0									
R5: 6.1	500	1000	0		10			100			10	
R6: 6.2	500	1000	0		10			100			10	
R6: 7.1	500	1000	0									
H2: 7.2	500	1000	0									

Table 2: Txqueuelen, speed(Kb/s) and added egress latency parameters per test

References

[1] Deepthi Devakti Akkoorath Bert Gijsbers. Performance simulation of buffer bloat in routers. March 2011.

[2] AMS-IX B.V. Traffic statistics ams-ix. `http://www.ams-ix.net/statistics/`.

[3] Harald Kleppe Danny Groenewegen. Buffchar. `https://github.com/hkleppe/Buffchar`, 2011.

[4] Allen Downey. Using pathchar to estimate internet link characteristics. In *In Proceedings of ACM SIGCOMM*, pages 241–250, 1999.

[5] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1:397–413.

[6] Jim Gettys. The criminal mastermind: bufferbloat! 12 2010.

[7] Jim Gettys. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, 15:96, 95, 2011.

[8] R. Govindan and V. Paxson. Estimating router icmp generation delays, 2002.

[9] Van Jacobson. pathchar a tool to infer characteristics of internet paths. Presented at: Mathematical Sciences Research Institute, Berkeley, CA, USA., April 1997.

[10] D. Black K. Ramakrishnan, S. Floyd. The addition of explicit congestion notification (ecn) to ip. Technical report, IETF Request For Comments, 2001. `http://tools.ietf.org/html/rfc3168`.

[11] John Nagle. On packet switches with infinite storage. Technical report, IETF Request For Comments, 1985. `http://tools.ietf.org/html/rfc970`.

[12] Jon Postel. Internet control message protocol. Technical report, IETF Request For Comments, 1981. `http://tools.ietf.org/html/rfc792`.

[13] W.R. Stevens and G.R. Wright. *TCP/IP illustrated: The protocols*. Addison-Wesley professional computing series. Addison-Wesley, 1994.