IP Performance Measurement
Internet-Draft
Intended status: Experimental
Expires: February 14, 2022

C. Paasch
R. Meyer
S. Cheshire
O. Shapira
Apple Inc.

August 13, 2021

Responsiveness under Working Conditions
draft-cpaasch-ippm-responsiveness-00

Abstract

Bufferbloat has been a long-standing problem on the Internet with more than a decade of work on standardizing technical solutions, implementations and testing. However, to this date, bufferbloat is still a very common problem for the end-users. Everyone "knows" that it is "normal" for a video conference to have problems when somebody else on the same home-network is watching a 4K movie.

The reason for this problem is not the lack of technical solutions, but rather a lack of awareness of the problem-space, and a lack of tooling to accurately measure the problem. We believe that exposing the problem of bufferbloat to the end-user by measuring the end-users' experience at a high level will help to create the necessary awareness.

This document is a first attempt at specifying a measurement methodology to evaluate bufferbloat the way common users are experiencing it today, using today's most frequently used protocols and mechanisms to accurately measure the user-experience. We also provide a way to express the bufferbloat as a measure of "Round-trips per minute" (RPM) to have a more intuitive way for the users to understand the notion of bufferbloat.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

For many years, bufferbloat has been known as an unfortunately common issue in todays networks [Bufferbloat]. Solutions like FQ-codel [RFC8289] or PIE [RFC8033] have been standardized and are to some extend widely implemented. Nevertheless, users still suffer from bufferbloat.

The way bufferbloat impacts the user-experience is very subtle. Whenever a network is actively being used at its full capacity, buffers are filling up and create latency for the traffic. These moments of a full buffer may be very brief during a medium-sized file-transfer, like an email-attachment. They create short-lived bursts of latency-spikes that users may experience. An example of this is lag occuring during a video-conference.

While on one side, bufferbloat disrupts the user-experience, its short-lived nature makes it hard to narrow down the problem and make the user sensible to it. Lack of well-known measurement tools and popular measurement platforms add to the "obscure" nature of the bufferbloat problem.

We believe that it is necessary to create a standardized way for measuring the extend of bufferbloat in a network and express it to the user in a user-friendly way. This should help existing measurement tools to add a bufferbloat measurement to their set of metrics. It will also allow to raise the awareness to the problem and shift the focus away from purely quantifying network quality through throughput and idle latency.

In this document, we describe a methodology for measuring bufferbloat and its impact on the user-experience. We create the term "Responsiveness under working conditions" to make it more user-accessible. We focus on using protocols that are most commonly used in end-user use-cases, as performance enhancing proxies and traffic classification for those protocols is very common. It is thus very important to use those protocols for the measurements to avoid focusing on use-cases that are not actually affecting the end-user. Finally, we propose to use "round-trips per minute" as a metric to express the extend of bufferbloat.

2. Measuring is hard

There are several challenges around measuring bufferbloat accurately on the Internet. These challenges are due to different factors. Namely the diverse and dynamic nature of the Internet, the large problem space, and the reproducibility of the measurement.

It is well-known that transparent TCP proxies are widely deployed on port 443 and/or port 80, while less common on other ports. Thus, choice of the port-number to measure bufferbloat has a significant influence on the result. Other factors are the protocols being used. TCP and UDP traffic may take a largely different path on the Internet and be subject to entirely different QoS constraints. Again, bufferbloat measured on UDP vs TCP may be entirely different. Another aspect is the queuing configuration on the bottleneck. It may be that fair-queuing is configured which may "hide" queuing latency that affects other flows.

The Internet is not just diverse; it is changing all the time. Since its inception as a network that can adapt to major outages, the Internet has demonstrated exceptional ability to survive disruptions. At any given moment, the Internet routing is changing to rebalance the traffic, so that a particular local outage will not have a global effect. The cost of such resiliency is the fact that the routing is constantly changing. Daily fluctuations in the demand for the traffic make the bottlenecks ebb and flow. Because of that, measuring the responsiveness during the peak hours is likely to encounter a different bottleneck queue compared to off-peak measurement. It seems that it's best to avoid extending the duration of the test beyond what's needed.

The problem space around the bufferbloat is huge. Traditionally, one thinks of bufferbloat happening on the routers and switches of the Internet. Thus, simply measuring bufferbloat at the transport layer would be sufficient. However, the networking stacks of the clients and servers can also experience huge amounts of bufferbloat. Data sitting in TCP sockets or waiting in the application to be scheduled for sending causes artificial latency, which affects user-experience the same way the "traditional" bufferbloat does.

Finally, measuring bufferbloat requires us to fill the buffers of the bottleneck and when buffer occupancy is at its peak, the latency measurement needs to be done. Achieving this in a reliable and reproducible way is not easy. First, one needs to ensure that buffers are actually full for a sustained period of time to allow for repeated latency measurements in this particular state. Filling of the buffers should happen with standard transport layer traffic - typical for the end-user's use of the network - and thus is subject to the transport's congestion control, implying rate-reduction which reduces the buffering in the network. The amount of bufferbloat is thus constantly fluctuating and a reliable measurement requires to overcome these fluctuations.

3. Goals

There are many different ways on how one can measure bufferbloat. The focus in this document is to capture how bufferbloat affects the user-experience and to provide this as a measurement-tool to non-expert users.

The focus on end-user experience means a number of things:

1. Today’s user-facing Internet traffic is primarily using HTTP/2 over TLS. Thus, the measurement should use that protocol.

   As a side-note: other types of traffic are gaining in popularity (HTTP/3) and/or are already being used widely (RTP). Due to traffic prioritization and QoS rules on the Internet, each of these may experience completely different path-characteristics and should also be measured separately.

2. The Internet is marked by the deployment of countless middleboxes like transparent TCP proxies or traffic prioritization for certain types of traffic. The measurement methodology should allow us to explore all of these as they affect the user-experience. This means, each stage of a user’s interaction with the Internet (DNS-request, TCP-handshake, TLS-handshake, and request/response) needs to be evaluated.

3. User-friendliness of the result means that it should be expressed in a non-technical way to the user. Users commonly look for a single "score" of their performance. This enables the goal of raising awareness to a large user-base on the problem of bufferbloat.

4. Finally, in order for this measurement to be user-friendly to a wide audience it is important that such a measurement finishes within a short time-frame and short being anything below 20 seconds.

4. Measuring Responsiveness

The ability to reliably measure the responsiveness under typical working conditions is predicated by the ability to reliably put the network in a state representative of the said conditions. Once the network has reached the required state, its responsiveness can be measured. The following explains how the former and the latter are achieved.

4.1. Working Conditions

For the purpose of this methodology, "typical working conditions" represent a state of the network, in which the bottleneck node is experiencing ingress and egress flows that are similar to those when used by humans in the typical day-to-day pattern.

While any network can be put momentarily into working condition by the means of a single HTTP transaction, taking measurements requires maintaining such conditions over sufficient time. Thus, measuring the network responsiveness in a consistent way depends on our ability to recreate typical working conditions on demand. The most reliable way to achieve this is by creating multiple large bulk data-transfers in either downstream or upstream direction. Similar to conventional speed-test applications that also create a varying number of streams to measure throughput. Working-conditions does the same. It also requires a way to detect when the network is in a persistent working condition, called "saturation". This can be achieved by monitoring the instantaneous goodput over time. When the goodput stops increasing, it means that a saturation has been reached and that responsiveness can be measured.

Desired properties of the "working condition actuation"

o Should not waste traffic, since the user may be paying for it

o Should finish within a short time-frame to avoid impacting other users on the same network and/or experience varying conditions

4.1.1. Parallel vs Sequential Uplink and Downlink

From an end-user perspective, bufferbloat can happen in both the upstream and the downstream direction. Both paths can be hugely different due to access-link conditions (e.g., 5G downstream and LTE upstream) or the routing in the ISPs. Users sending data to an Internet service will fill the bottleneck on the upstream path to the server and thus expose a potential for bufferbloat to happen at this bottleneck. On the downlink direction any download from an Internet service will encounter a bottleneck and thus exposes another potential for bufferbloat. Thus, when measuring responsiveness under working conditions it is important to consider both, the upstream and the downstream bufferbloat. This opens the door to measure both uplink and downlink in parallel.

Measuring in parallel allows to achieve higher overall confidence in the results within the time constraint of the entire test. For example, if the overall time constraint for the test is 20 seconds, running uplink and downlink sequentially would allow for only 10 seconds of test per direction, while parallel measurement will allow for 20 seconds of testing in both directions.

However, a number caveats come with measuring in parallel: - Half-duplex links may not expose uplink and downlink bufferbloat: A half-duplex link may not allow during parallel measurement to saturate both the uplink and the downlink direction. Thus, bufferbloat in either of the directions may not be exposed during parallel measurement. - Debuggability of the results becomes more obscure: During parallel measurement it is impossible to differentiate on whether the bufferbloat happens in the uplink or the downlink direction.

4.1.2. From single-flow to multi-flow

As described in RFC 6349, a single TCP connection may not be sufficient to saturate a path between a client and a server. On a high-BDP network, traditional TCP window-size constraints of 4MB are often not sufficient to fill the pipe. Additionally, traditional loss-based TCP congestion control algorithms aggressively reacts to packet-loss by reducing the congestion window. This reaction will reduce the queuing in the network, and thus "artificially" make the bufferbloat appear lesser.

The goal of the measurement is to keep the network as busy as possible in a sustained and persistent way. Thus, using multiple TCP connections is needed for a sustained bufferbloat by gradually adding TCP flows until saturation is needed.

4.1.3. Reaching saturation

It is best to detect when saturation has been reached so that the measurement of responsiveness can start with the confidence that the network is sufficiently saturated. For this, we first need to define what "saturation" means. Saturation means not only that the load-bearing connections are utilizing all the capacity, but also that the buffers are completely filled. Thus, this depends highly on the congestion control that is being deployed on the sender-side. Congestion control algorithms like BBR may reach high throughput without causing bufferbloat. (because the bandwidth-detection portion of BBR is effectively seeking the bottleneck capacity)

It is advised to rather use loss-based congestion controls like Cubic to "reliably" ensure that the buffers are filled.

An indication of saturation is when the observed goodput is no more increasing even as connections are being added to the pool of load-generating connections. An additional indication is the presence of packet-loss or ECN-marks signaling a congestion or even a full buffer of the bottleneck link.

4.1.4. Final algorithm

The following is a proposal for an algorithm to reach saturation of a network by using HTTP/2 upload (POST) or download (GET) requests of infinitely large files. The algorithm is the same for upload and download and thus we will use the same term "load-bearing connection" for either of them.

The algorithm takes into account that throughput gradually increases as TCP connections go through their TCP slow-start phase. Throughput-increase eventually stalls for a constant number of TCP-connections - usually due to receive-window limitations. At that point, the only means to further increase throughput is by adding more TCP connections to the pool of load-bearing connections. This will then either result in a further increase in throughput, or throughput will remain stable. In the latter case, this means that saturation has been reached and - more importantly - is stable.

In detail, the steps of the algorithm are the following

- Create 4 load-bearing connections
- At each 1-second interval:
  - Compute "instantaneous aggregate" goodput which is the number of bytes received within the last second.
  - Compute moving average as the last 4 "instantaneous aggregate goodput" measurements
  - If moving average > "previous" moving average + 5%:
    - We did not yet reach saturation, but if we haven't added more flows for 4 seconds, add 4 more flows to the mix.
  - Else, we reached saturation for the current flow-count.
    - If we added flows and for 4 seconds the moving average throughput did not change: We reached stable saturation
    - Else, add more flows

Note: It may be tempting to steer the algorithm through an initial base-RTT measurement and adjust the intervals as a function of the RTT. However, experiments have shown that this makes the saturation-detection extremely unstable in low-RTT environments. When the "unloaded" RTT is in the single-digit milli-second range, while under load the network's RTT increases to more than a hundred milliseconds, the intervals become much too low to accurately drive the algorithm.

4.2. Measuring Responsiveness

Once the network is in consistent working conditions, the network's responsiveness can be measured. As the focus of our responsiveness metric is to evaluate a real user-experience we focus on measuring the different stages of a separate network transaction as well as measuring on the load-bearing connections themselves.

Two aspects are being measured with this approach :

1. How the network handles new connections and their different stages (DNS-request, TCP-handshake, TLS-handshake, HTTP/2 request/response) while being under working conditions. E.g., the presence of fair-queueing on the bottleneck will drastically improve the experience on these connections.

2. How the network and the client/server networking stack handles the latency on the load-bearing connections themselves. E.g., Smart queuing techniques on the bottleneck will allow to keep the latency within a reasonable limit in the network and buffer-reducing techniques like TCP_NOTSENT_LOWAT makes sure the client and server TCP-stack is not a source of significant latency.

To measure the former, we send a DNS-request, establish a TCP-connection on port 443, establish a TLS-context using TLS1.3 and send an HTTP2 GET request for an object of a single byte large. This measurement will be repeated multiple times for accuracy. Each of these stages allows to collect a single latency measurement that can then be factored into the responsiveness computation.

To measure the latter, on the load-bearing connections (that uses HTTP/2) a GET request is multiplexed. This GET request is for a 1-byte object. This allows to measure the end-to-end latency on the connections that are using the network at full speed.

4.2.1. Aggregating Round-trips per Minute

The above described method will produce 5 sets of measurement results, namely: DNS-handshake, TCP-handshake, TLS handshake, HTTP/2 request/response on separate connections, HTTP/2 request/response on load-bearing connections. Each of these sets of numbers focuses on a specific aspect of a user’s interaction with the network. In order to expose a single "Responsiveness" number to the user the weighting among these sets needs to be decided. In our first iteration we give an equal weight to each of these measurements.

Finally, the resulting latency needs to be exposed to the users. Users have been trained to accept metrics that have a notion of "The higher the better". Latency measuring in units of seconds however is "the lower the better". Thus, converting the latency measurement to a frequency allows using the familiar notion of "The higher the better". The term frequency has a very technical connotation. What we are effectively measuring is the number of round-trips from the user's device to the server endpoint that can be done within a unit of time. This leads to the notion of Round-trips per Minute. It has the advantage that the range of values is within a reasonable 50 to 3000 Round-trips per Minute. It can also be abbreviated to "RPM" which is a wink to the "revolutions per minute" that we are used to in cars.

Thus, our unit of measure is "Round-trip per Minute" (RPM) that expresses responsiveness under working conditions.

4.2.2. Statistical Confidence

It remains an open question as to how many repetitions of the above described latency measurements a tool would need to execute. One could imagine a computation of the variance and confidence interval that would drive the number of measurements and balance the accuracy with the speed of the measurement itself.

5. Protocol Specification

By using standard protocols that are most commonly used by end-users, no new protocol needs to be specified. However, both client and servers need capabilities to execute this kind of measurement as well as a standard to flow to provision the client with the necessary information.

First, the capabilities of both the client and the server: It is expected that both hosts support HTTP/2 over TLS 1.3. That the client is able to send a GET-request and a POST. The server needs the ability to serve both of these HTTP commands. Further, the server endpoint is accessible through a hostname that can be resolved through DNS. Finally, the server has the ability to provide content upon a GET-request.

Given those capabilities, the server is expected to provide 4 URLs/ responses:

1. A config URL/response: This is the configuration file/format used by the client. It's a simple JSON file format that points the client at the various URLs mentioned below. All of the fields are required except "test_endpoint". If the service-procier can pin all of the requests for a test run to a specific node in the service (for a particular run), they can specify that node's name in the "test_endpoint" field. It's preferred that pinning of some sort is available. This is to ensure the measurement is against the same paths and not switching hosts during a test run (ie moving from near POP A to near POP B) Sample content of this JSON would be:

```
{
  "version": 1,
  "urls": {
    "small_https_download_url": "https://example.apple.com/api/v1/small",
    "large_https_download_url": "https://example.apple.com/api/v1/large",
    "https_upload_url": "https://example.apple.com/api/v1/upload"
  },
  "test_endpoint": "hostname123.cdnprovider.com"
}
```

2. A "small" URL/response: This needs to serve a status code of 200 and 1 byte in the body. The actual body content is irrelevant.

3. A "large" URL/response: This needs to serve a status code of 200 and a body size of at least 8GB. The body can be bigger, and will need to grow as network speeds increases over time. The actual body content is irrelevant. The client will probably never completely download the object.

4. An "upload" URL/response: This needs to handle a POST request with an arbitrary body size. Nothing needs to be done with the payload, it should be discarded.

Given those 4 services provided by the server, the client can bootstrap the responsiveness measurement by querying the JSON configuration. Upon which it has the URLs for creating the load-bearing connections in the upstream and downstream direction as well as the small object for the latency measurements.

6. Security Considerations

TBD

7. IANA Considerations

TBD

8. Acknowledgments

TBD

9. Informative References

[Bufferbloat]
Gettys, J. and K. Nichols, "Bufferbloat: Dark Buffers in the Internet", Communications of the ACM, Volume 55, Number 1 (2012) , n.d..

[RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <https://www.rfc-editor.org/info/rfc8033>.

[RFC8289] Nichols, K., Jacobson, V., McGregor, A., Ed., and J. Iyengar, Ed., "Controlled Delay Active Queue Management", RFC 8289, DOI 10.17487/RFC8289, January 2018, <https://www.rfc-editor.org/info/rfc8289>.

Authors' Addresses

Christoph Paasch
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: cpaasch@apple.com

Randall Meyer
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: rrm@apple.com

Stuart Cheshire
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: cheshire@apple.com

Omer Shapira
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: oesh@apple.com