IP Performance Measurement
Internet-Draft
Intended status: Experimental
Expires: January 8, 2023

C. Paasch
R. Meyer
S. Cheshire
O. Shapira
Apple Inc.
M. Mathis
Google, Inc

July 07, 2022

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

Abstract

For many years, a lack of responsiveness, variously called lag, latency, or bufferbloat, has been recognized as an unfortunate, but common symptom in today's networks. Even after a decade of work on standardizing technical solutions, it remains a common problem for the end users.

Everyone "knows" that it is "normal" for a video conference to have problems when somebody else at home is watching a 4K movie or uploading photos from their phone. However, there is no technical reason for this to be the case. In fact, various queue management solutions (fq_codel, cake, PIE) have solved the problem.

Our networks remain unresponsive, not from a lack of technical solutions, but rather a lack of awareness of the problem. We believe that creating a tool whose measurement matches people's every day experience will create the necessary awareness, and result in a demand for products that solve the problem.

This document specifies the "RPM Test" for measuring responsiveness. It uses common protocols and mechanisms to measure user experience especially when the network is under working conditions. The measurement is expressed as "Round-trips Per Minute" (RPM) and should be included with throughput (up and down) and idle latency as critical indicators of network quality.

working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 8, 2023.

Table of Contents

1. Introduction

For many years, a lack of responsiveness, variously called lag, latency, or bufferbloat, has been recognized as an unfortunate, but common symptom in today's networks [Bufferbloat]. Solutions like fq_codel [RFC8290] or PIE [RFC8033] have been standardized and are to some extent widely implemented. Nevertheless, people still suffer from bufferbloat.

Although significant, the impact on user experience can be transitory - that is, its effect is not always present. Whenever a network is actively being used at its full capacity, buffers can fill up and create latency for traffic. The duration of those full buffers may be brief: a medium-sized file transfer, like an email attachment or uploading photos, can create bursts of latency spikes. An example of this is lag occurring during a videoconference, where a connection is briefly shown as unstable.

These short-lived disruptions make it hard to narrow down the cause. We believe that it is necessary to create a standardized way to measure and express responsiveness.

Existing network measurement tools could incorporate a responsiveness measurement into their set of metrics. Doing so would also raise the awareness of the problem and make the standard "network quality measures" of throughput, idle latency, and responsiveness.

1.1. Terminology

A word about the term "bufferbloat" - the undesirable latency that comes from a router or other network equipment buffering too much data. This document uses the term as a general description of bad latency, using more precise wording where warranted.

"Latency" is a poor measure of responsiveness, since it can be hard for the general public to understand. The units are unfamiliar ("what is a millisecond?") and counterintuitive ("100 msec - that sounds good - it's only a tenth of a second!").

Instead, we create the term "Responsiveness under working conditions" to make it clear that we are measuring all, not just idle,

conditions, and use "round-trips per minute" as the metric. The advantage of round-trips per minute are two-fold: First, it allows for a metric that is "the higher the better". This kind of metric is often more intuitive for end-users. Second, the range of the values tends to be around the 4-digit integer range which is also a value easy to compare and read, again allowing for a more intuitive use. Finally, we abbreviate the measurement to "RPM", a wink to the "revolutions per minute" that we use for cars.

This document defines an algorithm for the "RPM Test" that explicitly measures responsiveness under working conditions.

2. Design Constraints

There are many challenges around measurements on the Internet. They include the dynamic nature of the Internet, the diverse nature of the traffic, the large number of devices that affect traffic, and the difficulty of attaining appropriate measurement conditions.

Internet paths are changing all the time. Daily fluctuations in the demand make the bottlenecks ebb and flow. To minimize the variability of routing changes, it’s best to keep the test duration relatively short.

TCP and UDP traffic, or traffic on ports 80 and 443, may take significantly different paths on the Internet and be subject to entirely different Quality of Service (QoS) treatment. A good test will use standard transport layer traffic - typical for people’s use of the network - that is subject to the transport’s congestion control that might reduce the traffic’s rate and thus its buffering in the network.

Traditionally, one thinks of bufferbloat happening on the routers and switches of the Internet. However, the networking stacks of the clients and servers can have huge buffers. Data sitting in TCP sockets or waiting for the application to send or read causes artificial latency, and affects user experience the same way as "traditional" bufferbloat.

Finally, it is important to note that queueing only happens behind a slow "bottleneck" link in the network, and only occurs when sufficient traffic is present. The RPM Test must ensure that buffers are actually full for a sustained period, and only then make repeated latency measurements in this particular state.

3. Goals

The algorithm described here defines an RPM Test that serves as a good proxy for user experience. This means:

1. Today's Internet traffic primarily uses HTTP/2 over TLS. Thus, the algorithm 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). Traffic prioritization and QoS rules on the Internet may subject traffic to completely different paths: these could 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 RPM Test must take into account their effect on TCP-handshake [RFC0793], TLS-handshake, and request/response.

3. The test result should be expressed in an intuitive, nontechnical form.

4. Finally, to be useful to a wide audience, the measurement should finish within a short time frame. Our target is 20 seconds.

4. Measuring Responsiveness Under Working Conditions

To make an accurate measurement, the algorithm must reliably put the network in a state that represents those "working conditions". During this process, the algorithm measures the responsiveness of the network. The following explains how the former and the latter are achieved.

4.1. Working Conditions

There are many different ways to define the state of "working conditions" to measure responsiveness. There is no one true answer to this question. It is a tradeoff between using realistic traffic patterns and pushing the network to its limits.

In this document we aim to generate a realistic traffic pattern by using standard HTTP transactions but exploring the worst-case scenario by creating multiple of these transactions and using very large data objects in these HTTP transactions.

This allows to create a stable state of working conditions during which the network is used at its nearly full capacity, without generating DoS-like traffic patterns (e.g., UDP flooding).

Finally, as end-user usage of the network evolves to newer protocols and congestion control algorithms, it is important that the working conditions also can evolve to continuously represent a realistic traffic pattern.

4.1.1. From single-flow to multi-flow

A single TCP connection may not be sufficient to reach the capacity of a path. For example, the 4MB constraints on TCP window size constraints may not fill the pipe. Additionally, traditional loss-based TCP congestion control algorithms react aggressively to packet loss by reducing the congestion window. This reaction (intended by the protocol design) decreases the queueing within the network, making it hard to reach the path's capacity.

The goal of the RPM Test is to keep the network in working conditions in a sustained and persistent way. It uses multiple TCP connections and gradually adds more TCP flows until full link utilization is reached.

4.1.2. Parallel vs Sequential Uplink and Downlink

Poor responsiveness can be caused by queues in either (or both) the upstream and the downstream direction. Furthermore, both paths may differ significantly due to access link conditions (e.g., 5G downstream and LTE upstream) or the routing changes within the ISPs. To measure responsiveness under working conditions, the algorithm must explore both directions.

One approach could be to measure responsiveness in the uplink and downlink in parallel. It would allow for a shorter test run-time.

However, a number of caveats come with measuring in parallel:

- Half-duplex links may not permit simultaneous uplink and downlink traffic. This means the test might not reach the path's capacity in both directions at once and thus not expose all the potential sources of low responsiveness.

- Debuggability of the results becomes harder: During parallel measurement it is impossible to differentiate whether the observed latency happens in the uplink or the downlink direction.

Thus, we recommend testing uplink and downlink sequentially. Parallel testing is considered a future extension.

4.1.3. Reaching full link utilization

The RPM Test gradually increases the number of TCP connections and measures "goodput" - the sum of actual data transferred across all connections in a unit of time. When the goodput stops increasing, it means that the network is used at its full capacity. At this point we are creating the worst-case scenario within the limits of the realistic traffic pattern.

The algorithm notes that throughput gradually increases until TCP connections complete their TCP slow-start phase. At that point, throughput eventually stalls usually due to receive window limitations. The only means to further increase throughput is by adding more TCP connections to the pool of load-generating connections. If new connections leave the throughput the same, full link utilization has been reached and - more importantly - the working condition is stable.

4.1.4. Final "Working Conditions" Algorithm

The following algorithm reaches working conditions 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 uses the same term "load-generating connection" for each. The actions of the algorithm take place at regular intervals. For the current draft the interval is defined as one (1) second.

Where

- i: The index of the current interval. i is initialized to 0 when the algorithm begins and increases by one for each interval.

- instantaneous aggregate goodput at interval p: The number of total bytes of data transferred within interval p. If p is less than 0, the number of total bytes of data transferred within the interval is considered to be 0.

- moving average aggregate goodput at interval p: The average of the number of total bytes of data transferred in the instantaneous average aggregate goodput at intervals p - x, for all 0<=x<4.

- moving average stability during the period between intervals b and e: Whether or not the differences between the moving average aggregate goodput at interval x and the moving average aggregate goodput at interval x+1 (for all b<=x<e) is less than 5%.

the steps of the algorithm are:

- Create four (4) load-generating connections.
- At each interval:
  - Compute the instantaneous aggregate goodput at interval i.
  - Compute the moving average aggregate goodput at interval i.
  - If the moving average aggregate goodput at interval i is more than a 5% increase over the moving average aggregate goodput at interval i - 1, the network has not yet reached full link utilization.
    - If no load-generating connections have been added within the last four (4) intervals, add four (4) more load-generating connections.
  - Else, the network has reached full link utilization with the existing load-generating connections. The current state is a candidate for stable working conditions.
    - If a) there have been load-generating connections added in the past four (4) intervals and b) there has been moving average stability during the period between intervals i-4 and i, then the network has reached full link utilization and the algorithm terminates.
    - Otherwise, add four (4) more load-generating connections.

In Section 3, it is mentioned that one of the goals is that the test finishes within 20 seconds. It is left to the implementation what to do when full link utilization is not reached within that time-frame. For example, an implementation might gather a provisional responsiveness measurement or let the test run for longer.

4.2. Measuring Responsiveness

Once the network is in a consistent working conditions, the RPM Test must "probe" the network multiple times to measure its responsiveness.

Each RPM Test probe measures:

1. The responsiveness of the different steps to create a new connection, all during working conditions.

To do this, the test measures the time needed to establish a TCP connection on port 443, establish a TLS context using TLS1.3 [RFC8446], and send an HTTP/2 GET request for a a one-byte object and wait for the response to be fully received. It repeats these steps multiple times for accuracy.

2. The responsiveness of the network and the client/server networking stacks for the load-generating connections themselves.

   To do this, the load-generating connections multiplex an HTTP/2 GET request for a one-byte object to get the end-to-end latency on the connections that are using the network at full speed.

4.2.1. Aggregating the Measurements

The algorithm produces sets of 4 times for each probe, namely: TCP handshake, TLS handshake, HTTP/2 request/response on separate (idle) connections, HTTP/2 request/response on load-generating connections. This fine-grained data is useful, but not necessary for creating a useful metric.

To create a single "Responsiveness" (e.g., RPM) number, this first iteration of the algorithm gives an equal weight to each of these values. That is, it sums the five time values for each probe, and divides by the total number of probes to compute an average probe duration. The reciprocal of this, normalized to 60 seconds, gives the Round-trips Per Minute (RPM).

4.2.2. Statistical Confidence

The number of probes necessary for statistical confidence is an open question. 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. Interpreting responsiveness results

The described methodology uses a high-level approach to measure responsiveness. By executing the test with regular HTTP requests a number of elements come into play that will influence the result. Contrary to more traditional measurement methods the responsiveness metric is not only influenced by the properties of the network but can significantly be influenced by the properties of the client and the server implementations. This section describes how the different elements influence responsiveness and how a user may differentiate them when debugging a network.

5.1. Elements influencing responsiveness

Due to the HTTP-centric approach of the measurement methodology a number of factors come into play that influence the results. Namely, the client-side networking stack (from the top of the HTTP-layer all the way down to the physical layer), the network (including potential transparent HTTP "accelerators"), and the server-side networking stack. The following outlines how each of these contributes to the responsiveness.

5.1.1. Client side influence

As the driver of the measurement, the client-side networking stack can have a large influence on the result. The biggest influence of the client comes when measuring the responsiveness in the uplink direction. Load-generation will cause queue-buildup in the transport layer as well as the HTTP layer. Additionally, if the network's bottleneck is on the first hop, queue-buildup will happen at the layers below the transport stack (e.g., NIC firmware).

Each of these queue build-ups may cause latency and thus low responsiveness. A well-designed networking stack would ensure that queue-buildup in the TCP layer layer is kept at a bare minimum with solutions like TCP_NOTSENT_LOWAT [draft-ietf-tcpm-rfc793bis]. At the HTTP/2 layer it is important that the load-generating data is not interfering with the latency-measuring probes. For example, the different streams should not be stacked one after the other but rather be allowed to be multiplexed for optimal latency. The queue-buildup at these layers would only influence latency on the probes that are sent on the load-generating connections.

Below the transport layer many places have a potential queue build-up. It is important to keep these queues at reasonable sizes or that they implement techniques like FQ-Codel. Depending on the techniques used at these layers, the queue build-up can influence latency on probes sent on load-generating connections as well as separate connections. If flow-queuing is used at these layers, the impact on separate connections will be negligible.

5.1.2. Network influence

The network obviously is a large driver for the responsiveness result. Propagation delay from the client to the server as well as queuing in the bottleneck node will cause latency. Beyond these traditional sources of latency, other factors may influence the results as well. Many networks deploy transparent TCP Proxies, firewalls doing deep packet-inspection, HTTP "accelerators",... As

the methodology relies on the use of HTTP/2, the responsiveness metric will be influenced by such devices as well.

The network will influence both kinds of latency probes that the responsiveness tests sends out. Depending on the network's use of Smart Queue Management and whether this includes flow-queuing or not, the latency probes on the load-generating connections may be influenced differently than the probes on the separate connections.

5.1.3. Server side influence

Finally, the server-side introduces the same kind of influence on the responsiveness as the client-side. With the difference that the responsiveness will be impacted during the downlink load generation.

5.2. Root-causing Responsiveness

Once an RPM result has been generated one might be tempted to try to localize the source of a potential low responsiveness. The responsiveness measurement is however aimed at providing a quick, top-level view of the responsiveness under working conditions the way end-users experience it. Localizing the source of low responsiveness involves however a set of different tools and methodologies.

Nevertheless, the responsiveness test allows to gain some insight into what the source of the latency is. The previous section described the elements that influence the responsiveness. From there it became apparent that the latency measured on the load-generating connections and the latency measured on separate connections may be different due to the different elements.

For example, if the latency measured on separate connections is much less than the latency measured on the load-generating connections, it is possible to narrow down the source of the additional latency on the load-generating connections. As long as the other elements of the network don't do flow-queueing, the additional latency must come from the queue build-up at the HTTP and TCP layer. This is because all other bottlenecks in the network that may cause a queue build-up will be affecting the load-generating connections as well as the separate latency probing connections in the same way.

6. RPM Test Server API

The RPM measurement uses standard protocols: no new protocol is defined.

Both the client and the server MUST support HTTP/2 over TLS 1.3. The client MUST be able to send a GET request and a POST. The server

MUST be able to respond to both of these HTTP commands. The server MUST have the ability to provide content upon a GET request. Both client and server SHOULD use loss-based congestion controls like Cubic. The server MUST use a packet scheduling algorithm that minimizes internal queueing to avoid affecting the client's measurement.

The server MUST respond to 4 URLs:

1. A "small" URL/response: The server must respond with a status code of 200 and 1 byte in the body. The actual body content is irrelevant. The server SHOULD specify the content-type as application/octet-stream.

2. A "large" URL/response: The server must respond with a status code of 200 and a body size of at least 8GB. The server SHOULD specify the content-type as application/octet-stream. The body can be bigger, and may need to grow as network speeds increases over time. The actual body content is irrelevant. The client will probably never completely download the object, but will instead close the connection after reaching working condition and making its measurements.

3. An "upload" URL/response: The server must handle a POST request with an arbitrary body size. The server should discard the payload.

4. A configuration URL that returns a JSON [RFC8259] object with the information the client uses to run the test (sample below). The server SHOULD specify the content-type as application/json. Sample JSON:

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

The client begins the responsiveness measurement by querying for the JSON configuration. This supplies the URLs for creating the load-generating connections in the upstream and downstream direction as well as the small object for the latency measurements.

7. Security Considerations

TBD

8. IANA Considerations

TBD

9. Acknowledgments

We would like to thank Rich Brown for his editorial pass over this I-D. We also thank Erik Auerswald and Will Hawkins for their constructive feedback on the I-D.

10. 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..

[draft-ietf-tcpm-rfc793bis] Eddy, W., "Transmission Control Protocol (TCP) Specification", Internet Engineering Task Force , n.d..

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <https://www.rfc-editor.org/info/rfc8290>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Appendix A. Example Server Configuration

This section shows fragments of sample server configurations to host an responsiveness measurement endpoint.

A.1. Apache Traffic Server

Apache Traffic Server starting at version 9.1.0 supports configuration as a responsiveness server. It requires the generator and the statichit plugin.

The sample configuration file then is:

```
map https://networkquality.example.com/api/v1/large \
    http://localhost/cache/4294967296/ \
    @plugin=generator.so

map https://networkquality.example.com/api/v1/carry/small \
    http://localhost/cache/1/ \
    @plugin=generator.so

map https://networkquality.example.com/api/v1/carry/config \
    http://localhost/ \
    @plugin=statichit.so @pparam=--file-path=config.example.com.json \
    @pparam=--mime-type=application/json

map https://networkquality.example.com/api/v1/carry/slurp \
    http://localhost/ \
    @plugin=generator.so
```

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

Matt Mathis
Google, Inc
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America

Email: mattmathis@google.com