IP Performance Measurement
Internet-Draft
Intended status: Standards Track
Expires: 14 September 2023

C. Paasch
R. Meyer
S. Cheshire
O. Shapira
Apple Inc.
W. Hawkins
University of Cincinnati
M. Mathis
Google, Inc

13 March 2023

Responsiveness under Working Conditions
draft-ietf-ippm-responsiveness-01

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 and deployment of its solutions. We believe that creating a tool that measures the problem and matches people’s everyday experience will create the necessary awareness, and result in a demand for solutions.

This document specifies the "Responsiveness Test" for measuring responsiveness. It uses common protocols and mechanisms to measure user experience specifically 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.

This Internet-Draft will expire on 14 September 2023.

Copyright Notice

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 visible to the user. 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.

Including the responsiveness-under-working-conditions test among other measurements of network quality (e.g., throughput and idle latency) would raise awareness of the problem and establish the expectation among users that their network providers deploy solutions.

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, because 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 define 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 unit. The advantage of using round-trips per minute as the unit are two-fold: First, it allows for a unit that is "the higher the better". This kind of unit 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 unit to "RPM", a wink to the "revolutions per minute" that we use for car engines.

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

2. Design Constraints

There are many challenges to defining measurements of the Internet: the dynamic nature of the Internet, the diverse nature of the traffic, the large number of devices that affect traffic, the difficulty of attaining appropriate measurement conditions, diurnal traffic patterns, and changing routes.

In order to minimize the effects of these challenges, 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 over the network between source and destination 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 layer's congestion control algorithms that might reduce the traffic's rate and thus its buffering in the network.

Traditionally, one thinks of bufferbloat happening in the network, i.e., on 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 in-network bufferbloat.

Finally, it is crucial to recognize that significant queueing only happens on entry to the lowest-capacity (or "bottleneck") hop on a network path. For any flow of data between two endpoints there is

always one hop along the path where the capacity available to that flow at that hop is the lowest among all the hops of that flow's path at that moment in time. It is important to understand that the existence of a lowest-capacity hop on a network path and a buffer to smooth bursts of data is not itself a problem. In a heterogeneous network like the Internet it is inevitable that there must necessarily be some hop along the path with the lowest capacity for that path. If that hop were to be improved, then some other hop would become the new lowest-capacity hop for that path. In this context a "bottleneck" should not be seen as a problem to be fixed, because any attempt to "fix" the bottleneck is futile -- such a "fix" can never remove the existence of a bottleneck on a path; it just moves the bottleneck somewhere else. Arguably, this heterogeneity of the Internet is one of its greatest strengths. Allowing individual technologies to evolve and improve at their own pace, without requiring the entire Internet to change in lock-step, has enabled enormous improvements over the years in technologies like DSL, cable modems, Ethernet, and Wi-Fi, each advancing independently as new developments became ready. As a result of this flexibility we have moved incrementally, one step at a time, from 56kb/s dial-up modems in the 1990s to Gb/s home Internet service and Gb/s wireless connectivity today.

Note that in a shared datagram network, conditions do not remain static. The hop that is the current bottleneck may change from moment to moment. For example, changes in simultaneous traffic may result in changes to a flow's share of a given hop. A user moving around may cause the Wi-Fi transmission rate to vary widely, from a few Mb/s when far from the Access Point, all the way up to Gb/s or more when close to the Access Point.

Consequently, if we wish to enjoy the benefits of the Internet's great flexibility, we need software that embraces and celebrates this diversity and adapts intelligently to the varying conditions it encounters.

Because significant queueing only happens on entry to the bottleneck hop, the queue management at this critical hop of the path almost entirely determines the responsiveness of the entire flow. If the bottleneck hop's queue management algorithm allows an excessively large queue to form, this results in excessively large delays for packets sitting in that queue awaiting transmission, significantly degrading overall user experience.

In order to discover the depth of the buffer at the bottleneck hop, the proposed Responsiveness Test mimics normal network operations and data transfers, with the goal of filling the bottleneck buffer to capacity, and then measures the resulting end-to-end latency under

these so-called working conditions. A well-managed bottleneck queue keeps its occupancy under control, resulting in consistently low round-trip times and consistently good responsiveness. A poorly managed bottleneck queue will not.

3. Goals

The algorithm described here defines a Responsiveness Test that serves as a good proxy for user experience. Therefore:

1. Because today's Internet traffic primarily uses HTTP/2 over TLS, the test's 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. Because the Internet is marked by the deployment of countless middleboxes like transparent TCP proxies or traffic prioritization for certain types of traffic, the Responsiveness Test algorithm must take into account their effect on TCP-handshake [RFC0793], TLS-handshake, and request/response.

3. Because the goal of the test is to educate end users, the results should be expressed in an intuitive, nontechnical form and not commit the user to spend a significant amount of their time (we target 20 seconds).

4. Measuring Responsiveness Under Working Conditions

Overall, the test to measure responsiveness under working conditions proceeds in two steps:

1. Put the network connection into "working conditions"

2. Measure responsiveness of the network.

The following explains how the former and the latter are achieved.

4.1. Working Conditions

What are _the_ conditions that best emulate how a network connection is used? 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.

The Responsiveness Test defines working conditions as the condition where the path between the measuring endpoints is utilized at its end-to-end capacity and the queue at the bottleneck link is at (or beyond) its maximum occupancy. Under these conditions, the network connection's responsiveness will be at its worst.

The Responsiveness Test algorithm for reaching working conditions combines multiple standard HTTP transactions with very large data objects according to realistic traffic patterns to create these conditions.

This allows to create a stable state of working conditions during which the bottleneck of the path between client and server has its buffer filled up entirely, without generating DoS-like traffic patterns (e.g., intentional UDP flooding). This creates a realistic traffic mix representative of what a typical user's network experiences in normal operation.

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. Single-flow vs multi-flow

A single TCP connection may not be sufficient to reach the capacity and full buffer occupancy of a path quickly. Using a 4MB receive window, over a network with a 32 ms round-trip time, a single TCP connection can achieve up to 1Gb/s throughput. Additionally, deep buffers along the path between the two endpoints may be significantly larger than 4MB. TCP allows larger receive window sizes, up to 1GB. However, most transport stacks aggressively limit the size of the receive window to avoid consuming too much memory.

Thus, the only way to achieve full capacity and full buffer occupancy on those networks is by creating multiple connections, allowing to actively fill the bottleneck's buffer to achieve maximum working conditions.

Even if a single TCP connection would be able to fill the bottleneck's buffer, it may take some time for a single TCP connection to ramp up to full speed. One of the goals of the Responsiveness Test is to help the user quickly measure their network. As a result, the test must load the network, take its measurements, and then finish as fast as possible.

Finally, 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 harder to determine the depth of the bottleneck queue reliably.

The purpose of the Responsiveness Test is not to productively move data across the network in a useful way, the way a normal application does. The purpose of the Responsiveness Test is, as quickly as possible, to simulate a representative traffic load as if real applications were doing sustained data transfers, measure the resulting round-trip time occurring under those realistic conditions. Because of this, using multiple simultaneous parallel connections allows the Responsiveness Test to complete its task more quickly, in a way that overall is less disruptive and less wasteful of network capacity than a test using a single TCP connection that would take longer to bring the bottleneck hop to a stable saturated state.

In this document, we impose an upper bound on the number of parallel load-generating connections to 16.

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 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 restriction 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. Achieving Full Buffer Utilization

The Responsiveness Test gradually increases the number of TCP connections (known as load-generating connections) and measures "goodput" (the sum of actual data transferred across all connections in a unit of time) continuously. By definition, once goodput is maximized, buffers will start filling up, creating the "standing queue" that is characteristic of bufferbloat. At this moment the test starts measuring the responsiveness until it, too, reaches saturation. At this point we are creating the worst-case scenario within the limits of the realistic traffic pattern.

The algorithm notes that throughput increases rapidly until TCP connections complete their TCP slow-start phase. At that point, throughput eventually stalls, often due to receive window limitations, particularly in cases of high network bandwidth, high network round-trip time, low receive window size, or a combination of all three. 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.

4.2. Test parameters

A number of parameters serve as input to the test methodology. The following lists their names, default values and explanation. Hereafter the detailed description of the methodology will explain how these parameters are being used. Experience has shown that these parameters allow for a low runtime and accurate results among a wide range of environments.

Name	Explanation	Default Value
MAD	Moving average distance (number of intervals to take into account for the moving average)	4
TMP	Trimmed Mean Percentage to be removed	95%
SDT	Standard Deviation Tolerance for stability detection	5%
ID	Interval duration at which the algorithm reevaluates stability	1 second
MNP	Maximum number of parallel transport-layer connections	16
MPS	Maximum responsiveness probes per second	100
PTC	Percentage of Total Capacity the probes are allowed to consume	5%

Table 1

4.3. Measuring Responsiveness

Measuring responsiveness while achieving working conditions is a process of continuous measurement. It requires a sufficiently large sample-size to have confidence in the results.

The measurement of the responsiveness happens by sending probe-requests. There are two types of probe requests:

1. A HTTP GET request on a separate connection ("foreign probes"). This test mimics the time it takes for a web browser to connect to a new web server and request the first element of a web page (e.g., "index.html"), or the startup time for a video streaming client to launch and begin fetching media.

2. A HTTP GET request multiplexed on the load-generating connections ("self probes"). This test mimics the time it takes for a video streaming client to skip ahead to a different chapter in the same video stream, or for a navigation client to react and fetch new map tiles when the user scrolls the map to view a different area. In a well functioning system fetching new data over an existing connection should take less time than creating a brand new TLS connection from scratch to do the same thing.

Foreign probes will provide 3 sets of data-points. First, the duration of the TCP-handshake (noted hereafter as tcp_f). Second, the TLS round-trip-time (noted tls_f). For this, it is important to note that different TLS versions have a different number of round-trips. Thus, the TLS establishment time needs to be normalized to the number of round-trips the TLS handshake takes until the connection is ready to transmit data. And third, the HTTP elapsed time between issuing the GET request for a 1-byte object and receiving the entire response (noted http_f).

Self probes will provide a single data-point for the duration of time between when the HTTP GET request for the 1-byte object is issued on the load-generating connection and the full HTTP response has been received (noted http_s).

tcp_f, tls_f, http_f and http_s are all measured in milliseconds.

The more probes that are sent, the more data available for calculation. In order to generate as much data as possible, the Responsiveness Test specifies that a client issue these probes regularly. There is, however, a risk that on low-capacity networks the responsiveness probes themselves will consume a significant amount of the capacity. Because the test mandates first saturating capacity before probing for responsiveness, we are able to accurately estimate how much of the capacity the responsiveness probes will consume and never send more probes than the network can handle.

Limiting the data used by probes can be done by providing an estimate of the number of bytes exchanged for a responsiveness probe. Taking TCP and TLS overheads into account, we can estimate the amount of data exchanged for a probe on a foreign connection to be around 5000 bytes. On load-generating connections we can expect an overhead of no more than 1000 bytes.

Given this information, we recommend that at each responsiveness probing interval does not send more than MPS (Maximum responsiveness Probes per Second - default to 100) probes per second. The probes should be spread out equally over the duration of the interval with an equal split between foreign and different load-generating connections. For the probes on load-generating connections, the connection should be selected randomly for each probe.

This would result in a total amount of data per second of 300 KB or 2400Kb, meaning a total capacity utilization of 2400 Kbps for the probing.

On high-speed networks, this will provide a significant amount of samples, while at the same time minimizing the probing overhead. However, on severely capacity-constrained networks the probing traffic could consume a significant portion of the available capacity. The Responsiveness Test must adjust its probing frequency and in such a way that the probing traffic does not consume more than PTC (Percentage of Total Capacity - default to 5%) of the available capacity.

4.3.1. Aggregating the Measurements

The algorithm produces sets of 4 times for each probe, namely: tcp_f, tls_f, http_f, http_l (from the previous section). The responsiveness evolves over time as buffers gradually reach saturation. Once the buffers are saturated responsiveness is stable over time. Thus, the aggregation of the measurements considers the last MAD (Moving Average Distance - default to 4) intervals worth of completed responsiveness probes.

Over the timeframe of these intervals a potentially large number of samples has been collected. These may be affected by noise in the measurements, and outliers. Thus, to aggregate these we suggest to use a trimmed mean at the TMP (Trimmed Mean Percentage - default to 95%) percentile, thus providing the following numbers: TM(tcp_f), TM(tls_f), TM(http_f), TM(http_l).

The responsiveness is then calculated as the weighted mean:

```
Responsiveness = 60000 /
(1/6*(TM(tcp_f) + TM(tls_f) + TM(http_f)) +
  1/2*TM(http_s))
```

This responsiveness value presents round-trips per minute (RPM).

4.4. Final Algorithm

Considering the previous two sections, where we explain what the meaning of working conditions is and the definition of responsiveness, we can design the final algorithm. In order to measure the worst-case latency we need to transmit traffic at the full capacity of the path as well as ensure the buffers are filled to the maximum. We can achieve this by continuously adding HTTP sessions to the pool of connections in a ID (Interval duration - default to 1 second) interval. This will ensure that we quickly reach capacity and full buffer occupancy. First, the algorithm reaches stability for the goodput. Once goodput stability has been achieved, responsiveness probes are being transmitted until responsiveness stability is reached.

We consider both, goodput and responsiveness to be stable, when the standard deviation of the past MAD intervals is within SDT (Standard Deviation Tolerance - default to 5%) of the last of the moving averages.

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 second.

Where

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

* moving average aggregate goodput at interval p: The number of total bytes of data transferred within interval p and the three immediately preceding intervals, divided by four times the interval duration.

the steps of the algorithm are:

* Create a load-generating connection.

* At each interval:

  - Create an additional load-generating connection.

  - If goodput has not saturated:

- Compute the moving average aggregate goodput at interval i as current_average.
- If the standard deviation of the past MAD average goodput values is less than SDT of the current_average, declare saturation and move on to probe responsiveness.

- If goodput has saturated:
  - Compute the responsiveness at interval i as current_responsiveness.
  - If the standard deviation of the past MAD responsiveness values is less than SDT of the current_responsiveness, declare saturation and report current_responsiveness.

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 stability is not reached within that time-frame. For example, an implementation might gather a provisional responsiveness measurement or let the test run for longer.

Finally, if at any point one of these connections terminates with an error, the test should be aborted.

4.4.1. Confidence of test-results

As described above, a tool running the algorithm typically defines a time-limit for the execution of each of the stages. For example, if the tool allocates a total run-time of 40 seconds, and it executes a full downlink followed by a uplink test, it may allocate 10 seconds to each of the saturation-stages (downlink capacity saturation, downlink responsiveness saturation, uplink capacity saturation, uplink responsiveness saturation).

As the different stages may or may not reach stability, we can define a "confidence score" for the different metrics (capacity and responsiveness) the methodology was able to measure.

We define "Low" confidence in the result if the algorithm was not even able to execute 4 iterations of the specific stage. Meaning, the moving average is not taking the full window into account.

We define "Medium" confidence if the algorithm was able to execute at least 4 iterations, but did not reach stability based on standard deviation tolerance.

We define "High" confidence if the algorithm was able to fully reach stability based on the define standard deviation tolerance.

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 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 a responsiveness 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. Responsiveness Test Server API

The responsiveness measurement is built upon a foundation of standard protocols: IP, TCP, TLS, HTTP/2. On top of this foundation, a minimal amount of new "protocol" is defined, merely specifying the URLs that used for GET and PUT in the process of executing the test.

Both the client and the server MUST support HTTP/2 over TLS. 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. The server MUST use a packet scheduling algorithm that minimizes internal queueing to avoid affecting the client's measurement.

As clients and servers become deployed that use L4S congestion control (e.g., TCP Prague with ECT(1) packet marking), for their normal traffic when it is available, and fall back to traditional loss-based congestion controls (e.g., Reno or CUBIC) otherwise, the same strategy SHOULD be used for Responsiveness Test traffic. This is RECOMMENDED so that the synthetic traffic generated by the Responsiveness Test mimics real-world traffic for that server.

Delay-based congestion-control algorithms (e.g., Vegas, FAST, BBR) SHOULD NOT be used for Responsiveness Test traffic because they take much longer to discover the depth of the bottleneck buffers. Delay-based congestion-control algorithms seek to mitigate the effects of bufferbloat, by detecting and responding to early signs of increasing round-trip delay, and reducing the amount of data they have in flight before the bottleneck buffer fills up and overflows. In a world where bufferbloat is common, this is a pragmatic mitigation to allow software to work better in that environment. However, that approach does not fix the underlying problem of bufferbloat; it merely avoids it in some cases, and allows the problem in the network to persist. For a diagnostic tool made to identify symptoms of bufferbloat in the network so that they can be fixed, using a transport protocol explicitly designed to mask those symptoms would be a poor choice, and would require the test to run for much longer to deliver the same results.

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 message content is irrelevant. The server SHOULD specify the content-type as application/octet-stream. The server SHOULD minimize the size, in bytes, of the response fields that are encoded and sent on the wire.

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 message 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. The actual POST message content is irrelevant. The client will probably never completely upload the object, but will instead close the connection after reaching working condition and making its measurements.

4. A .well-known URL [RFC8615] which contains configuration information for the client to run the test (See Section 7, below.)

The client begins the responsiveness measurement by querying for the JSON [RFC8259] 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. Responsiveness Test Server Discovery

It makes sense for a service provider (either an application service provider like a video conferencing service or a network access provider like an ISP) to host Responsiveness Test Server instances on their network so customers can determine what to expect about the quality of their connection to the service offered by that provider. However, when a user performs a Responsiveness Test and determines that they are suffering from poor responsiveness during the connection to that service, the logical next questions might be,

1. "What’s causing my poor performance?"

2. "Is it poor buffer management by my ISP?"

3. "Is it poor buffer management in my home Wi-Fi Access point?"

4. "Something to do with the service provider?"

5. "Something else entirely?"

To help an end user answer these questions, it will be useful for test clients to be able to easily discover Responsiveness Test Server instances running in various places in the network (e.g., their home router, their Wi-Fi access point, their ISP's head-end equipment, etc).

Consider this example scenario: A user has a cable modem service offering 100 Mb/s download speed, connected via gigabit Ethernet to one or more Wi-Fi access points in their home, which then offer service to Wi-Fi client devices at different rates depending on distance, interference from other traffic, etc. By having the cable modem itself host a Responsiveness Test Server instance, the user can then run a test between the cable modem and their computer or smartphone, to help isolate whether bufferbloat they are experiencing is occurring in equipment inside the home (like their Wi-Fi access points) or somewhere outside the home.

7.1. Well-Known Uniform Resource Identifier (URI) For Test Server Discovery

Any organization that wishes to host their own instance of a Responsiveness Test Server can advertise that capability by hosting at the network quality well-known URI a resource whose content type is application/json and contains a valid JSON object meeting the following criteria:

```
{
  "version": 1,
  "urls": {
    "large_download_url":"https://nq.example.com/api/v1/large",
    "small_download_url":"https://nq.example.com/api/v1/small",
    "upload_url":        "https://nq.example.com/api/v1/upload"
  }
  "test_endpoint": "hostname123.provider.com"
}
```

The server SHALL specify the content-type of the resource at the well-known URI as application/json.

The content of the "version" field SHALL be "1". Integer values greater than "1" are reserved for future versions of this protocol. The content of the "large_download_url", "small_download_url", and

"upload_url" SHALL all be validly formatted "http" or "https" URLs. See above for the semantics of the fields. All of the fields in the sample configuration are required except "test_endpoint". If the test server provider can pin all of the requests for a test run to a specific host in the service (for a particular run), they can specify that host name in the "test_endpoint" field.

For purposes of registration of the well-known URI [RFC8615], the application name is "nq". The media type of the resource at the well-known URI is "application/json" and the format of the resource is as specified above. The URI scheme is "https". No additional path components, query strings or fragments are valid for this well-known URI.

7.2. DNS-Based Service Discovery for Test Server Discovery

To further aid the test client in discovering instances of the Responsiveness Test Server, organizations wishing to host their own instances of the Test Server MAY advertise their availability using DNS-Based Service Discovery [RFC6763] using conventional, unicast DNS [RFC1034] or multicast DNS [RFC6762] on the organization network's local link(s).

The Responsiveness Test Service instances should advertise using the service type [RFC6335] "_nq._tcp". Population of the appropriate DNS zone with the relevant unicast discovery records can be performed automatically using a Discovery Proxy [RFC8766], or in some scenarios simply by having a human administrator manually enter the required records.

7.2.1. Example

An obscure service provider hosting a Responsiveness Test Server instance for their organization (obs.cr) on the "rpm.obs.cr" host would return the following answers to PTR and SRV conventional DNS queries:

```
$ nslookup -q=ptr _nq._tcp.obs.cr.
Non-authoritative answer:
_nq._tcp.obs.crname = rpm._nq._tcp.obs.cr.
$ nslookup -q=srv rpm._nq._tcp.obs.cr.
Non-authoritative answer:
rpm._nq._tcp.obs.crservice = 0 0 443 rpm.obs.cr.
```

Given those conventional DNS query responses, the client would proceed to access the rpm.obs.cr host on port 443 at the .well-known/nq well-known URI to begin the test.

8. Security Considerations

TBD

9. IANA Considerations

IANA has been requested to record the service type "_nq._tcp" (Network Quality) for advertising and discovery of Responsiveness Test Server instances.

10. Acknowledgments

Special thanks go to Will Hawkins and Jeroen Schickendantz for their tireless enthousiasm around the project and their contributions to this I-D and the development of the Go responsiveness measurement tool. We would also like to thank Rich Brown for his editorial pass over this I-D. We also thank Erik Auerswald for his constructive feedback on the I-D.

11. 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", RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[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>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC8766] Cheshire, S., "Discovery Proxy for Multicast DNS-Based Service Discovery", RFC 8766, DOI 10.17487/RFC8766, June 2020, <https://www.rfc-editor.org/info/rfc8766>.

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 remap configuration file then is:

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

   map https://nq.example.com/api/v1/large \
       http://localhost/cache/8589934592/ \
       @plugin=generator.so

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

   map https://nq.example.com/api/v1/upload \
       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

Will Hawkins
University of Cincinnati
Email: hawkinwh@ucmail.uc.edu

Matt Mathis
Google, Inc
1600 Amphitheatre Parkway
Mountain View, CA 94043,
United States of America
Email: mattmathis@google.com