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 its
   solutions. We believe that creating a tool whose measurement matches
   people's everyday 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
   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.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

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.

   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 would help establish a new expectation
   that the standard measures of network quality should -- in addition
   to throughput and idle latency -- also include latency under load,
   or, as we prefer to call it, responsiveness under working conditions.

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 car engines.

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 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 communicating
   devices, 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 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 to make it no longer the lowest-capacity hop,
   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 other 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 RPM Test mimics normal network operations and data transfers, to cause this bottleneck buffer to fill to capacity, and then measures the resulting end-to-end latency under these operating conditions. A well managed bottleneck queue keeps its queue occupancy under control, resulting in consistently low round-trip time and consistently good responsiveness. A poorly managed bottleneck queue will not.

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.

   The working conditions we try to achieve is a scenario where the path
   between the measuring endpoints is utilized at its full end-to-end
   capacity. An ideal sender could send at just this link-speed without
   building a queue on the bottleneck. Thus, in order to measure the
   worst-case responsiveness we need to ensure that a queue is building
   up on the bottleneck, meaning that responsiveness is at its worst.

   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 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. From single-flow to 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 RPM
test is to quickly load the network, take its measurements, and then
finish. 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 RPM Test is not to productively move data across
the network in a useful way, the way a normal application does. The
purpose of the RPM 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, and then end the test.
Because of this, using multiple simultaneous parallel connections
allows the RPM 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.

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 buffer 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) as well as responsiveness
   continuously. When both goodput and responsiveness stop changing, it
   means that the test managed to fill the buffer of the bottleneck. 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. Measuring Responsiveness

   Measuring responsiveness during the previously explained working
   conditions creation is a continuous process during the duration of
   the test. It requires a sufficiently large sample-size to have
   confidence in the results.

   The measurement of the responsiveness happens by sending probe-
   requests for a small object. 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_foreign).
Second, the TLS round-trip-time (noted tls_foreign). 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_foreign).

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_self).

tcp_foreign, tls_foreign, http_foreign and http_self are all measured
in milliseconds.

The more probes that are sent, the more data is available for
calculation. In order to generate as much data as possible, the
methodology requires a client to issue these probes every 100ms. For
the probes on the load-generating connections, the client needs to
use one of the initial load-generating connections. This means that
every 100ms, 2 probes are being evaluated. The total amount of data
used for these probes would be no more than about 50KB worth of data
within one second.

4.2.1. Aggregating the Measurements

The algorithm produces sets of 4 times for each probe, namely:
tcp_foreign, tls_foreign, http_foreign, http_self (from the previous
section). Each of these sets will have a large number of samples.
Use the following methodology to calculate a single RPM value from
these data:

1. Among each set, take the 90th percentile, thus resulting in 4
    individual numbers (tcp_foreign_p90, tls_foreign_p90,
    http_foreign_p90, http_self_p90).

2. Calculate the RPM as the weighted mean:

Responsiveness = 60000 /
((1/3*tcp_foreign_p90 + 1/3*tls_foreign_p90 + 1/3*http_foreign_p90 +
  http_self_p90)/2)

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

4.3. 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 1-second interval. This
   will ensure that we quickly reach capacity and full buffer occupancy.
   We need to continuously measure goodput and responsiveness and as
   soon as we detect stability for both metrics we can ensure that the
   full working conditions have been reached.

   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.

   * instantaneous aggregate goodput at interval p: The number of total
      bytes of data transferred within interval p, divided by the
      interval duration. If p is negative (i.e., a time interval
      logically prior to the start of the test beginning, used in moving
      average calculations), the number of total bytes of data
      transferred within that interval is considered to be 0.

   * 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.

   * Start probing for responsiveness every 100ms, as described in the
      previous section.

   * At each interval:

      - Create an additional load-generating connection.

      - Compute the instantaneous aggregate goodput at interval i.

      - Compute the moving average aggregate goodput at interval i.

      - Compute the responsiveness

      - 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. Continue for 4 more iterations.

      - If the responsiveness at interval i is more than a 5% reduction
         over the responsiveness at interval i - 1, the network has not
         yet reached full buffer occupancy. Continue for 4 more
         iterations.

   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.

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 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 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 RPM test traffic. This is
RECOMMENDED so that the synthetic traffic generated by the RPM test
mimics real-world traffic for that server.

Delay-based congestion-control algorithms (e.g., Vegas, FAST, BBR)
SHOULD NOT be used for RPM 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 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. RPM 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 RPM Test Server instances on their
   network so customers can determine what to expect about the the
   quality of their connection to the service offered by that provider.
   However, when a user performs an RPM test and determines that they
   are suffering from poor RPM 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 RPM 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 an RPM 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 an RPM
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_https_download_url":"https://nq.example.com/api/v1/large",
    "small_https_download_url":"https://nq.example.com/api/v1/small",
    "https_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_https_download_url",
"small_https_download_url", and "https_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 RPM
   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 RPM 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 an RPM 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 RPM Test Server
   instances.

10. Acknowledgments

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>.

   [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

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