IP Performance Measurement
Internet-Draft
Intended status: Standards Track
Expires: 8 January 2026

C. Paasch
R. Meyer
S. Cheshire
Apple Inc.
W. Hawkins
University of Cincinnati

7 July 2025

Responsiveness under Working Conditions
draft-ietf-ippm-responsiveness-07

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 have solved the problem.

Our network connections continue to suffer from an unacceptable amount of delay, not for 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 goodput (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.

This Internet-Draft will expire on 8 January 2026.

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], PIE [RFC8033], Cake [Cake] and L4S [RFC9330] have been standardized and implemented, and are, to some extent, deployed. Nevertheless, poor network responsiveness, resulting from bufferbloat and other causes, continues to affect many services and the people who use them.

Whenever a network path is actively being used at its full capacity, if the bottleneck link has poor buffer management then it may allow an overly large queue to build up, resulting in high delay for the traffic in that queue. Although bufferbloat significantly degrades user experience, the impact can be transitory. On a network that is generally underutilized except for occasional medium-sized data transfers, like sending an email with an attachment, or uploading a

photo, the user-experience disruptions can be intermittent and brief, making them hard to diagnose. The user may notice that the network seems sluggish for a few seconds, or a videoconferencing application may briefly show a message saying that the connection is “unstable”, but before the user has a chance to run any diagnostic tools to investigate, the problem has disappeared. People have come to accept that the Internet will have “glitches” from time to time, and it has almost become considered normal. Upgrading to an Internet connection with higher capacity does not eliminate these disruptions, but it can shorten their duration. Ironically, this has the effect of making the problem even harder to diagnose, so instead of fixing the true problem, this “upgrade” creates a situation where the actual underlying problem is even more likely to remain unsolved. Instead of eliminating Internet glitches, it just makes them a little more elusive and harder to investigate.

To help engineers work on eliminating these glitches it is useful to have a tool that (a) reliably recreates the conditions that cause systems with poor buffer management to exhibit latency spikes, and (b) quantifies the magnitude of the resulting latency spikes. This helps engineers identify where these problems exist. After design changes are made, it helps engineers quantify how much their changes have improved the situation.

Note that this document does not advocate for entirely eliminating queues from networking, or even for mandating shallow queues of some arbitrary fixed size. Packet-switched network traffic tends to be bursty, to varying degrees depending on the technology, and queuing performs the essential function of smoothing out those bursts to avoid packet loss. What this document recommends is that (a) network queues should be large enough to perform their essential smoothing function, without being so large that they add additional unnecessary delay, and (b) that when a persistent standing queue begins to form (therefore exceeding the amount of queueing needed for smoothing) network queues should communicate feedback back to the sender, telling to to reduce its sending rate to allow the standing queue to drain.

Including the responsiveness-under-working-conditions test among other measurements of network quality (e.g., goodput and idle latency) helps customers make informed choices about the Internet service they buy.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses terminology and concepts from HTTP [RFC9110], such as request and response header fields and content.

The term "bufferbloat" describes the situation where a network device is prone to buffering an excessive number of packets in transit through that device, causing a backlog of packets waiting to go out, resulting in those waiting packets experiencing excessive unnecessary delay [Bufferbloat].

1.2. Round-Trips per Minute (RPM)

Although engineers are comfortable thinking about and discussing latency in terms of milliseconds, this is a poor way to report responsiveness to end users, because it is not intuitive to the general public. The units are unfamiliar to many ("what is a millisecond?") and even when the units are understood in principle, they can still be misleading to many ("100 ms -- that sounds good -- it's only a tenth of a second!").

In addition, most current "latency" tests report the round-trip time when the network is otherwise idle, which not a good predictor of how a network will behave when it is actively being used for normal data transfer.

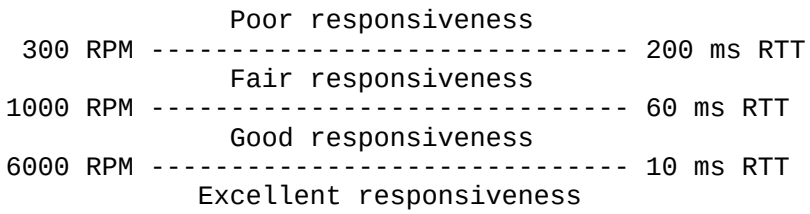
Instead, this document defines an algorithm for the "Responsiveness Test" that explicitly measures responsiveness under working conditions, not only in idle conditions.

The results are presented as "round-trips per minute" (RPM), which is calculated by dividing 60 by the round-trip time in seconds (or 60,000 divided by the round-trip time in milliseconds).

The advantages of expressing working latency in round-trips per minute (RPM) are two-fold: First, it allows for a unit that where "higher is better". This kind of unit is often more familiar to end-users. Second, the range of the values tends to be around the four-digit integer range, which is 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.

Some readers have complained that they don’t understand what "RPM" means, and that it is just an arbitrary number with no way to judge what should be considered a bad RPM rating or a good RPM rating. We counter that argument by saying that round-trip times expressed in milliseconds are equally arbitrary, and people similarly judge whether a certain number of milliseconds is “good” or “bad” from their personal experience and in relation to the range of round trip times that are observed across a range of technologies. In the same way, the RPM of a car engine is also interpreted relative to the range of values that people have learned to expect. We have learned that 8000 is a high operating RPM for a car engine, and anything below 1000 RPM is low. Interestingly, the range of values observed for network responsiveness happen to fall into this familiar range.

This document classifies responsiveness scores into four broad categories:



The reasoning for these classifications is two-fold. Partly they are rooted in the unchangeable laws of physics, and partly they are determined by human factors.

We know that we can never beat the speed-of-light propagation delay. To pick one representative example, the coast-to-coast distance across the USA (Stanford to MIT) is about 4,500 km. That makes the round-trip distance 9,000 km. The speed of light is 300,000 km/s, so the Stanford-MIT round-trip time has to be at least 30 ms. But light in fiber-optic cables travels at just 200,000 km/s (the propagation rate for electrical signals in copper wire is about the same) making the round-trip time at least 45 ms [L96] for these kinds of fiber-optic or electrical Internet circuits. Of course, software overhead, queuing effects, and other delays will add to that round-trip time. If our actual measured working round-trip time is at most 2x the 45 ms theoretical minimum, then we feel the network is operating well. If our actual measured round-trip time is 10x the 45 ms minimum or worse, then we feel we have considerable opportunity for improvement. That 45 ms time is for communications coast-to-coast across a continent, and not everyone is communicating over such long distances. In fact, modern content delivery networks (CDNs) have data centers in many locations around the world, partly to keep data closer to the user and reduce round-trip times. For communication

over a shorter distance, within a single state or within a city, we should be striving to achieve consistent reliable round-trip times below 10 ms. This is why, in the context of the public Internet, we consider that only working-latency values below 10 ms (6000 RPM or better) deserve to be classified as “excellent”.

Human factors influence latency requirements, and by lucky coincidence the speed of light and the size of our planet are such that physics allows us to meet those requirements on planet Earth.

For example, decades of research in the telephone community has concluded that when mouth-to-ear round-trip times exceed 250 ms, the quality of conversation begins to degrade. For example, when the person talking pauses briefly, the listener can mistakenly think that the speaker has finished and is waiting for a reply, and then inadvertently talk over their next sentence. A long round-trip delay means that the person interrupting speaks longer before realizing their mistake, and these repeated interruptions makes the conversation become stilted and awkward. Given that various other components of Internet audio (digitizing audio from a microphone, compressing that audio, quantizing it into discreet packets, decompressing the received audio, and playing the audio through a speaker) also require time, we believe it is reasonable to allow 200 ms for packet transmission over the network, and 50 ms for other elements of the process. Accordingly, we consider that working-latency values above 200 ms (300 RPM or worse) deserve to be classified as “poor”.

For another example, consider that Virtual Reality (VR) systems typically refresh their screens at 90Hz (11 ms per frame) or higher. Similarly, Apple recently increased the refresh rate of iPhone screens from 60Hz (16 ms) to 120Hz (8 ms), because human users can tell the difference, and appreciate the improved responsiveness provided by a lower screen-refresh delay.

Consequently, while we consider the minumum for an acceptable telephone call to be around 300 RPM (200 ms), we acknowledge that better responsiveness (1000 RPM or 60 ms RTT) can deliver a telephone call that is considered “good” rather than just “fair”.

Looking to the future, we feel that the networking community should be able to achieve consistent working round-trip delays on par with current screen refresh times, which is why we set 6000 RPM (10 ms) as our threshhold for a network to be considered “excellent.”

At this end of the performance spectrum, we note that the speed of sound (in air) is roughly 350 m/s (a million times slower than the speed of light). This is 35 cm per millisecond, or roughly a foot

per millisecond. This gives us a parameter to evaluate what constitues a “reasonable” goal for network latency. While lower latency may always be better for certain computer interactions, pursuing ever-lower network latencies, down to one millisecond or lower, may not be necessary for human audio. A one millisecond difference in network delay is equivalent to a change of listener position, relative to the loudspeaker playing the audio, of just one foot.

The BITAG “Latency Explained” report [BITAG] has excellent detailed information on the causes of network latency, and the human factors that influence latency requirements.

2. Overview

In the last few decades, the networking industry has made enormous advances in terms of the quantity of data our wired and wireless links can transfer. Rates have gone from kilobits per second, to megabits, to gigabits, and continue to increase. We also have broad industry agreement on the units for measuring network capacity -- bits per second. We have a wide array of tools available for measuring capacity in these units, and these tools generally produce results that are consistent and comparable between different tools.

In contrast to our remarkable improvements in throughput, we have not done a good job developing networks that deliver consistently low delay in all normal operating conditions. In fact, many people may have unconsciously assumed that it would not be possible to achieve such a capability. Correspondingly, we have not had industry agreement on what constitutes reasonable fair working conditions under which to measure a network’s round-trip delay, and, consequently, for a long time we have measured and reported the round-trip time on an idle network as the only metric. The actual round-trip times observed when traffic is flowing have generally been so much worse than the idle round-trip time (often by a factor of 100 or more) that it seemed almost impolite to draw attention to them.

And so, by measuring and reporting throughput and idle round-trip time, we have motivated vendors and operators to focus on those aspects of performance, and to neglect other factors that also impact the quality of user experience.

Measurements of idle round-trip time can be informative, but users care about how well their network performs when they are using it, not only how well it performs when no one is using it. In this document we specify how to measure network round-trip time under reasonable representative operating conditions. This document focusses on this specific aspect of network quality, because we feel

it is a particularly pressing issue at this time. Future companion documents could address how to measure and report other aspects of network quality that affect user experience.

2.1. Relevance

The most important property of this test tool is that its results should be a good predictor of user experience. High scores on the Responsiveness Test should correlate with good user experience, and low scores should correlate with poor user experience. A test with this predictive power gives network engineers a way to experiment with code changes and then use the test to evaluate whether the changes are beneficial, instead of having to judge the effects subjectively, for example, by conducting a video conference and trying to assess whether it seems to be working better or worse.

2.2. Repeatability

For the test to be useful, it has to produce consistent results. If results vary wildly from run to run, then an engineer doesn't know whether a different result from the test is a result of a software change or just an artifact of randomness in the test.

2.3. Convenience

The test should complete as quickly as possible, ideally with ten seconds, but only if this can be achieved without sacrificing relevance and repeatability.

2.4. Interaction with End Systems

Network delays that occur when traffic is flowing are not purely a property of the network. How traffic flows is a result of the interaction between the behavior of the network and the behavior of the end systems generating and receiving the traffic. Consequently, if we are to obtain useful measurements pertaining to the network itself, uncontaminated by noise or bias introduced by the test endpoints, we need to ensure that the test endpoints are of the highest quality and are not responsible for introducing significant delays (or delay variation) themselves. This means that the test endpoints should be using source buffer management techniques like TCP_NOTSENT_LOWAT [RFC9293][SBM] to keep the backlog of unsent data limited to a reasonable amount, in conjunction with the current best-in-class rate adaptation algorithms (such as the Prague congestion controller [Prague]) and the current best-in-class network signalling (such as L4S [RFC9330]). These techniques allow the network to signal when the source is sending too fast and a queue is starting to build up, so that the source can adjust its sending rate accordingly.

We believe that having the network signal when the source is sending more data than the network can carry (rather than just letting the situation worsen until a queue overflows and packets are lost) is vital for creating networks that deliver consistent low delay. If our belief is correct, we expect the test results to reflect that networks with this signalling yield lower delays than networks without it. In any case, if the sending and receiving test endpoints are not able to make use of L4S signalling, then the test will be unable to measure and report the effect of L4S support (or its absence) in the network's bottleneck links.

2.5. Purpose of Test Tool

The primary purpose of this test tool is to evaluate network behavior, and that is the main focus of this document.

However, a secondary use can be to evaluate client and server software.

If a particular test client, over a particular network path, produces good responsiveness scores when communicating with a known-good test server, but poor results when using another test server, that suggests problems with the other test server. We have already had cases where we used an existing HTTP server as a test endpoint, got worse-than-expected responsiveness scores, and as a result realized that the HTTP server had some poor configuration settings, which were also causing unnecessary slowness for the other user-facing traffic it was serving. Tuning these configuration settings for higher responsiveness lowered delays across the board for all traffic delivered from that server.

Similarly, if a new test client, connecting to a known-good test server over a particular network path, produces worse results than an existing known-good test client, then this suggests problems with the new test client, possibly in the new test client's code, or in it's operating system's networking implementation. These insights can lead to enhancements in the networking code that improve responsiveness for all software running on that operating system.

2.6. Use of HTTP

The test specified in this document is based on HTTP/2 and HTTP/3. HTTP was selected because it is representative of a broad class of commonly used request/response protocols, and protocols that do bulk data transfer, adapting their sending rate to match the available network capacity.

Many protocols share the property that over a single communication channel a client:

(a) may read small or large data objects from the server,
(b) may write small or large data objects to the server,
(c) may have multiple operations executing concurrently, and
(d) may choose to cancel outstanding operations if circumstances change and the operation is no longer needed.

If a client reads a very large data object from a server and then a small data object, it is preferable if the small data object doesn't have to wait until after the very large data object has been received in its entirety.

HTTP supports reads, writes, concurrent operations, cancellation of outstanding operations, and interleaving of data from concurrent operations. This makes HTTP a representative proxy for virtually any request/response protocol. The specific details of the message syntax, or the byte values used, are not important.

HTTP has the additional convenience that it is very widely deployed in today's world, and it is fairly easy to configure many modern web servers to operate as Responsiveness Test servers.

If a network is able to deliver consistent low latency even for the challenging case of a bulk data transfer over HTTP, then it is reasonable to assume that this network will deliver consistent low latency for all IP traffic, including interactive audio and video telephony traffic, which can be regarded as a kind of sender-limited bulk transfer.

2.7. Resisting Cheating

A desired property of the test was that it should resist cheating, both intentional and accidental.

Suppose we were to measure network round-trip delay using ICMP Echo Request packets. A network engineer given the task of improving a vendor score on this metric might choose to simply prioritize ICMP Echo Request packets so that they always go directly to the front of the transmission queue. Or, while exploring various options, the engineer might innocently make a code change that inadvertently has this effect. A quick run of the test tool would show that the engineer had achieved the assigned goal -- on busy networks the tool reports that the round trip time is now lower. Unfortunately, this change would in no way improve actual user experience, because normal data transfer is not performed using ICMP Echo Request packets.

To avoid that pitfall, the Responsiveness Test was designed to obtain its measurements using normal client operations over HTTP. These are representative of the kinds of operations a normal client might do if rapidly fetching map tiles while a user scrolls and zooms a map to view an area of interest on their smartphone, or fetching new video data when a viewer skips ahead to a different place in a streaming video. Ultimately, application-layer responsiveness is what affects user experience, not lower-layer performance metrics.

If a creative engineer does find a way to “cheat” the Responsiveness Test to get a better score, then it is almost certain that such a “cheat” would have the effect of making real-world operations faster too. This is a kind of “cheating” we are happy to tolerate.

3. 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.

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. The traditional delay measurement tools use ICMP, which may experience even more drastically different behavior than TCP or UDP. Thus, 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.

Significant queueing in the network 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 is some hop along a network path between two hosts with the lowest capacity for that path. If that hop were to be improved by increasing its capacity, 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.

Note that in a shared datagram network, conditions do not remain static. The properties of the hop that is the current bottleneck for a particular flow may change from moment to moment. The amount of other traffic sharing that bottleneck may change, or the underlying capacity of that hop itself may vary. If the available capacity on that hop increases, then a different hop may become the bottleneck for that flow. For example, changes in the simultaneous transmission of data flows by hosts communicating over the same hop may result in changes to the share of bandwidth allocated to each flow. A user who physically moves around may cause the Wi-Fi transmission rate to vary widely, fluctuating between a few Mb/s when they are far from the Access Point, and 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 in the network 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 path. 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 Responsiveness Test mimics normal network operations and data transfers, with the goal of filling the bottleneck link to capacity, and then measures the resulting end-to-end delay under these so-called working conditions. A well managed bottleneck queue keeps its buffer occupancy under control, resulting in consistently low round-trip times and consistently good responsiveness. A poorly managed bottleneck queue will not.

This section discusses bufferbloat in terms of a problem that occurs in a network, i.e., in routers and switches. However, there are some other system components that can also add delay.

In some cases the lowest capacity hop on a path is the first hop. In this case network bufferbloat is not usually a significant concern because the source device is simply unable to transmit data fast enough to build up a significant queue anywhere else in the network. However, in this case source-device bufferbloat (excessive queuing in the device' s own outgoing network interface) can result in excessive self-induced delays for the traffic it is sending [SBM].

The job of the rate adaptation (congestion control) algorithm of the sender's transport protocol is to determine this flow's share of the bottleneck hop on the path, and to restrain its own transmission rate so as not to exceed that bottleneck rate. If the transport protocol does not generate appropriate backpressure to the application (e.g., using TCP_NOTSENT_LOWAT [RFC9293][SBM]) then the transport protocol itself can cause significant delay by buffering an excessive amount of application data that has not even been sent yet.

Finally, an application can make delay even worse by maintaining its own queue of data that it hasn't even given to the transport protocol for sending yet. Any time data spends sitting in this application queue adds to the delay it experiences while waiting to be set out of the network interface and the delay it experiences while in transit traversing the network.

4. Goals

The algorithm described here defines the Responsiveness Test that serves as a means of quantifying user experience of latency in a network connection. 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) [RFC9114] and/or are already being used widely (RTP) [RFC1889]. 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 [RFC9293], 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 (it is left to the implementation to chose a suitable time-limit and we recommend for any implementation to allow the user to configure the duration of the test).

5. 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.

5.1. 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.

Current capacity-seeking transport protocols, like TCP and QUIC, seek to achieve the highest throughput that the network can carry, by increasing their sending rate until the network signals that the transport protocol has reached the optimal throughput and should not increase further. That signal from the network can take the form of packet loss (when a bottleneck queue overflows) or ECN signals (prior to queue overflow).

The Responsiveness Test defines working conditions as the condition where the path between the measuring endpoints is fully utilized at its end-to-end capacity, and senders are sending a little faster to discover if more capacity is available, causing a queue to build up at the ingress to the bottleneck hop. How the device at the ingress to the bottleneck hop manages and limits the growth of that queue will influence the network connection’s responsiveness.

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 creates a stable state of working conditions during which the bottleneck of the path between client and server is fully utilized at its capacity, revealing the behavior of its buffer management or Active Queue Management (AQM), 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.

5.1.1. Single-Flow vs Multi-Flow

The purpose of the Responsiveness Test is not to productively move data across the network, the way a normal application does. The purpose of the Responsiveness Test is to, as quickly as possible, simulate a representative traffic load as if real applications were doing sustained data transfers and measure the resulting round-trip time occurring under those realistic conditions. 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. For higher capacity connections or higher round-trip times, a 4MB receive window is insufficient. In addition, deep buffers along the path between the two endpoints may be significantly larger than 4MB, and using a 4MB TCP receive window would be insufficient to fully expose the depth of these buffers.

TCP allows larger receive window sizes, up to 1GB. However, to avoid consuming too much memory, most transport stacks today do not use such large receive windows.

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 measure their network quickly. As a result, the test should 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.

For all these reasons, 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.

One of the configuration parameters for the test is an upper bound on the number of parallel load-generating connections. We recommend a default value for this parameter of 16.

5.1.2. Concurrent vs Sequential Uplink and Downlink

Poor responsiveness can be caused by bloated 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.

Most of today's widely used network benchmark tools measure the throughput in one direction at a time. In a real sense this is very much a "softball" test, contrived to let the network perform at its absolute best so it can produce the most impressive-looking benchmark results.

In the 1950s we had analog telephone lines, with the property that only one person at a time could make a telephone call. In the 1990s we had dial-up services like AOL, with the property that only one person at a time could use the telephone line to connect to AOL. Two people in a home could not share a single telephone line and both connect to AOL at the same time.

But, in contrast, by the 1990s we had packet switching, as embodied in the Internet, Ethernet, Wi-Fi, and similar connectionless datagram technologies. The huge benefit of these connectionless datagram technologies is that an arbitrary number of people and devices can share a network and use it for different purposes at the same time, in stark contrast to a telephone line that can only support one telephone call at a time. Indeed, it is common today for a home network to have dozens of devices sharing a user's network -- home security cameras streaming video, adults working from home via videoconferencing, children watching streaming video or playing video games, etc.

Given that today's home networks are frequently used by multiple people and devices at the same time, (and given that this is arguably the whole reason connectionless datagram networking was invented in the first place) it makes sense for a network benchmark tool to evaluate how well the network performs when it is being used this way, rather than using only a carefully contrived artificial scenario, intentionally doing only one thing at a time so as to show the network in the best possible light.

For this reason, the Apple "networkQuality" tool currently performs the upload and download tests simultaneously, to properly reflect how well a network performs when it is used this way.

However, a number of caveats come with measuring concurrently:

* 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. However, this circumstance might also yield some useful benefit. If a user's Internet connection performs reasonably well (in terms of both capacity and responsiveness) when carrying predominantly uplink traffic, and performs reasonably well when carrying predominantly downlink traffic, but degrades badly when the user tries to do both upload and download at the same time, that is information that might be very useful to the user.

* Debugging the results becomes harder: During concurrent measurement it is impossible to differentiate whether the observed delay happens in the uplink or the downlink direction.

For this reason, a test tool should also offer the option of performing the upload and download tests sequentially, to help engineers diagnose whether the source of excessive delay is in the upstream direction, downstream direction, or both.

When performing a concurrent test, a single overall responsiveness score is reported, to reflect the overall experience a user can expect during normal unrestricted network usage.

When performing uplink and downlink tests sequentially the two responsiveness scores are reported individually, to give the user visibility into whether their network is performing better in one direction than the other.

5.1.3. Achieving Steady-State 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 -- if the transport protocol emits more traffic into the network than is needed, the additional traffic will either get dropped, or be buffered and thus create a "standing queue" that is characteristic of bufferbloat. At this moment the test starts measuring the responsiveness until that metric reaches saturation. At this point we are creating the worst-case scenario (best-case for throughput, worst-case for latency) within the limits of the realistic traffic pattern. Well designed network equipment handles this scenario without creating excessive delay.

The algorithm presumes that goodput increases rapidly until TCP connections complete their TCP slow-start phase. At that point, goodput 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 goodput is by adding more TCP connections to the pool of load-generating connections. If new connections don't result in an increase in goodput, full link utilization has been reached. At this point, adding more connections will reveal the extent to which the network is willing to buffer excess packets, with a resulting increase in round-trip delay (decrease in responsiveness). When the bottleneck queue signals the sender(s) to slow down (either via packet drop or via ECN marking) then the round-trip delay will stabilize.

5.1.4. Avoiding Test Hardware Bottlenecks

The Responsiveness Test could be run from various devices that are either consumer devices or Internet infrastructure such as routers. Many home routers are cost-sensitive embedded devices optimised for switching packets rather than terminating TLS connections at line rate. As a result, they may not have sufficient processing power or memory bandwidth to saturate a bottleneck link in order to be a useful test client for the responsiveness test.

In order to measure responsiveness from these devices, the test can be conducted without TLS over plain HTTP. Whenever possible, it is preferred to test using TLS to resemble typical Internet traffic to the maximum extent.

5.2. Test Parameters

A number of parameters can be used to configure the test methodology. The following list contains the names of those parameters and their default values. The detailed description of the methodology that follows will explain how these parameters are being used. Experience has shown that the default values for these parameters allow for a low runtime for the test and produce accurate results in 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
ID	Interval duration at which the algorithm reevaluates stability	1 second
TMP	Trimmed Mean Percentage to be trimmed	95%
SDT	Standard Deviation Tolerance for stability detection	5%
INP	Initial number of concurrent transport connections at the start of the test	1
INC	Number of transport connections to add to the pool of load-generating connections at each interval	1
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

5.3. Measuring Responsiveness

Measuring responsiveness under working conditions is an iterative process. Moreover, it requires a sufficiently large sample of measurements to have confidence in the results.

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

1. An HTTP GET request on a connection separate from the load-generating connections ("foreign probes"). This probe type 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. An HTTP GET request multiplexed on the load-generating connections ("self probes"). This probe type 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 mapping application 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 three (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. In the case that TLS is not being used, it is undefined. 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 that measures the duration of time between when the HTTP GET request for the 1-byte object is issued on the load-generating connection and when the full HTTP response has been received (noted http_l). For cases where multiplexing GET requests into the load generation connections is not possible (e.g. due to only HTTP/1.1 being available), the TCP stack estimated round-trip-time can be used as a proxy or substitute value.

Note that self-probe requests MUST NOT use HTTP priorities or similar techniques in pursuit of an artificially inflated RPM score. The purpose of self-probe requests is to imitate the activity of a real application requesting new data elements (like new video segments, or new map tiles) while an existing data transfer is already proceeding, and to evaluate how rapidly these new data elements are delivered. For example, in a real map application, using a higher HTTP priority every time a user scrolls the map would lead to an escalating ladder of ever-higher priorities where every new operation is more important than everything that went before it. Priorities are a solution to the problem of having too much data already in flight, buffered in the network at a bottleneck queue, or prematurely committed to the sending machine's transport protocol (e.g., TCP or QUIC) [SBM], by creating a mechanism for new data to bypass the backlog of stale data. We believe that the best way to avoid delays caused by having an excessive backlog of stale data is the simplest: avoid having a backlog of stale data in the first place, rather than assuming that a large backlog of stale data is inevitable, and then using complicated mechanisms like priorities to work around that problem. When the backlog of stale data is minimized, all traffic achieves low delay, not just a privileged minority of "priority" traffic.

tcp_f, tls_f, http_f and http_l 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 starting to probe for responsiveness, the test will have an accurate estimate of 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 each of the probe types. Taking TCP and TLS overheads into account, we can estimate the amount of data exchanged for a foreign probe to be around 5000 bytes. For self probes we can expect an overhead of no more than 1000 bytes.

Given this information, we recommend that a test client does not send more than MPS (Maximum responsiveness Probes per Second - default to 100) probes per ID. The same amount of probes should be sent on load-generating as well as on separate connections. The probes should be spread out equally over the duration of the interval, with the two types of probes interleaving with each other. The test client should uniformly and randomly select from the active load-generating connections on which to send self probes.

According to the default parameter values, the probes will consume 300 KB (or 2400Kb) of data per second, meaning a total capacity utilization of 2400 Kbps for the probing.

On high-speed networks, these default parameter values 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 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.

5.3.1. Aggregating the Measurements

5.3.1.1. For the TLS-Enabled Case

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 of the network connection being tested evolves over time as buffers gradually reach saturation. Once the buffers are saturated, responsiveness will stabilize. Thus, the final calculation of network responsiveness considers the last MAD (Moving Average Distance - default to 4) intervals worth of completed responsiveness probes.

Over that period of time, a large number of samples will have been collected. These may be affected by noise in the measurements, and outliers. Thus, to aggregate these we suggest using a single-sided 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 average of the "foreign responsiveness" on separate connections and the responsiveness on load-generating connections.

```
Foreign_Responsiveness = 60000 / ((TM(tcp_f)+TM(tls_f)+TM(http_f))/3)
Loaded_Responsiveness = 60000 / TM(http_l)
Responsiveness = (Foreign_Responsiveness + Loaded_Responsiveness) / 2
```

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

5.3.1.2. For the TCP-Only Case

If there are no TLS connections being used, then the notion of a normalised round-trip time for the TLS handshake does not apply. Zeroes cannot be substituted for tls_f, as that will result in an artificially low responsiveness value. Instead, the same principle of giving each contribution to the foreign RTT equal weight, and then giving the foreign and self RTTs equal weights is applied.

The TCP-only responsiveness is therefore calculated in the following way:

```
Foreign_Responsiveness = 60000 / ((TM(tcp_f) + TM(http_f)) / 2)
Loaded_Responsiveness = 60000 / TM(http_l)
Responsiveness = (Foreign_Responsiveness + Loaded_Responsiveness) / 2
```

5.4. Final Algorithm

Considering the previous two sections, where we explained the meaning of working conditions and the definition of responsiveness, we can now describe the design of the final algorithm. In order to measure the worst-case delay, 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 an ID (Interval duration - default to 1 second) interval. This technique ensures that we quickly reach full capacity. In parallel with this process we send responsiveness probes. First, the algorithm reaches stability for the goodput. Once goodput stability has been achieved, the responsiveness stability is tracked until it is shown to be stable at which point the final measurement can be computed.

We consider both goodput and responsiveness to be stable when the standard deviation of the moving averages of the responsiveness calculated in the immediately preceding MAD intervals is within SDT (Standard Deviation Tolerance - default to 5%) of the moving average calculated in the most-recent ID.

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 MAD - 1 immediately preceding intervals, divided by MAD times ID.

the steps of the algorithm are:

* Create INP load-generating connections (INP defaults to 1, but can be increased if one has prior knowledge on the capacity of the link).

* Launch a new foreign and self probe (according to the specification set forth above) every 1/MPS seconds until MPS*ID pairs of probes have been launched or the end of the interval is reached, whichever comes first. probes to not exceed the percentage of total capacity (PTC).

* At each interval:

  - Create INC additional load-generating connections (INC defaults to 1, but can be increased for a more aggressive ramp-up phase).

  - If goodput has not saturated:

    o Compute the moving average aggregate goodput at interval i as current_average.

    o If the standard deviation of the past MAD average goodput values is less than SDT of the current_average, declare goodput saturation and move on to probe responsiveness.

  - If goodput saturation has been declared:

    o 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 responsiveness saturation and report current_responsiveness as the final test result.

In Section 4, it is mentioned that implementations may chose to implement a time-limit on the duration of the test. If stability is not reached within the time-frame, the implementation can report the current results with a indication of confidence in the result as described in the following section.

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

5.4.1. 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 MAD iterations of the specific stage. Meaning, the moving average is not taking the full window into account. This can happen if the time-limit of the test has been reached before MAD intervals could execute.

We define "Medium" confidence if the algorithm was able to execute at least MAD 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 defined standard deviation tolerance.

It must be noted that depending on the chosen standard deviation tolerance or other parameters of the methodology and the network-environment it may be that a measurement never converges to a stable point. This is expected and part of the dynamic nature of networking and the accompanying measurement inaccuracies, which is why the importance of imposing a time-limit is so crucial, together with an accurate depiction of the "confidence" the methodology was able to generate. The confidence score should be reported to the user as part of the main results.

6. Interpreting Responsiveness Results

The described methodology uses a high-level approach to measure responsiveness. By executing the test with normal 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 is fully intentional as the properties of the client and the server implementations have a direct impact on the perceived responsiveness by the user. This section describes how the different elements influence responsiveness and how a user may differentiate them when debugging a network.

6.1. 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.

6.1.1. 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 can 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 can happen at the layers below the transport stack (e.g., in the outgoing network interface).

Each of these queue build-ups may cause delay and thus low responsiveness. A well designed networking stack ensures that queue-buildup in the TCP layer is kept at a bare minimum with solutions

like TCP_NOTSENT_LOWAT [RFC9293][SBM]. 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.

Flow-queueing techniques like FQ-Codel are valuable for insulating well behaved flows from badly behaved flows, but flow-queueing alone without intelligent queue management cannot insulate a flow from its own capacity-seeking behavior.

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. This is why the Responsiveness Test performs probes on load-generating connections, to detect and report the responsiveness experienced by capacity-seeking flows, not only the responsiveness experienced by low-rate flows.

6.1.2. Network Influence

The network obviously is a large driver for the responsiveness result. Propagation delay from the client to the server, waiting to access a shared resource like radio spectrum, queuing in the bottleneck node, and other sources of delay all add to latency [BITAG]. 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" and similar middleboxes. 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 AQM 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.

6.1.3. 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 primarily during the downlink load generation.

Beyond the server's networking stack influence, the geographic location of the server impacts the result as well. The network path chosen between client and server is influenced by the server selection mechanism. Modern content delivery networks (CDNs) generally have data centers in many locations around the world, partly to keep data closer to the user and reduce round-trip times. To evaluate the real-world responsiveness of a particular CDN, the RPM test should use the same server selection mechanism clients normally use to determine which server to contact when accessing content from that CDN. For a general-purpose RPM test that is intended to evaluate general network conditions, ideally the provider of the test should have test servers available that are distributed around the world, like a typical CDN. If such geographic distribution is not possible, it may be useful to account for the geographic distance, so separate the (unavoidable) speed-of-light delay from other (avoidable) network delays.

6.2. Investigating Poor Responsiveness

Once a responsiveness result has been generated, one might be motivated to try to localize the source of a 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 users to gain some insight into what is responsible for excessive delay. To gain this insight, implementations of the responsiveness test are encouraged to have an optional verbose mode that exposes the inner workings of the algorithm as well as statistics from the lower layers. The following is a non-exhaustive list of additional information that can be exposed in the verbose mode: Idle latency (measured at the beginning from the initial connections), achieved capacity on load-generating connections, TM(tcp_f), TM(tls_f), TM(http_f) and TM(http_l) (and even their raw values), HTTP-protocol (HTTP/1.1, HTTP/2, HTTP/3), transport protocol (TCP, QUIC, ...), congestion-control algorithm (Cubic, BBR, ...) used on the client-side, ECN or L4S statistics, IP version, ... and many more.

The previous section described the elements that influence responsiveness. It is apparent that the delay measured on the load-generating connections and the delay measured on separate connections may be different due to the different elements.

For example, if the delay measured on separate connections is much less than the delay measured on the load-generating connections, it is possible to narrow down the source of the additional delay on the load-generating connections. As long as the other elements of the network don't do flow-queueing, the additional delay 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.

Beyond the difference in the delay of the load-generating connections and the separate connections, another element can provide additional information: Namely, testing against different servers located in different places along the path will allow, to some extent, the opportunity to separate the network's path into different segments. For example, if the cable modem and a more distant ISP server are hosting responsiveness measurement endpoints, some localization of the issue can be done. If the RPM to the cable modem is very high, it means that the network segment from the client endpoint to the cable modem does not have responsiveness issues, thus allowing the user to conclude that possible responsiveness issues are beyond the cable modem. It must be noted, though, that due to the high level approach to the testing (including HTTP), a low responsiveness to the cable modem does not necessarily mean that the network between client and cable modem is the problem (as outlined in the above previous paragraphs).

7. Responsiveness Test Server

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 POST in the process of executing the test.

Both the client and the server MUST support HTTP/2 over TLS. The client and the server MAY also support HTTP/3 over QUIC. The client MUST be able to send a request with a GET or POST method. The client MUST send the GET with the "Accept-Encoding" header set to "identity" to ensure the server will not compress the data. The server MUST be able to respond to both of these HTTP commands. The server MUST have the ability to respond to a GET request with content.

The server MUST respond to 4 URLs:

1. A "small" URL/response: The server must respond with a status code of 200 (OK) and 1 byte of content. The actual message content is irrelevant. The server SHOULD specify the Content-Type header field with the media type "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 (OK) and content size of at least 8GB. The server SHOULD specify the Content-Type header field with the media type "application/octet-stream". The content can be larger, 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 conditions and making its measurements.

3. An "upload" URL/response: The server must handle a POST request with an arbitrary content size. The server should discard the content. The client SHOULD specify the Content-Type header field for the POST request with the media type "application/octet-stream". 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 conditions and making its measurements.

4. A .well-known URL [RFC8615] that serves a JSON object providing the three test URLs described above and other configuration information for the client to run the test (See Section 8, 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.

8. 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 Servers 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. "Something to do with my home Wi-Fi Access point?"
3. "Something to do with my home gateway?"
4. "Something to do with my 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 Servers 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, 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.

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

An organization or device that hosts a Responsiveness Test Server advertises that service using the network quality well-known URI [RFC8615]. The URI scheme is "https" and the application name is "nq" (e.g., https://example.com/.well-known/nq). No additional path components, query strings or fragments are valid for this well-known URI. The media type of the resource at the well-known URI is "application/json" and the format of the resource is a valid JSON object as specified below:

```
{
  "version": 1,
  "urls": {
    "large_download_url": "<URL for bulk download>",
    "small_download_url": "<URL for small download>",
    "upload_url":         "<URL for small upload>"
  }
  "test_endpoint":        "<hostname for server to use for testing>"
}
```

The fields can appear in any order.

The content of the "version" field SHALL be "1". Integer values greater than "1" are reserved for possible use in future updates to this protocol.

The content of the "large_download_url", "small_download_url", and "upload_url" SHALL all be validly formatted "http" or "https" URLs. All three URLs MUST contain the same <host> part so that they are eligible to be multiplexed onto the same TCP or QUIC connection. If the three URLs do not contain the same <host> part then the JSON configuration information object is invalid and the client MUST ignore the entire JSON object.

The "version" field and the three URLs are mandatory and each MUST appear exactly once in the JSON object. If any of these fields are missing or appear more than once, the configuration information is invalid and the entire JSON object MUST be ignored. If a client implementing the current version of this specification encounters a JSON configuration information object where the "version" field is not "1", then the client should assume that it is unable to safely parse the configuration information object (that's what incrementing the version field indicates) and the client MUST ignore the entire JSON object.

The "test_endpoint" field is OPTIONAL, and if present MUST appear exactly once in the JSON object. If the "test_endpoint" field appears more than once, the configuration information is invalid and the entire JSON object MUST be ignored. If present, then for the purpose of determining the IP address to which it should connect the test client MUST ignore the <host> part in the URLs and instead connect to one of the IP addresses indicated by the "test_endpoint" field, as determined by the test client's address resolution policy (e.g., Happy Eyeballs [RFC8305]). The test client then sends HTTP GET and POST requests (as determined by the test procedure) to the host indicated by the "test_endpoint" field, forming its HTTP requests using the <host> and <path> from the specified URLs. In other words, when the "test_endpoint" field is present the test client operates as if it were simply using the specified URLs directly, except that it behaves as if it had a local (e.g., "/etc/hosts") entry overriding the IP address(es) of the <host> given in the URLs to be the IP address(es) of the "test_endpoint" hostname instead. In the case of a large web site served by multiple load-balanced servers, this feature gives the administrator more precise control over which of those servers are used for responsiveness testing. In a situation where some of a site's servers have been configured to deliver low-delay HTTP responses and some have not, it can be useful to be able to measure the responsiveness of different servers with different configurations to see how they compare when handling identical HTTP GET and POST requests.

Servers implementing the current version of this specification SHOULD NOT include any names in the JSON configuration information other than those documented here. Future updates to this specification may define additional names in the JSON configuration information. To allow for these future backwards-compatible updates, clients implementing the current version of this specification MUST silently ignore any unrecognized names in the JSON configuration information they receive, and MUST process the required names as documented here, behaving as if the unrecognized names were not present.

8.2. DNS-Based Service Discovery for Test Server Discovery

To further aid the test client in discovering Responsiveness Test Servers, organizations or devices offering Responsiveness Test Servers MAY advertise their availability using DNS-Based Service Discovery [RFC6763] over Multicast DNS [RFC6762] or conventional unicast DNS [RFC1034], using the service type [RFC6335] "_nq._tcp".

When advertising over Multicast DNS, typically the device offering the test service generally advertises its own Multicast DNS records.

When advertising over unicast DNS, population of the appropriate DNS zone with the relevant unicast discovery records can be performed by having a human administrator manually enter the required records, or automatically using a Discovery Proxy [RFC8766].

8.2.1. Unicast DNS-SD 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 conventional PTR and SRV 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 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.

9. Security Considerations

The security considerations that apply to any Active Measurement of live paths are relevant here [RFC4656][RFC5357].

If server-side resource consumption is a concern, a server can choose to not reply or delay its response to the initial request for the configuration information through the .well-known URL.

10. DISCUSSION

We seek feedback from designers of delay-sensitive applications, such as on-line games and videoconferencing applications, about whether this test accurately predicts their real-world user experience.

As this document as evolved through multiple revisions, it has arrived at an algorithm that discards the worst 5% of round-trip measurements, and reports the arithmetic mean of the the best 95%, because this enabled the algorithm to generate results that are both quick to produce and consistent from one run to the next.

However, the BITAG “Latency Explained” report [BITAG] states that the 95th, 98th, or 99th percentile round-trip time is indicative of the behaviour of videoconferencing applications, which is more or less the opposite of the current Responsiveness Test.

While repeatability and convenience are both valuable properties of the test, we need to ensure that we have not sacrificed relevance in the pursuit of these two other goals.

If we cannot achieve all three goals in a single test, we may need to have two versions of the test: a “quick” version that runs in about ten seconds and gives an approximate “ball park” result, suitable for a user to determine quickly whether their Internet connection is good or terrible, and a “precise” version that runs for longer and gives highly accurate and repeatable results, suitable for an equipment vendor or network operator to use when advertising the quality of their offerings.

The current specification of the Responsiveness Test includes a number of configurable parameters. If we want the Responsiveness Test to produce a score that can be compared meaningfully between different hardware and different network operators, we need to specify required values for these configurable parameters. For engineering diagnostic purposes different parameter values may be useful, but for comparisons between vendors or operators we need IETF consensus on fixed standardized test parameters that everyone uses.

11. IANA Considerations

11.1. Well-Known URI

This specification registers the "nq" well-known URI in the "Well-Known URIs" registry [RFC5785].

URI suffix: nq

11.2. Service Name

IANA has added the following value to the "Service Name and Transport Protocol Port Number Registry".

Service Name:	nq
Transport Protocol:	TCP
Assignee:	Stuart Cheshire
Contact:	Stuart Cheshire
Description:	Network Quality test server endpoint

12. Acknowledgments

Special thanks go to Jeroen Schickendantz and Felix Gaudin for their enthusiasm around the project and the development of the Go responsiveness measurement tool and the librespeed implementation. We also thank Greg White, Lucas Pardue, Sebastian Moeller, Rich Brown, Erik Auerswald, Matt Mathis and Omer Shapira for their constructive feedback on the document.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

13.2. Informative References

[BITAG] Broadband Internet Technical Advisory Group, "Latency Explained", 2022, <https://www.bitag.org/latency-explained.php>.

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

[Cake] Hiland-Jrgensen, T., Taht, D., and J. Morton, "Piece of CAKE: A Comprehensive Queue Management Solution for Home Gateways", 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN) , n.d..

[L96] Cheshire, S., "It' s the Latency, Stupid", 1996, <http://stuartcheshire.org/rants/Latency.html>.

[Prague] De Schepper, K., Tilmans, O., Briscoe, B., and V. Goel, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrg-prague-congestion-control-04, 24 July 2024, <https://datatracker.ietf.org/doc/html/draft-briscoe-iccrg-prague-congestion-control-04>.

[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>.

[RFC1889] Audio-Video Transport Working Group, Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, DOI 10.17487/RFC1889, January 1996, <https://www.rfc-editor.org/info/rfc1889>.

[RFC4656] Shalunov, S., Teitelbaum, B., Karp, A., Boote, J., and M. Zekauskas, "A One-way Active Measurement Protocol (OWAMP)", RFC 4656, DOI 10.17487/RFC4656, September 2006, <https://www.rfc-editor.org/info/rfc4656>.

[RFC5357] Hedayat, K., Krzanowski, R., Morton, A., Yum, K., and J. Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)", RFC 5357, DOI 10.17487/RFC5357, October 2008, <https://www.rfc-editor.org/info/rfc5357>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <https://www.rfc-editor.org/info/rfc5785>.

[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>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

[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>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/info/rfc9114>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

[RFC9330] Briscoe, B., Ed., De Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture", RFC 9330, DOI 10.17487/RFC9330, January 2023, <https://www.rfc-editor.org/info/rfc9330>.

[SBM] Cheshire, S., "Source Buffer Management", Work in Progress, Internet-Draft, draft-cheshire-sbm-02, 16 March 2025, <https://datatracker.ietf.org/doc/html/draft-cheshire-sbm-02>.

Appendix A. Apache Traffic Server Example Configurations

Apache Traffic Server starting at version 9.1.0 supports configuration as a Responsiveness Test Server, using the generator and the statichit plugin.

This section shows fragments of sample server configurations to host a Responsiveness Test Server.

A.1. Single Server Configuration

Given a local file “config.example.com.json” with the following content:

```
{
  "version": 1,
  "urls": {
    "large_download_url":"https://www.example.com/large",
    "small_download_url":"https://www.example.com/small",
    "upload_url":        "https://www.example.com/upload"
  }
}
```

The sample remap configuration file then is:

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

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

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

map https://www.example.com/upload \
    http://localhost/ \
    @plugin=generator.so
```

A.2. Alternate Server Configuration

If a separate test_endpoint server is desired, then on the main www.example.com server(s) a JSON configuration file like the one shown below is used:

```
{
  "version": 1,
  "urls": {
    "large_download_url":"https://www.example.com/large",
    "small_download_url":"https://www.example.com/small",
    "upload_url":        "https://www.example.com/upload"
  }
  "test_endpoint": "nq-test-server.example.com"
}
```

On the main www.example.com server(s) a remap configuration file like the one shown below is used, to serve the JSON configuration file to clients:

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

On nq-test-server.example.com a remap configuration file like the one shown below is used, to handle the test downloads and uploads:

```
   map https://www.example.com/large \
       http://localhost/cache/8589934592/ \
       @plugin=generator.so

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

   map https://www.example.com/upload \
       http://localhost/ \
       @plugin=generator.so
```

Authors' Addresses

Christoph Paasch
Email: christoph.paasch+ietf@gmail.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

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