Stream: Independent Submission
RFC: 8802
Category: Informational
Published: July 2020
ISSN: 2070-1721
Authors: J.J. Aranda, *Nokia*; M. Cortés, *Nokia*; J. Salvachúa, *Univ. Politecnica de Madrid*; M. Narganes, *Tecnalia*
I. Martínez-Sarriegui, *Optiva Media*

RFC 8802
The Quality for Service (Q4S) Protocol

Abstract

This memo describes an application-level protocol for the communication of end-to-end QoS compliance information based on the HyperText Transfer Protocol (HTTP) and the Session Description Protocol (SDP). The Quality for Service (Q4S) protocol provides a mechanism to negotiate and monitor latency, jitter, bandwidth, and packet loss, and to alert whenever one of the negotiated conditions is violated.

Implementation details on the actions to be triggered upon reception/detection of QoS alerts exchanged by the protocol are out of scope of this document; it is either application dependent (e.g., act to increase quality or reduce bit-rate) or network dependent (e.g., change connection's quality profile).

This protocol specification is the product of research conducted over a number of years; it is presented here as a permanent record and to offer a foundation for future similar work. It does not represent a standard protocol and does not have IETF consensus.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8802.

Copyright Notice

Table of Contents

1. Introduction

The World Wide Web (WWW) is a distributed hypermedia system that has gained widespread acceptance among Internet users. Although WWW browsers support other, preexisting Internet application protocols, the primary protocol used between WWW clients and servers became the HyperText Transfer Protocol (HTTP) ([RFC7230], [RFC7231], [RFC7232], [RFC7233], [RFC7234], and [RFC7235]). Since then, HTTP over TLS (known as HTTPS and described in [RFC2818]) has become an imperative for providing secure and authenticated WWW access. The mechanisms described in this document are equally applicable to HTTP and HTTPS.

The ease of use of the Web has prompted its widespread employment as a client/server architecture for many applications. Many of such applications require the client and the server to be able to communicate with each other and exchange information with certain quality constraints.

Quality in communications at the application level consists of four measurable parameters:

Latency: The time a message takes to travel from source to destination. It may be approximated as RTT/2 (round-trip time), assuming the networks are symmetrical. In this context, we will consider the statistical median formula.

Jitter: Latency variation. There are some formulas to calculate jitter, and in this context, we will consider the arithmetic mean formula.

Bandwidth: Bit rate of communication. To ensure quality, a protocol must ensure the availability of the bandwidth needed by the application.

Packet loss: The percentage of packet loss is closely related to bandwidth and jitter. Packet loss affects bandwidth because a high packet loss sometimes implies retransmissions that also consumes extra bandwidth, other times the retransmissions are not achieved (for example, in video streaming over UDP), and the information received is less than the required bandwidth. In terms of jitter, a packet loss sometimes is seen by the destination as a larger time between arrivals, causing a jitter growth.

Any other communication parameter, such as throughput, is not a network parameter because it depends on protocol window size and other implementation-dependent aspects.

The Q4S protocol provides a mechanism for quality monitoring based on an HTTP syntax and the Session Description Protocol (SDP) in order to be easily integrated in the WWW, but it may be used by any type of application, not only those based on HTTP. Quality requirements may be needed by any type of application that communicates using any kind of protocol, especially those with real-time constraints. Depending on the nature of each application, the constraints may be different, leading to different parameter thresholds that need to be met.

Q4S is an application-level client/server protocol that continuously measures session quality for a given flow (or set of flows), end-to-end (e2e) and in real time; raising alerts if quality parameters are below a given negotiated threshold and sending recoveries when quality parameters are restored. Q4S describes when these notifications, alerts, and recoveries need to be sent and the entity receiving them. The actions undertaken by the receiver of the alert are out of scope of the protocol.

Q4S is session-independent from the application flows to minimize the impact on them. To perform the measurements, two control flows are created on both communication paths (forward and reverse directions).

This protocol specification is the product of research conducted over a number of years and is presented here as a permanent record and to offer a foundation for future similar work. It does not represent a standard protocol and does not have IETF consensus.

1.1. Scope

The purpose of Q4S is to measure end-to-end network quality in real time. Q4S does not transport any application data. This means that Q4S is designed to be used jointly with other transport protocols such as Real-time Transport Protocol (RTP) [RFC3550], Transmission Control Protocol (TCP) [RFC0793], QUIC [QUIC], HTTP [RFC7230], etc.

Some existent transport protocols are focused on real-time media transport and certain connection metrics are available, which is the case of RTP and RTP Control Protocol (RTCP) [RFC3550]. Other protocols such as QUIC provide low connection latencies as well as advanced congestion control. These protocols transport data efficiently and provide a lot of functionalities. However, there are currently no other quality measurement protocols offering the same level of function as Q4S. See Section 1.4 for a discussion of the IETF's quality measurement protocols, One-Way Active Measurement Protocol (OWAMP) and Two-Way Active Measurement Protocol (TWAMP).

Q4S enables applications to become reactive under e2e network quality changes. To achieve it, an independent Q4S stack application must run in parallel with the target application. Then, Q4S metrics may be used to trigger actions on the target application, such as speed adaptation to latency in multiuser games, bitrate control at streaming services, intelligent commutation of delivery node at Content Delivery Networks, and whatever the target application allows.

1.2. Motivation

Monitoring quality of service (QoS) in computer networks is useful for several reasons:

- It enables real-time services and applications to verify whether network resources achieve a certain QoS level. This helps real-time services and applications to run over the Internet, allowing the existence of Application Content Providers (ACPs), which offer guaranteed real-time services to the end users.
- Real-time monitoring allows applications to adapt themselves to network conditions (application-based QoS) and/or request more network quality from the Internet Service Provider (ISP) (if the ISP offers this possibility).
- Monitoring may also be required by peer-to-peer (P2P) real-time applications for which Q4S can be used.
- Monitoring enables ISPs to offer QoS to any ACP or end user application in an accountable way.
- Monitoring enables e2e negotiation of QoS parameters, independently of the ISPs of both endpoints.

A protocol to monitor QoS must address the following issues:

- Must be ready to be used in conjunction with current standard protocols and applications, without forcing a change on them.
- Must have a formal and compact way to specify quality constraints desired by the application to run.
- Must have measurement mechanisms that avoid application disruption and minimize network resources consumption.
- Must have specific messages to alert about the violation of quality constraints in different directions (forward and reverse) because network routing may not be symmetrical, and of course, quality constraints may not be symmetrical.
- After having alerted about the violation of quality constraints, must have specific messages to inform about the recovery of quality constraints in corresponding directions (forward and reverse).
- Must protect the data (constraints, measurements, QoS levels demanded from the network) in order to avoid the injection of malicious data in the measurements.

1.3. Summary of Features

The Quality for Service (Q4S) protocol is a message-oriented communication protocol that can be used in conjunction with any other application-level protocol. Q4S is a measurement protocol. Any action taken derived from its measurements are out of scope of the protocol. These actions depend on the application provider and may be application-level adaptive reactions, may involve requests to the ISP, or whatever the application provider decides.

The benefits in quality measurements provided by Q4S can be used by any type of application that uses any type of protocol for data transport. It provides a quality monitoring scheme for any communication that takes place between the client and the server, not only for the Q4S communication itself.

Q4S does not establish multimedia sessions, and it does not transport application data. It monitors the fulfillment of the quality requirements of the communication between the client and the server; therefore, it does not impose any restrictions on the type of application, protocol, or usage of the monitored quality connection.

Some applications may vary their quality requirements dynamically for any given quality parameter. Q4S is able to adapt to the changing application needs, modifying the parameter thresholds to the new values and monitoring the network quality according to the new quality constraints. It will raise alerts if the new constraints are violated.

The Q4S session lifetime is composed of four phases with different purposes: Handshake, Negotiation, Continuity, and Termination. Negotiation and Continuity phases perform network parameter measurements per a negotiated measurement procedure. Different measurement procedures could be used inside Q4S, although one default measurement mechanism is needed for compatibility reasons and is the one defined in this document. Basically, Q4S defines how to transport application quality requirements and measurement results between a client and server and how to provide monitoring and alerting, too.

Q4S must be executed just before starting a client-server application that needs a quality connection in terms of latency, jitter, bandwidth, and/or packet loss. Once the client and server have succeeded in establishing communication under quality constraints, the application can start, and Q4S continues measuring and alerting if necessary.

The quality parameters can be suggested by the client in the first message of the Handshake phase, but it is the server that accepts these parameter values or forces others. The server is in charge of deciding the final values of quality connection.

1.4. Differences from OWAMP/TWAMP

OWAMP [RFC4656] and TWAMP [RFC5357] are two protocols to measure network quality in terms of RTT, but they have a different goal than Q4S. The main difference is the scope: Q4S is designed to assist reactive applications, whereas OWAMP/TWAMP is designed to measure just network delay.

The differences can be summarized in the following points:

- OWAMP and TWAMP are not intended for measuring availability of resources (certain bandwidth availability, for example) but only RTT. However, Q4S is intended for measuring required bandwidth, packet loss, jitter, and latency in both directions. Available bandwidth is not measured by Q4S, but bandwidth required for a specific application is.
- OWAMP and TWAMP do not have responsivity control (which defines the speed of protocol reactions under network quality changes) because these protocols are designed to measure network performance, not to assist reactive applications, and do not detect the fluctuations of quality within certain time intervals to take reactive actions. However, responsivity control is a key feature of Q4S.
- OWAMP and TWAMP are not intended to run in parallel with reactive applications, but the Q4S protocol's goal is to run in parallel and assist reactive applications in making decisions based on Q4S-ALERT packets, which may trigger actions.

2. 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.

3. Overview of Operation

This section introduces the basic operation of Q4S using simple examples. This section is of a tutorial nature and does not contain any normative statements.

The first example shows the basic functions of Q4S: communication establishment between a client and a server, quality requirement negotiations for the requested application, application start and continuous quality parameter measurements, and finally communication termination.

The client triggers the establishment of the communication by requesting a specific service or application from the server. This first message must have a special URI [RFC3986], which may force the use of the Q4S protocol if it is implemented in a standard web browser. This message consists of a Q4S BEGIN method, which can optionally include a proposal for the communication quality requirements in an SDP body. This option gives the client a certain negotiation capacity about quality requirements, but it will be the server who finally decides the stated requirements.

This request is answered by the server with a Q4S 200 OK response letting the client know that it accepts the request. This response message must contain an SDP body with the following:

- The assigned Q4S sess-id.
- The quality constraints required by the requested application.
- The measurement procedure to use.

- "alerting-mode" attribute: There are two different scenarios for sending alerts that trigger actions either on the network or in the application when measurements identify violated quality constraints. In both cases, alerts are triggered by the server.
  (a) Q4S-aware-network scenario: The network is Q4S aware and reacts by itself to these alerts. In this scenario, Q4S-ALERT messages are sent by the server to the client, and network elements inspect and process these alert messages. The alerting mode in this scenario is called Q4S-aware-network alerting mode.

  (b) Reactive scenario: As shown in Figure 1, the network is not Q4S aware. In this scenario, alert notifications are sent to a specific node, called an Actuator, which is in charge of making decisions regarding what actions to trigger: either to change application behavior to adapt it to network conditions and/or invoke a network policy server in order to reconfigure the network and request better quality for application flows.

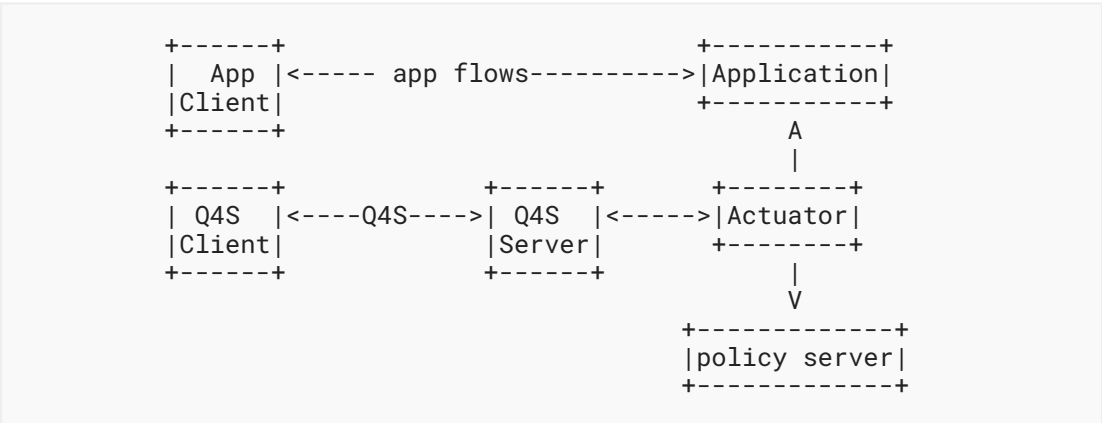


Figure 1: Reactive Scenario

The format of messages exchanged between the server stack and the Actuator doesn't follow Q4S codification rules; their format will be implementation dependent. In this way, we will call the messages sent from the server stack to the Actuator "notifications" (e.g., alert notifications) and the messages sent from the Actuator to the server stack in response to notifications "acknowledges" (e.g., alert acknowledges).

- "alert-pause" attribute: The amount of time between consecutive alerts. In the Q4S-aware-network scenario, the server has to wait this period of time between Q4S-ALERT messages sent to the client. In the Reactive scenario, the server stack has to wait this period of time between alert notifications sent to the Actuator. Measurements are not stopped in Negotiation or Continuity phases during this period of time, but no alerts are sent, even with violated network quality constraints, in order to leave time for network reconfiguration or for application adjustments.
- "recovery-pause" attribute: The amount of time the Q4S server waits before trying to recover the initial "qos-level" (Section 7.2.1). After having detected violation of quality constraints several times, the "qos-level" will have been increased accordingly. If this violation detection finally stops, the server waits for a period of time (recovery time), and if the situation

persists, it tries to recover to previous "qos-level" values gradually by sending Q4S-RECOVERY messages to the client in the Q4S-aware-network scenario, or recovery notifications to the Actuator in the Reactive scenario (Section 7.9).

It is important to highlight that any Q4S 200 OK response sent by the server to the client at any time during the life of a quality session may contain an SDP body with new values of quality constraints required by the application. Depending on the phase and the state of the measurement procedure within the specific phase, the client will react accordingly to take into account the new quality constraints in the measurement procedure.

Once the communication has been established (i.e., the Handshake phase is finished), the protocol will verify that the communication path between the client and the server meets the quality constraints in both directions, from and to the server (Negotiation phase). This Negotiation phase requires taking measurements of the quality parameters: latencies, jitter, bandwidth, and packet loss. This phase is initiated with a client message containing a Q4S READY method, which will be answered by the server with a Q4S 200 OK response.

Negotiation measurements are achieved in two sequential stages:

Stage 0: latency and jitter measurements

Stage 1: bandwidth and packet loss measurements

Stage 0 measurements are taken through Q4S PING messages sent from both the client and the server. All Q4S PING requests will be answered by Q4S 200 OK messages to allow for bidirectional measurements.

Different client and server implementations may send a different number of PING messages for measuring, although at least 255 messages should be considered to perform the latency measurement. The Stage 0 measurements only may be considered ended when neither client nor server receive new PING messages after an implementation-dependent guard time. Only after Stage 0 has ended, can the client send a "READY 1" message.

After a pre-agreed number of measurements have been performed, determined by the measurement procedure sent by the server, three scenarios may be possible:

(a) Measurements do not meet the requirements: in this case, the stage 0 is repeated after sending an alert from the server to the client or from the server stack to the Actuator, depending on the alerting mode defined in the Handshake phase. Notice that measurements continue to be taken but no alerts are sent during the "alert-pause" time. In the Reactive scenario, the Actuator will decide either to forward the alert notification to the network policy server or to the application, depending on where reconfiguration actions have to be taken.

(b) Measurements do meet the requirements: in this case, client moves to stage 1 by sending a new READY message.

(c) At any time during the measurement procedure, the Q4S 200 OK message sent by the server to the client, in response to a Q4S PING message, contains an SDP body with new values of quality constraints required by the application. This means the application has varied their quality requirements dynamically; therefore, quality thresholds used while monitoring quality parameters have to be changed to the new constraints. In this case, the client moves to the beginning of Stage 0 for initiating the negotiation measurements again.

Stage 1 is optional. Its purpose is to measure the availability of application-needed bandwidth. If the "bandwidth" attribute is set to zero kbps in the SDP, the client can skip stage 1 by sending a "READY 2" message after completion of stage 0. Stage 1 measurements are achieved through Q4S BWIDTH messages sent from both the client and the server. Unlike PING messages, Q4S BWIDTH requests will not be answered.

If Stage 0 and 1 meet the application quality constraints, the application may start. Q4S will enter the Continuity phase by measuring the network quality parameters through the Q4S PING message exchange on both connection paths and raising alerts in case of violation.

Once the client wants to terminate the quality session, it sends a Q4S CANCEL message, which will be acknowledged by the server with another Q4S CANCEL message. Termination of quality sessions are always initiated by the client because Q4S TCP requests follow the client/server schema.

Figure 2 depicts the message exchange in a successful scenario.

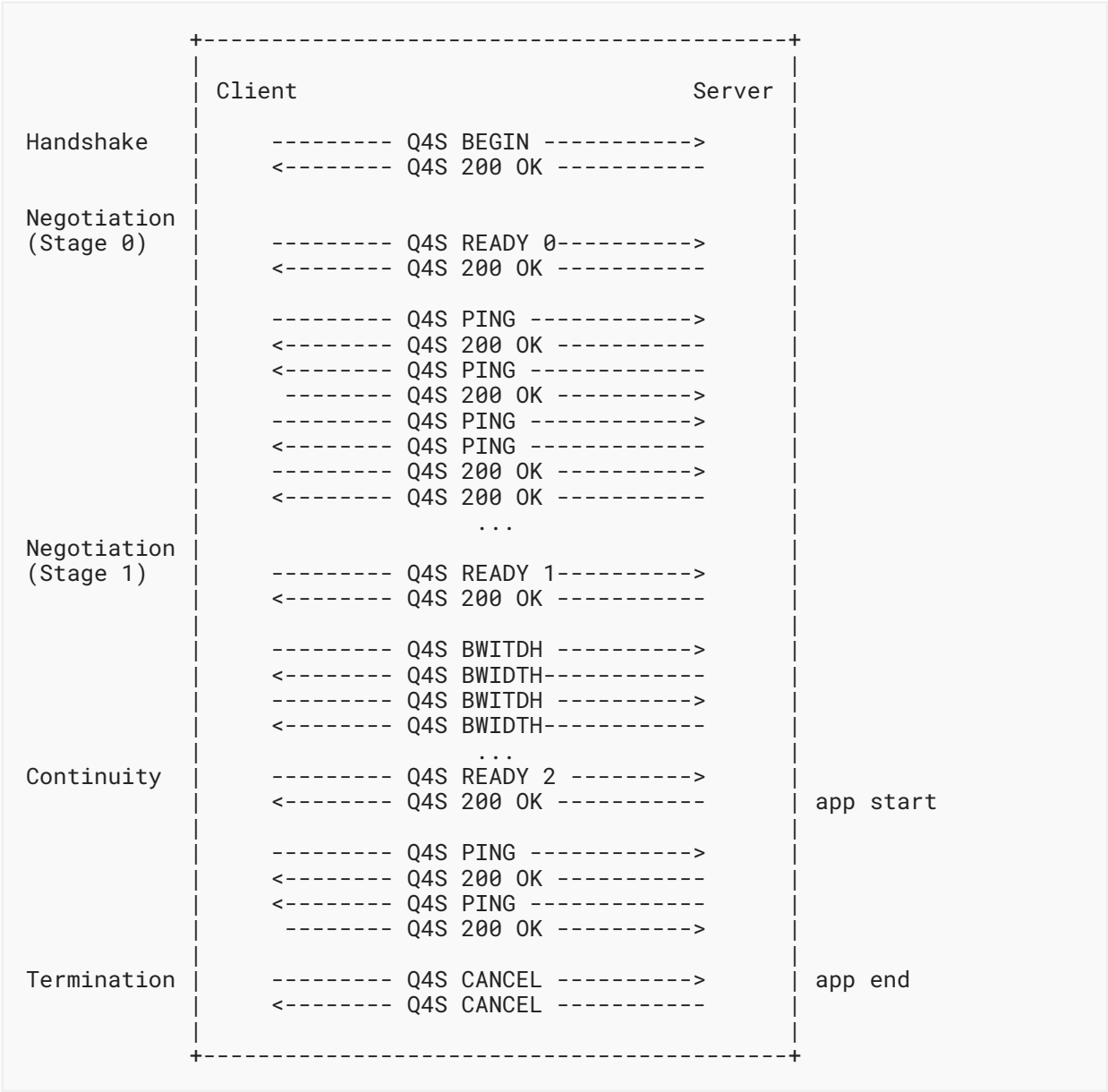
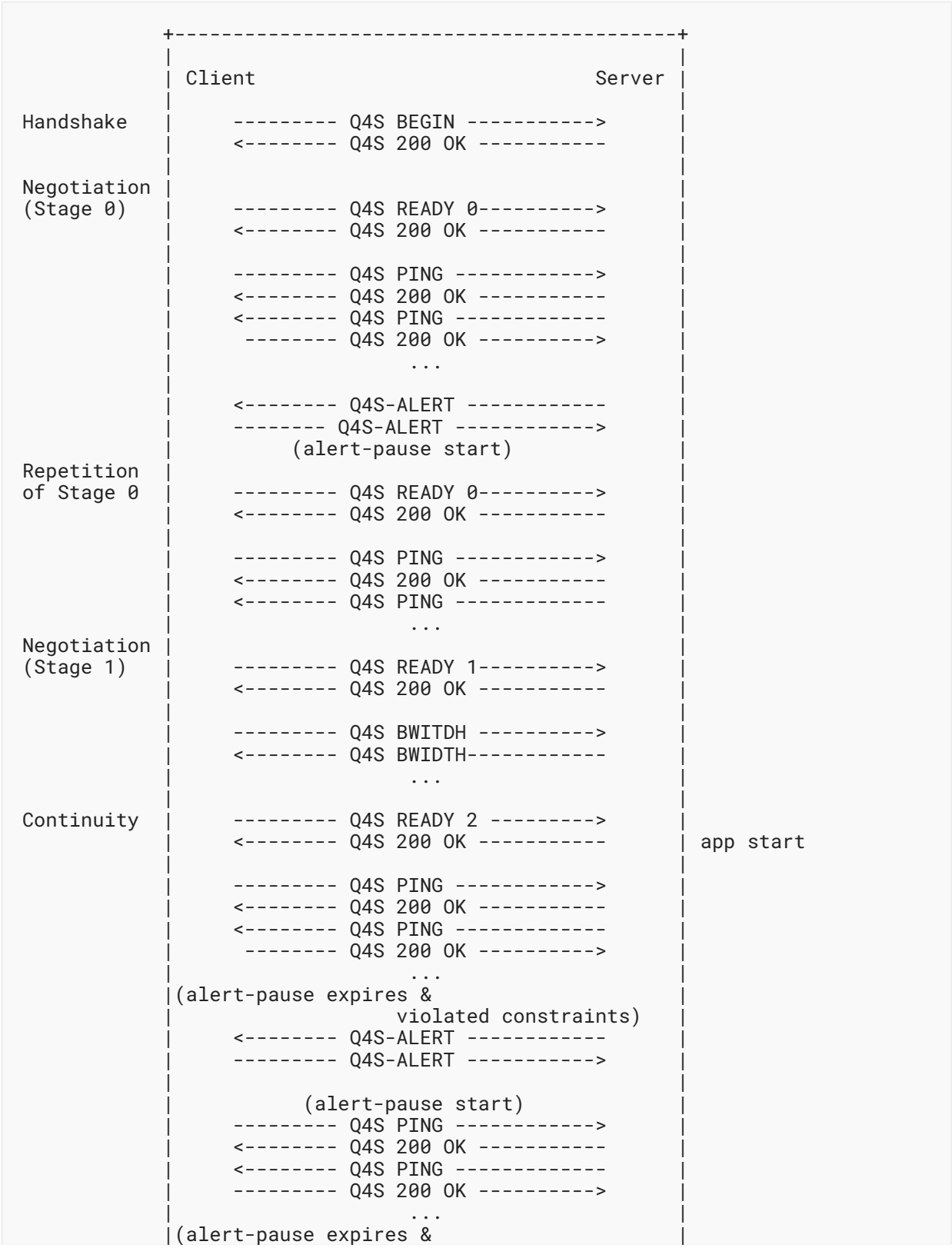


Figure 2: Successful Q4S Message Exchange

Both client and server measurements are included in the PING and BWIDTH messages, allowing both sides of the communication channel to be aware of all measurements in both directions.

The following two examples show the behavior of the Q4S protocol when quality constraints are violated, and alerts are generated; and, later on, when the violation of quality constraints stops leading to the execution of the recovery process. The first example (Figure 3) shows the Q4S-aware-network alerting mode scenario:


Client
Server
Handshake
Q4S BEGIN
Q4S 200 OK
Negotiation (Stage 0)
Q4S READY 0
Q4S 200 OK
Q4S PING
Q4S 200 OK
Q4S PING
Q4S 200 OK
...
Q4S-ALERT
Q4S-ALERT
(alert-pause start)
Repetition of Stage 0
Q4S READY 0
Q4S 200 OK
Q4S PING
Q4S 200 OK
Q4S PING
...
Negotiation (Stage 1)
Q4S READY 1
Q4S 200 OK
Q4S BWITDH
Q4S BWIDTH
...
Continuity
Q4S READY 2
Q4S 200 OK
app start
Q4S PING
Q4S 200 OK
Q4S PING
Q4S 200 OK
...
(alert-pause expires &
violated constraints)
Q4S-ALERT
Q4S-ALERT
(alert-pause start)
Q4S PING
Q4S 200 OK
Q4S PING
Q4S 200 OK
...
(alert-pause expires &

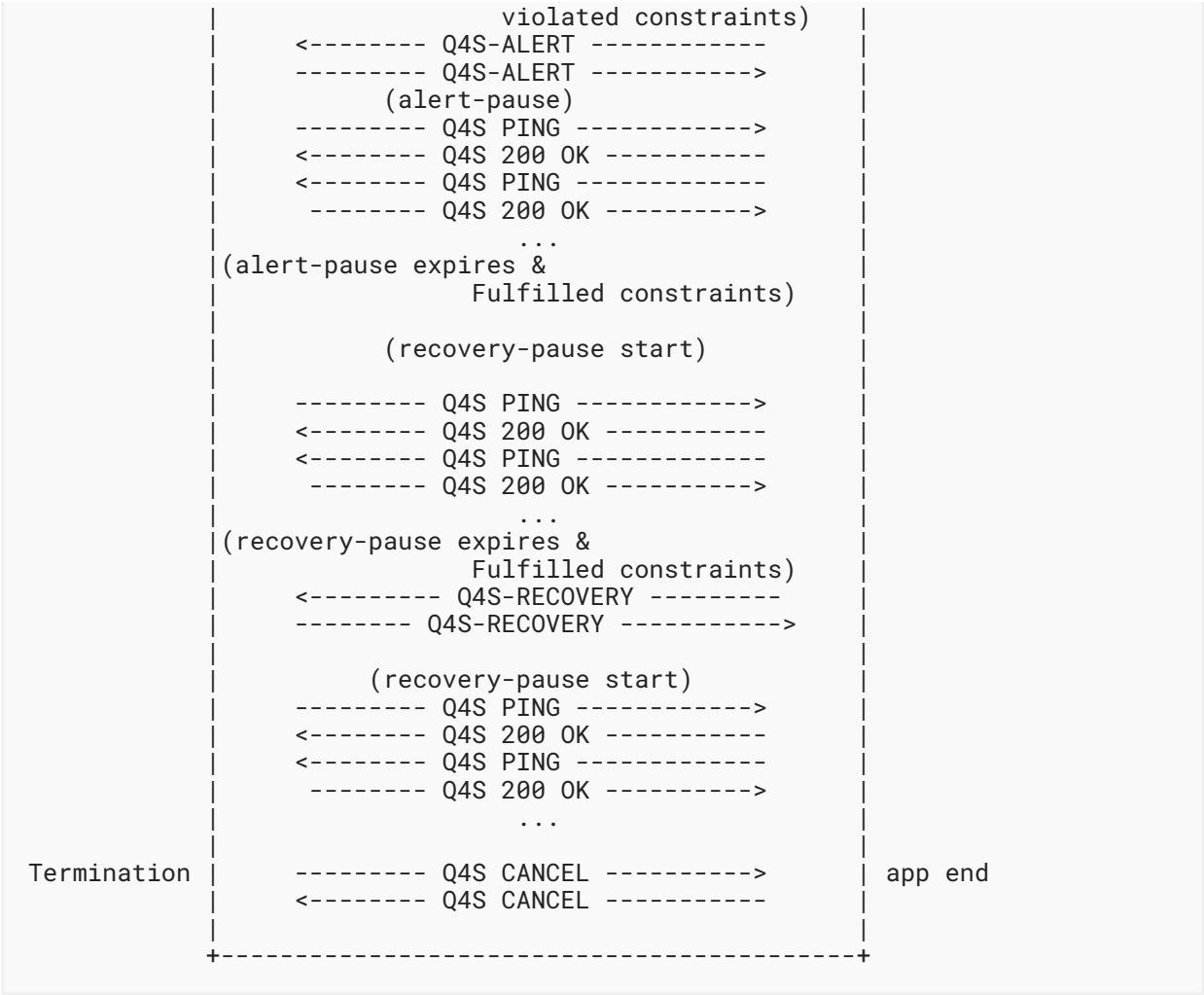
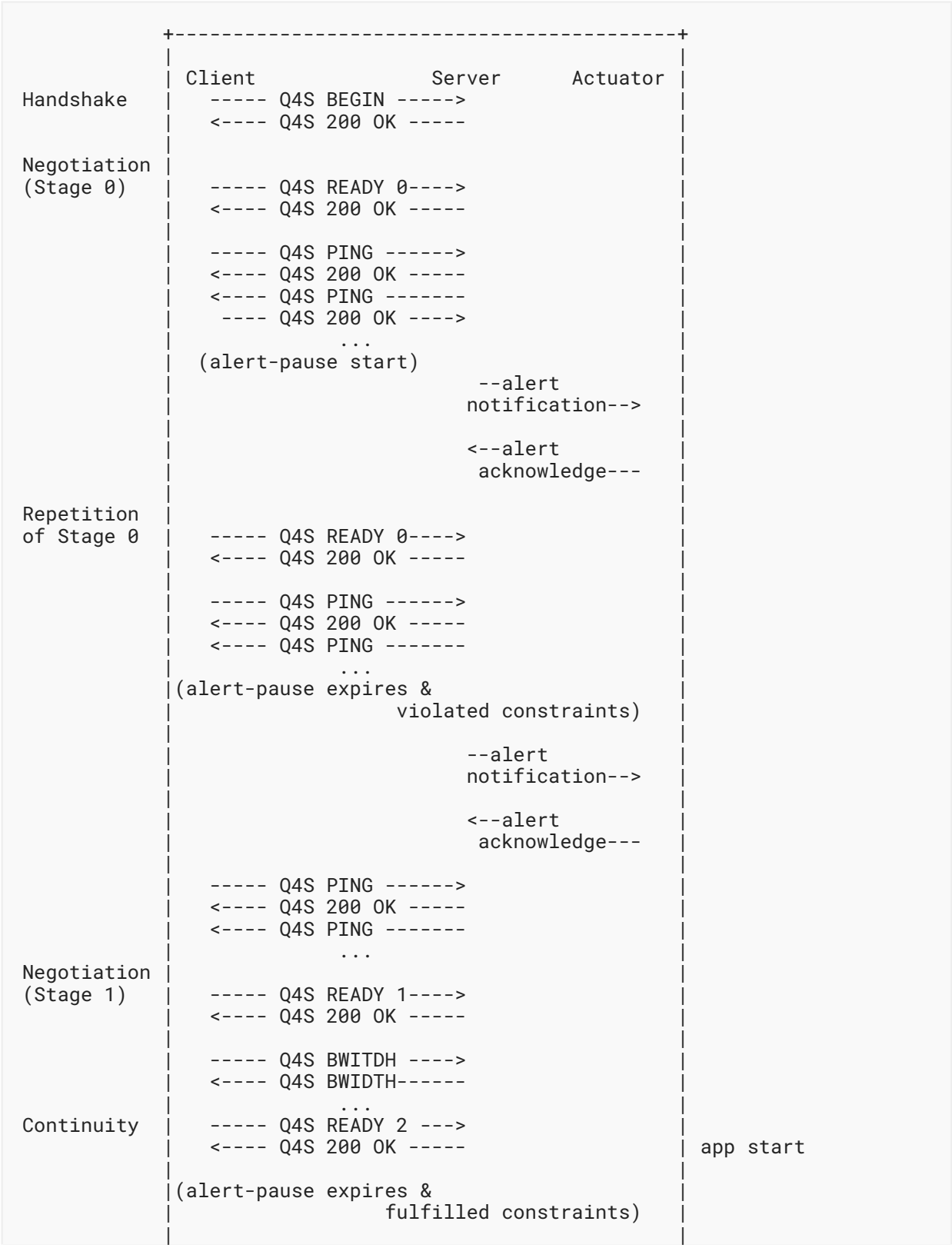


Figure 3: Q4S-Aware-Network Alerting Mode

In this Q4S-aware-network alerting mode scenario, the server may send Q4S alerts to the client at any time upon detection of violated quality constraints. This alerting exchange must not interrupt the continuity quality parameter measurements between client and server.

The second example depicted in Figure 4 represents the Reactive scenario, in which alert notifications are sent from the server stack to the Actuator, which is in charge of deciding to act over application behavior and/or to invoke a network policy server. The Actuator is an entity that has a defined set of different quality levels and decides how to act depending on the actions stated for each of these levels; it can take actions for making adjustments on the application, or it can send a request to the policy server for acting on the network. The policy server also has a defined set of different quality levels previously agreed upon between the Application Content Provider and the ISP. The Reactive alerting mode is the default mode.


Client
Server
Actuator
Handshake
Q4S BEGIN
Q4S 200 OK
Negotiation (Stage 0)
Q4S READY 0
Q4S 200 OK
Q4S PING
Q4S 200 OK
Q4S PING
Q4S 200 OK
...
(alert-pause start)
--alert notification-->
<--alert acknowledge---
Repetition of Stage 0
Q4S READY 0
Q4S 200 OK
Q4S PING
Q4S 200 OK
Q4S PING
...
(alert-pause expires & violated constraints)
--alert notification-->
<--alert acknowledge---
Q4S PING
Q4S 200 OK
Q4S PING
...
Negotiation (Stage 1)
Q4S READY 1
Q4S 200 OK
Q4S BWITDH
Q4S BWIDTH
...
Continuity
Q4S READY 2
Q4S 200 OK
app start
(alert-pause expires & fulfilled constraints)

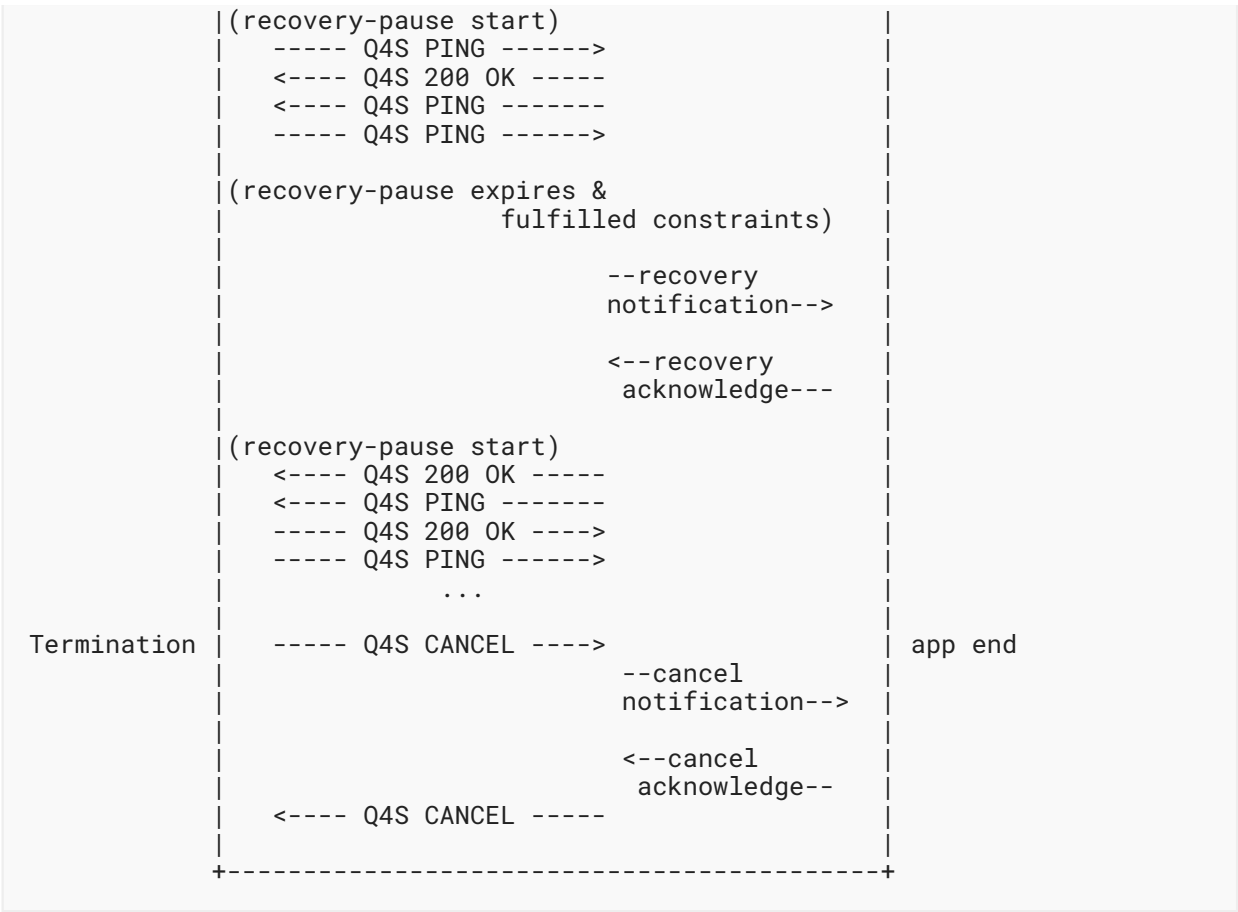


Figure 4: Reactive Alerting Mode

At the end of any stage of the Negotiation phase, the server sends an alert notification to the Actuator if quality constraints are violated. During the period of time defined by the "alert-pause" attribute, no further alert notifications are sent, but measurements are not interrupted. This way, both the client and the server will detect network improvements as soon as possible. In a similar way during the Continuity phase, the server may send alert notifications at any time to the Actuator upon detection of violated quality constraints. This alerting exchange must not interrupt the continuity measurements between client and server.

Finally, in the Termination phase, Q4S CANCEL messages sent from the client to the server must be forwarded from the server to the Actuator in order to release possible assigned resources for the session.

4. Q4S Messages

Q4S is a text-based protocol and uses the UTF-8 charset [RFC3629]. A Q4S message is either a request or a response.

Both request and response messages use the basic format of Internet Message Format [RFC5322]. Both types of messages consist of a start-line, one or more header fields, an empty line indicating the end of the header fields, and an optional message-body. This document uses ABNF notation [RFC5234] for the definitions of the syntax of messages.

The start-line, each message-header line, and the empty line **MUST** be terminated by a carriage-return line-feed sequence (CRLF). Note that the empty line **MUST** be present even if the message-body is not.

```
      generic-message  =  start-line CRLF
                          *message-header CRLF
                          CRLF
                          [ message-body ]
      start-line       =  Request-Line / Status-Line
```

Much of Q4S's messages and header field syntax are identical to HTTP/1.1. However, Q4S is not an extension of HTTP.

4.1. Requests

Q4S requests are distinguished by having a Request-Line for a start-line. A Request-Line contains a method name, a Request-URI, and the protocol version separated by a single space (SP) character.

The Request-Line ends with CRLF. No CR or LF are allowed except in the end-of-line CRLF sequence. No linear whitespace (LWSP) is allowed in any of the elements.

```
    Request-Line  =  Method SP Request-URI SP Q4S-Version CRLF
```

Method: This specification defines seven methods: BEGIN for starting and negotiating quality sessions, READY for synchronization of measurements, PING and BWIDTH for quality measurements purposes, CANCEL for terminating sessions, Q4S-ALERT for reporting quality violations, and Q4S-RECOVERY for reporting quality recovery.

Request-URI: The Request-URI is a Q4S URI [RFC3986] as described in Section 7.4. The Request-URI **MUST NOT** contain unescaped spaces or control characters and **MUST NOT** be enclosed in "<>".

Q4S-Version: Both request and response messages include the version of Q4S in use. To be compliant with this specification, applications sending Q4S messages **MUST** include a Q4S-Version of "Q4S/1.0". The Q4S-Version string is case insensitive, but implementations **MUST** send uppercase. Unlike HTTP/1.1, Q4S treats the version number as a literal string. In practice, this should make no difference.

4.2. Responses

Q4S responses are distinguished from requests by having a Status-Line as their start-line. A Status-Line consists of the protocol version followed by a numeric Status-Code and its associated textual phrase, with each element separated by a single SP character. No CR or LF is allowed except in the final CRLF sequence.

```
   Status-Line  =  Q4S-Version SP Status-Code SP Reason-Phrase CRLF
```

The Status-Code is a 3-digit integer result code that indicates the outcome of an attempt to understand and satisfy a request. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata, whereas the Reason-Phrase is intended for the human user. A client is not required to examine or display the Reason-Phrase.

While this specification suggests specific wording for the Reason-Phrase, implementations **MAY** choose other text, for example, in the language indicated in the Accept-Language header field of the request.

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. For this reason, any response with a status code between 100 and 199 is referred to as a "1xx response", any response with a status code between 200 and 299 as a "2xx response", and so on. Q4S/1.0 allows following values for the first digit:

1xx: Provisional -- request received, continuing to process the request;

2xx: Success -- the action was successfully received, understood, and accepted;

3xx: Redirection -- further action needs to be taken in order to complete the request;

4xx: Request Failure -- the request contains bad syntax or cannot be fulfilled at this server;

5xx: Server Error -- the server failed to fulfill an apparently valid request;

6xx: Global Failure -- the request cannot be fulfilled at any server.

The status codes are the same as described in HTTP [RFC7231]. In the same way as HTTP, Q4S applications are not required to understand the meaning of all registered status codes, though such understanding is obviously desirable. However, applications **MUST** understand the class of any status code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 status code of that class.

The Q4S-ALERT, Q4S-RECOVERY, and CANCEL requests do not have to be responded to. However, after receiving a Q4S-ALERT, Q4S-RECOVERY, or CANCEL request, the server **SHOULD** send a Q4S-ALERT, Q4S-RECOVERY, or CANCEL request to the client.

4.3. Header Fields

Q4S header fields are identical to HTTP header fields in both syntax and semantics.

Some header fields only make sense in requests or responses. These are called request header fields and response header fields, respectively. If a header field appears in a message that does not match its category (such as a request header field in a response), it **MUST** be ignored.

4.3.1. Common Q4S Header Fields

These fields may appear in request and response messages.

Session-Id: the value for this header field is the same sess-id used in SDP (embedded in the SDP "o=" line) and is assigned by the server. The messages without SDP **MUST** include this header field. If a message has an SDP body, this header field is optional. The method of sess-id allocation is up to the creating tool, but it is suggested that a UTC timestamp be used to ensure uniqueness.

Sequence-Number: sequential and cyclic positive integer number assigned to PING and BWIDTH messages and acknowledged in 200 OK responses.

Timestamp: this optional header field contains the system time (with the best possible accuracy). It indicates the time in which the PING request was sent. If this header field is present in PING messages, then the 200 OK response messages **MUST** include this value.

Stage: this is used in the client's READY requests and the server's 200 OK responses during the Negotiation and Continuity phases in order to synchronize the initiation of the measurements. Example: Stage: 0

4.3.2. Specific Q4S Request Header Fields

In addition to HTTP header fields, these are the specific Q4S request header fields:

User-Agent: this header field contains information about the implementation of the user agent. This is for statistical purposes, the tracing of protocol violations, and the automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. User agents **SHOULD** include this field with requests. The field **MAY** contain multiple product tokens and comments identifying the agent and any sub-products that form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application.

Signature: this header field contains a digital signature that can be used by the network, Actuator, or policy server to validate the SDP, preventing security attacks. The Signature is an optional header field generated by the server according to the pre-agreed security policies between the Application Content Provider and the ISP. For example, a hash algorithm and encryption method such as SHA256 [RFC6234] and RSA [RFC8017] based on

the server certificate could be used. This certificate is supposed to be delivered by a Certification Authority (CA) or policy owner to the server. The signature is applied to the SDP body.

```
Signature= RSA ( SHA256 (<sdp>), <certificate> )
```

If the Signature header field is not present, other validation mechanisms **MAY** be implemented in order to provide assured quality with security and control.

Measurements: this header field carries the measurements of the quality parameters in PING and BWIDTH requests. The format is:

```
Measurements: "l=" " "|[0..9999] ", j=" " "|[0..9999] ", pl="
" "|[0.00 .. 100.00] ", bw=" " "|[0..999999]
```

Where "l" stands for latency followed by the measured value (in milliseconds) or an empty space, "j" stands for jitter followed by the measured value (in milliseconds) or an empty space, "pl" stands for packet loss followed by the measured value (in percentage with two decimals) or an empty space, and "bw" stands for bandwidth followed by the measured value (in kbps) or an empty space.

4.3.3. Specific Q4S Response Header Fields

Expires: its purpose is to provide a sanity check and allow the server to close inactive sessions. If the client does not send a new request before the expiration time, the server **MAY** close the session. The value **MUST** be an integer, and the measurement units are milliseconds.

In order to keep the session open, the server **MUST** send a Q4S alert before the session expiration (Expires header field), with the same quality levels and an alert cause of "keep-alive". The purpose of this alert is to avoid TCP sockets, which were opened with READY message, from being closed, specially in NAT scenarios.

4.4. Bodies

Requests, including new requests defined in extensions to this specification, **MAY** contain message bodies unless otherwise noted. The interpretation of the body depends on the request method.

For response messages, the request method and the response status code determine the type and interpretation of any message body. All responses **MAY** include a body.

The Internet media type of the message body **MUST** be given by the Content-Type header field.

4.4.1. Encoding

The body **MUST NOT** be compressed. This mechanism is valid for other protocols such as HTTP and SIP [RFC3261], but a compression/coding scheme will limit the way the request is parsed to certain logical implementations, thus making the protocol concept more implementation dependent. In addition, the bandwidth calculation may not be valid if compression is used. Therefore, the HTTP Accept-Encoding request header field cannot be used in Q4S with values different from "identity", and if it is present in a request, the server **MUST** ignore it. In addition, the response header field Content-Encoding is optional, but if present, the unique permitted value is "identity".

The body length in bytes **MUST** be provided by the Content-Length header field. The "chunked" transfer encoding of HTTP/1.1 **MUST NOT** be used for Q4S.

> Note: The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each one with its own size indicator.

5. Q4S Method Definitions

The Method token indicates the method to be performed on the resource identified by the Request-URI. The method is case sensitive.

```
Method  = "BEGIN" | "READY" | "PING" | "BWIDTH" |
          "Q4S-ALERT" | "Q4S-RECOVERY" | "CANCEL" | extension-method

extension-method = token
```

The list of methods allowed by a resource can be specified in an Allow header field [RFC7231]. The return code of the response always notifies the client when a method is currently allowed on a resource, since the set of allowed methods can change dynamically. Any server application **SHOULD** return the status code 405 (Method Not Allowed) if the method is known, but not allowed for the requested resource, and 501 (Not Implemented) if the method is unrecognized or not implemented by the server.

5.1. BEGIN

The BEGIN method requests information from a resource identified by a Q4S URI. The purpose of this method is to start the quality session.

This method is used only during the Handshake phase to retrieve the SDP containing the sess-id and all quality and operation parameters for the desired application to run.

When a BEGIN message is received by the server, any current quality session **MUST** be canceled, and a new session should be created.

The response to a Q4S BEGIN request is not cacheable.

5.2. READY

The READY method is used to synchronize the starting time for the sending of PING and BWIDTH messages over UDP between clients and servers. Including the Stage header field in this method is mandatory.

This message is used only in Negotiation and Continuity phases, and only just before making a measurement. Otherwise (outside of this context), the server **MUST** ignore this method.

5.3. PING

This message is used during the Negotiation and Continuity phases to measure the RTT and jitter of a session. The message **MUST** be sent only over UDP ports.

The fundamental difference between the PING and BWIDTH requests is reflected in the different measurements achieved with them. PING is a short message, and it **MUST** be answered in order to measure RTT and jitter, whereas BWIDTH is a long message and **MUST NOT** be answered.

PING is a request method that can be originated by either the client or the server. The client **MUST** also answer the server PING messages, assuming a "server role" for these messages during the measurement process.

Including the Measurements header field in this method is mandatory, and provides updated measurements values for latency, jitter, and packet loss to the counterpart.

5.4. BWIDTH

This message is used only during the Negotiation phase to measure the bandwidth and packet loss of a session. The message **MUST** be sent only over UDP ports.

BWIDTH is a request method that can be originated by either the client or the server. Both client and server **MUST NOT** answer BWIDTH messages.

Including the Measurements header field in this method is mandatory and provides updated measurements values for bandwidth and packet loss to the counterpart.

5.5. Q4S-ALERT

This is the request message that Q4S generates when the measurements indicate that quality constraints are being violated. It is used during the Negotiation and Continuity phases.

This informative message indicates that the user experience is being degraded and includes the details of the problem (bandwidth, jitter, packet loss measurements). The Q4S-ALERT message does not contain any detail on the actions to be taken, which depend on the agreements between all involved parties.

Unless there is an error condition, an answer to a Q4S-ALERT request is optional and is formatted as a request Q4S-ALERT message. If there is an error condition, then a response message is sent. The response to a Q4S-ALERT request is not cacheable.

This method **MUST** be initiated by the server in both alerting modes. In the Q4S-aware-network alerting mode, the Q4S-ALERT messages are sent by the server to the client, advising the network to react by itself. In the Reactive alerting mode, alert notifications are triggered by the server stack and sent to the Actuator (see Figure 1, "Reactive Scenario").

```
Client----q4s----SERVER STACK--->ACTUATOR-->APP OR POLICY SERVER
```

The way in which the server stack notifies the Actuator is implementation dependent, and the communication between the Actuator and the network policy server is defined by the protocol and API that the policy server implements.

5.6. Q4S-RECOVERY

This is the request message that Q4S generates when the measurements indicate that quality constraints, which had been violated, have been fulfilled during a period of time ("recovery-pause"). It is used during the Negotiation and Continuity phases.

This informative message indicates that the "qos-level" could be increased gradually until the initial "qos-level" is recovered (the "qos-level" established at the beginning of the session that was decreased during violation of constraints. See Section 7.9). The Q4S-RECOVERY message does not contain any detail on the actions to be taken, which depends on the agreements between all involved parties.

The answer to a Q4S-RECOVERY request is formatted as a request Q4S-RECOVERY message. A Q4S-RECOVERY request **MUST NOT** be answered with a response message unless there is an error condition. The response to a Q4S-RECOVERY request is not cacheable.

Like the Q4S-ALERT message, the Q4S-RECOVERY method is always initiated by the server in both alerting modes. In the Q4S-aware-network alerting mode, the Q4S-RECOVERY messages are sent by the server to the client, advising the network to react by itself. In the Reactive alerting mode, recovery notifications are triggered by the server stack and sent to the Actuator (see Figure 1, "Reactive Scenario").

5.7. CANCEL

The purpose of the CANCEL message is the release of the Q4S Session-Id and the possible resources assigned to the session. This message could be triggered by the Q4S stack or by the application using the stack (through an implementation-dependent API).

In the same way as Q4S-ALERT, CANCEL must not be answered with a response message, but with an answer formatted as a request Q4S-CANCEL message.

In the Reactive scenario, the server stack **MUST** react to the Q4S CANCEL messages received from the client by forwarding a cancel notification to the Actuator, in order to release possible assigned resources for the session (at the application or at the policy server). The Actuator **MUST** answer the cancel notification with a cancel acknowledge towards the server stack, acknowledging the reception.

6. Response Codes

Q4S response codes are used for TCP and UDP. However, in UDP, only the response code 200 is used.

The receiver of an unknown response code must take a generic action for the received error group (1xx, 2xx, 3xx, 4xx, 5xx, 6xx). In case of an unknown error group, the expected action should be the same as with the 6xx error group.

6.1. 100 Trying

This response indicates that the request has been received by the next-hop server and that some unspecified action is being taken on behalf of this request (for example, a database is being consulted). This response, like all other provisional responses, stops retransmissions of a Q4S-ALERT during the "alert-pause" time.

6.2. Success 2xx

2xx responses give information about the success of a request.

6.2.1. 200 OK

The request has succeeded.

6.3. Redirection 3xx

3xx responses give information about the user's new location or about alternative services that might be able to satisfy the request.

The requesting client **SHOULD** retry the request at the new address(es) given by the Location header field.

6.4. Request Failure 4xx

4xx responses are definite failure responses from a particular server. The client **SHOULD NOT** retry the same request without modification (for example, adding appropriate header fields or SDP values). However, the same request to a different server might be successful.

6.4.1. 400 Bad Request

The request could not be understood due to malformed syntax. The Reason-Phrase **SHOULD** identify the syntax problem in more detail, for example, "Missing Sequence-Number header field".

6.4.2. 404 Not Found

The server has definitive information that the user does not exist at the domain specified in the Request-URI. This status is also returned if the domain in the Request-URI does not match any of the domains handled by the recipient of the request.

6.4.3. 405 Method Not Allowed

The method specified in the Request-Line is understood, but not allowed for the address identified by the Request-URI.

The response **MUST** include an Allow header field containing a list of valid methods for the indicated address.

6.4.4. 406 Not Acceptable

The resource identified by the request is only able to generate response entities that have content characteristics that are not acceptable according to the Accept header field sent in the request.

6.4.5. 408 Request Timeout

The server could not produce a response within a suitable amount of time, and the client **MAY** repeat the request without modifications at any later time.

6.4.6. 413 Request Entity Too Large

The server is refusing to process a request because the request entity-body is larger than the one that the server is willing or able to process. The server **MAY** close the connection to prevent the client from continuing the request.

6.4.7. 414 Request-URI Too Long

The server is refusing to process the request because the Request-URI is longer than the one that the server accepts.

6.4.8. 415 Unsupported Media Type

The server is refusing to process the request because the message body of the request is in a format not supported by the server for the requested method. The server **MUST** return a list of acceptable formats using the Accept, Accept-Encoding, or Accept-Language header field, depending on the specific problem with the content.

6.4.9. 416 Unsupported URI Scheme

The server cannot process the request because the scheme of the URI in the Request-URI is unknown to the server.

6.5. Server Failure 5xx

5xx responses are failure responses given when a server itself is having trouble.

6.5.1. 500 Server Internal Error

The server encountered an unexpected condition that prevented it from fulfilling the request. The client **MAY** display the specific error condition and **MAY** retry the request after several seconds.

6.5.2. 501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when a server does not recognize the request method, and it is not capable of supporting it for any user.

Note that a 405 (Method Not Allowed) is sent when the server recognizes the request method, but that method is not allowed or supported.

6.5.3. 503 Service Unavailable

The server is temporarily unable to process the request due to a temporary overloading or maintenance of the server. The server **MAY** indicate when the client should retry the request in a Retry-After header field. If no Retry-After is given, the client **MUST** act as if it had received a 500 (Server Internal Error) response.

A client receiving a 503 (Service Unavailable) **SHOULD** attempt to forward the request to an alternate server. It **SHOULD NOT** forward any other requests to that server for the duration specified in the Retry-After header field, if present.

Servers **MAY** refuse the connection or drop the request instead of responding with 503 (Service Unavailable).

6.5.4. 504 Server Time-Out

The server did not receive a timely response from an external server it accessed in attempting to process the request.

6.5.5. 505 Version Not Supported

The server does not support, or refuses to support, the Q4S protocol version that was used in the request. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, other than with this error message.

In the case that the Q4S version is not supported, this error may be sent by the server in the Handshake phase just after receiving the first BEGIN message from client.

6.5.6. 513 Message Too Large

The server was unable to process the request because the message length exceeded its capabilities.

6.6. Global Failures 6xx

6xx responses indicate that a server has definitive information about a particular policy not satisfied for processing the request.

6.6.1. 600 Session Does Not Exist

The Session-Id is not valid.

6.6.2. 601 Quality Level Not Allowed

The "qos-level" requested is not allowed for the client/server pair.

6.6.3. 603 Session Not Allowed

The session is not allowed due to some policy (the number of sessions allowed for the server is exceeded, or the time band of the Q4S-ALERT is not allowed for the client/server pair, etc.).

6.6.4. 604 Authorization Not Allowed

The policy server does not authorize the Q4S-ALERT quality session improvement operation due to an internal or external reason.

7. Protocol

This section describes the measurement procedures, the SDP structure of the Q4S messages, the different Q4S protocol phases, and the messages exchanged in them.

7.1. Protocol Phases

All elements of the IP network contribute to quality in terms of latency, jitter, bandwidth, and packet loss. All these elements have their own quality policies in terms of priorities, traffic mode, etc., and each element has its own way to manage the quality. The purpose of a quality connection is to establish end-to-end communication with enough quality for the application to function flawlessly.

To monitor quality constraints of the application, four phases are defined and can be seen in Figure 5:

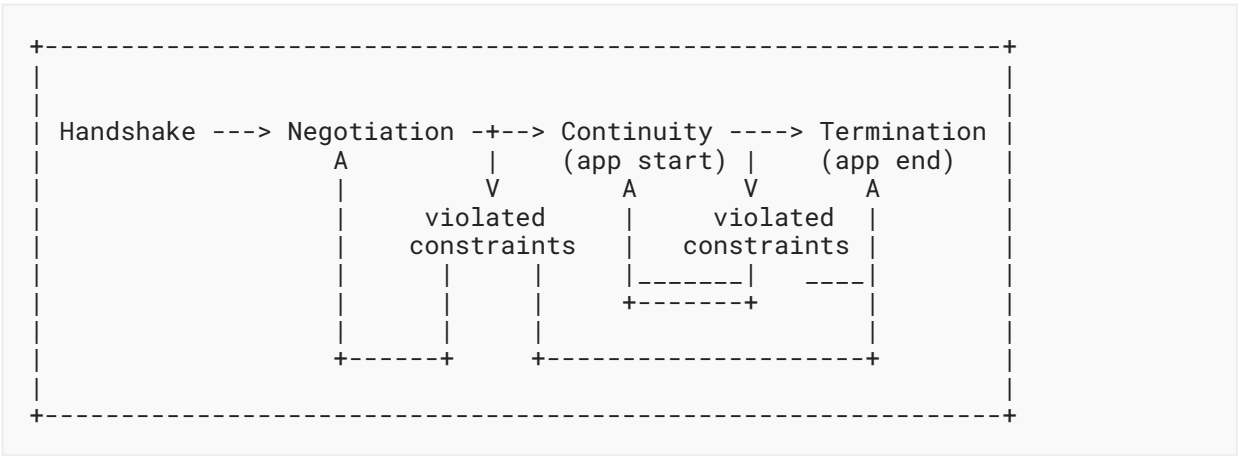


Figure 5: Session Lifetime Phases

Handshake phase: in which the server is contacted by the client, and in the answer message, the quality constraints for the application are communicated in the embedded SDP.

Negotiation phase: in which the quality of the connection is measured in both directions (latency, jitter, bandwidth, and packet loss), and Q4S messages may be sent in order to alert if the measured quality does not meet the constraints. This phase is iterative until quality constraints are reached, or the session is canceled after a number of measurement cycles with consistent violation of the quality constraints. The number of measurement cycles executed depends on the "qos-level", which is incremented in each cycle until a maximum "qos-level" value is reached. Just after reaching the quality requirements, Q4S provides a simple optional mechanism using HTTP to start the application.

Continuity phase: in which quality is continuously measured. In this phase, the measurements **MUST** avoid disturbing the application by consuming network resources. If quality constraints are not met, the server stack will notify the Actuator with an alert notification. If later the quality improves, the server stack will notify the Actuator, in this case with a recovery notification. After several alert notifications with no quality improvements, the Q4S stack **SHOULD** move to the Termination phase.

Termination phase: in which the Q4S session is terminated. The application may be closed also or may not start.

7.2. SDP Structure

The original goal of SDP was to announce necessary information for the participants and multicast MBONE (Multicast Backbone) applications. Right now, its use has been extended to the announcement and the negotiation of multimedia sessions. The purpose of Q4S is not to establish media stream sessions, but to monitor a quality connection. This connection may be later used to establish any type of session including media sessions; Q4S does not impose any conditions on the type of communication requiring quality parameters.

SDP will be used by Q4S to exchange quality constraints and will therefore always have all the media descriptions ("m=") set to zero.

The SDP embedded in the messages is the container of the quality parameters. As these may vary depending on the direction of the communication (to and from the client), all quality parameters need to specify the uplink and downlink values: <uplink> / <downlink> (see Section 7.5.3 for an example). When one or both of these values are empty, it **MUST** be understood as needing no constraint on that parameter and/or that direction.

The uplink direction **MUST** be considered as being the communication from the client to the server. The downlink direction **MUST** be considered as being the communication from the server to the client.

The SDP information can comprise all or some of the following parameters shown in the example below. This is an example of an SDP message used by Q4S included in the 200 OK response to a Q4S BEGIN request.

```
v=0
o=q4s-UA 53655765 2353687637 IN IP4 192.0.2.33
s=Q4S
i=Q4S parameters
t=0 0
a=qos-level:0/0
a=alerting-mode:Reactive
a=alert-pause:5000
a=public-address:client IP4 198.51.100.51
a=public-address:server IP4 198.51.100.58
a=measurement:procedure default(50/50,75/75,5000,40/80,100/256)
a=latency:40
a=jitter:10/10
a=bandwidth:20/6000
a=packetloss:0.50/0.50
a=flow:app clientListeningPort TCP/10000-20000
a=flow:app clientListeningPort UDP/15000-18000
a=flow:app serverListeningPort TCP/56000
a=flow:app serverListeningPort UDP/56000
a=flow:q4s clientListeningPort UDP/55000
a=flow:q4s clientListeningPort TCP/55001
a=flow:q4s serverListeningPort UDP/56000
a=flow:q4s serverListeningPort TCP/56001
```

As quality constraints may be changed by applications at any time during the Q4S session lifetime, any Q4S 200 OK response sent by the server to the client in the Negotiation and Continuity phases could also include an SDP body with the new quality requirements stated by the applications from then on. Therefore, in response to any PING request sent by the client to the server, the server could send a Q4S 200 OK with an embedded SDP message that specifies new quality constraints requested by the application.

7.2.1. "qos-level" Attribute

The "qos-level" attribute contains the QoS level for uplink and downlink. Default values are 0 for both directions. The meaning of each level is out of scope of Q4S, but a higher level **SHOULD** correspond to a better service quality.

Appropriate attribute values: [0..9] "/" [0..9]

The "qos-level" attribute may be changed during the session lifetime, raising or lowering the value as necessary following the network measurements and the application needs.

7.2.2. "alerting-mode" Attribute

The "alerting-mode" attribute specifies the player in charge of triggering Q4S alerts in the case of constraint violation. There are two possible values:

Appropriate attribute values: <"Q4S-aware-network"|"Reactive">

Q4S-aware-network: Q4S-ALERT messages are triggered by the server to the client. In this case, the network is supposed to be Q4S aware, and reacts by itself to these alerts.

Reactive: alert notifications are sent by the server stack to the Actuator. In this case, the network is not Q4S aware, and a specific node (Actuator) is in charge of triggering tuning mechanisms, either on the network or in the application.

The "alerting-mode" attribute is optional, and if not present, Reactive alerting mode is assumed.

7.2.3. "alert-pause" Attribute

In the Q4S-aware-network scenario, the "alert-pause" attribute specifies the amount of time (in milliseconds) the server waits between consecutive Q4S-ALERT messages sent to the client. In the Reactive scenario, the "alert-pause" attribute specifies the amount of time (in milliseconds) the server stack waits between consecutive alert notifications sent to the Actuator. Measurements are not stopped in Negotiation or Continuity phases during this period of time, but no Q4S-ALERT messages or alert notifications are fired, even with violated quality constraints, allowing for either network reconfigurations or application adjustments.

Appropriate attribute values: [0..60000]

7.2.4. "recovery-pause" Attribute

In the Q4S-aware-network scenario, the "recovery-pause" attribute specifies the amount of time (in milliseconds) the server waits for initiating the "qos-level" recovery process. Once the recovery process has started, the "recovery-pause" attribute also states the amount of time (in milliseconds) between consecutive Q4S-RECOVERY messages sent by the server to the client (in the Q4S-aware-network scenario) or between recovery notifications sent by the server stack to the Actuator (in the Reactive scenario).

Appropriate attribute values: [0..60000]

7.2.5. "public-address" Attributes

This attribute contains the public IP address of the client and the server. The server fills these attributes with its own public IP address and the public IP address of the first message received from the client in the Handshake phase.

The purpose of these attributes is to make available the addressing information to the network policy server or other external entities in charge of processing Q4S-ALERT messages.

Appropriate attribute values: <"client"|"server"> <"IP4"|"IP6"> <value of IP address>

7.2.6. "latency" Attribute

The maximum latency (considered equal for uplink and downlink) tolerance is specified in the "latency" attribute, expressed in milliseconds. In the Q4S-aware-network scenario, if the latency constraints are not met, a Q4S-ALERT method will be sent to the client. In the Reactive scenario, if the latency constraints are not met, an alert notification will be sent to the Actuator. If the "latency" attribute is not present or has a 0 value, no latency constraints need to be met, and no measurements **MAY** be taken.

Appropriate attribute values: [0..9999]

7.2.7. "jitter" Attribute

The maximum uplink and downlink jitter tolerance is specified in the "jitter" attribute, expressed in milliseconds. In the Q4S-aware-network scenario, if the jitter constraints are not met, a Q4S-ALERT method will be sent to the client. In the Reactive scenario, if the latency constraints are not met, an alert notification will be sent to the Actuator. If the "jitter" attribute is not present or has a 0 value, no jitter constraints need to be met, and no measurements **MAY** be taken.

Appropriate attribute values: [0..9999] "/" [0..9999]

7.2.8. "bandwidth" Attribute

The minimum uplink and downlink bandwidth is specified in the "bandwidth" attribute, expressed in kbps. In the Q4S-aware-network scenario, if the bandwidth constraints are not met, a Q4S-ALERT method will be sent to the client. In the Reactive scenario, an alert notification will be sent to the Actuator. If the "bandwidth" attribute is not present or has a 0 value, no bandwidth constraints need to be met, and no measurements **MAY** be taken.

Appropriate attribute values: [0..99999] "/" [0..99999]

7.2.9. "packetloss" Attribute

The maximum uplink and downlink packet loss tolerance is specified in the "packetloss" attribute expressed in percentage (two decimal accuracy). In the Q4S-aware-network scenario, if the packetloss constraints are not met, a Q4S-ALERT method will be sent to the client. In the Reactive scenario, an alert notification will be sent to the Actuator. If the "packetloss" attribute is not present or has a 0 value, no packet loss constraints need to be met, and no measurements **MAY** be taken.

Appropriate attribute values: [0.00 ..100.00] "/"[0.00 ..100.00]

7.2.10. "flow" Attributes

These attributes specify the flows (protocol, destination IP/ports) of data over TCP and UDP ports to be used in uplink and downlink communications.

Several "flow" attributes can be defined. These flows identify the listening port (client or server), the protocol (TCP [RFC0793] or UDP [RFC0768]) with the range of ports that are going to be used by the application and, of course, by the Q4S protocol (for quality measurements). All defined flows ("app" and "q4s") will be considered within the same quality profile, which is determined by the "qos-level" attribute in each direction. This allows us to assume that measurements on "q4s" flows are the same as experienced by the application, which is using "app" flows.

During Negotiation and Continuity phases, the specified Q4S ports in the "flow:q4s" attributes of SDP will be used for Q4S messages.

The Q4S flows comprise two UDP flows and two TCP flows (one uplink and one downlink for each one), whereas application traffic **MAY** consist of many flows, depending on its nature. The Handshake phase takes place through the Q4S Contact URI, using the standard Q4S TCP port. However, the Negotiation and Continuity phases will take place on the Q4S ports (UDP and TCP) specified in the SDP.

The "clientListeningPort" is a port on which the client listens for server requests and **MUST** be used as the origin port of client responses. The "serverListeningPort" is a port on which the server is listening for incoming messages from the client. The origin port of server responses may be different than the "serverListeningPort" value.

If "clientListeningPort" is zero ("a=flow:q4s clientListeningPort TCP/0"), the client **MAY** choose one randomly per OS standard rules. Client ports inside the SDP must always be matched against actual received port values on the server side in order to deal with NAT/NAPT devices. If a zero value or incorrect value is present, the server must set the value to the received origin port in the next message with SDP (200 OK, ALERT, and CANCEL messages).

```
Attribute values:
   <"q4s"|"app"> <"serverListeningPort"|"clientListeningPort">
<"UDP"|"TCP"> <0..65535> [ "-" [0..65535]]
```

7.2.11. "measurement" Attributes

These attributes contain the measurement procedure and the results of the quality measurements.

Measurement parameters are included using the session attribute "measurement". The first measurement parameter is the procedure. Q4S provides a "default" procedure for measurements, but others like RTP/RTCP might be used and defined later. This document will only define and explain the "default" procedure.

In the initial client request, a set of measurement procedures can be sent to the server for negotiation. One measurement procedure line **MUST** be included in the SDP message for each proposed method. The server **MUST** answer with only one line with the chosen procedure.

For each procedure, a set of values of parameters separated by "," can be included in the same attribute line. The amount and type of parameters depends on the procedure type.

In the following example, the "default" procedure type is chosen:

```
   a=measurement:procedure default(50/50,75/75,5000,40/80,100/256)
```

In the "default" procedure, the meaning of these parameters is the following:

- The first parameter is the interval of time (in milliseconds) between PING requests during the Negotiation phase. Uplink and downlink values from the client's point of view are

separated by "/". This allows different responsiveness values depending on the control resources used in each direction.

- The second parameter is the time interval (in milliseconds) between PING requests during the Continuity phase. Uplink and downlink values are separated by "/". This allows two different responsiveness values depending on the control resources used in each direction.
- The third parameter is the time interval to be used to measure bandwidth during the Negotiation phase.
- The fourth parameter indicates the window size for jitter and latency calculations. Uplink and downlink values are separated by "/".
- The fifth parameter indicates the window size for packet loss calculations. Uplink and downlink values are separated by "/".

There are four more "measurement" attributes:

```
a=measurement:latency 45
a=measurement:jitter 3/12
a=measurement:bandwidth 200/9800
a=measurement:packetloss 0.00/1.00
```

The "measurement:latency", "measurement:jitter", "measurement:bandwidth", and "measurement:packetloss" attributes contain the values measured for each of these quality parameters in uplink and downlink directions. Notice that latency is considered equal for uplink and downlink directions. Quality parameter values in these "measurement" attributes provide a snapshot of the quality reached and **MUST** only be included in Q4S-ALERT messages in the SDP body such that they can be protected from malicious attacks as these alerts include a signature of the SDP body in the header. The rest of the messages will include the measured values in the Measurements header field.

In the case of the "default" procedure, the valid values are as follows:

```
a=measurement:procedure default,[0..999]"/" [0..999]  "," [0..999]
"/" [0..999] "," [0..9999] "," [0..999]/[0..999] ","
[0..999]/[0..999]
```

7.2.12. "max-content-length" Attribute

The adaptation of measurement traffic to approximate the actual data streams' characteristics is convenient to accurately estimate the expected QoS for applications. Particularly, packet size can have a remarkable effect on bandwidth estimations. Moreover, this can produce problems depending on the MTU of the end hosts and links along the path.

Therefore, the maximum content length **MAY** be set in an attribute denoted as "max-content-length". Its value **MUST** be given in bytes and **MUST NOT** include application, transport, network, or link layer headers, i.e., size of the content length at the application layer. If not set, the value **MUST** be 1000 bytes.

Furthermore, this attribute **MAY** be used to communicate MTU limits in endpoints, hence reducing possible bias as a result of network-layer fragmentation.

For instance:

a=max-content-length:1300

7.3. Measurements

This section describes the way quality parameters are measured as defined by the "default" procedure. Measurements **MUST** be taken for any quality parameter with constraints, that is, specified in the SDP attributes with non-zero values. For absent attributes, measurements **MAY** be omitted.

7.3.1. Latency

Latency measurements will be performed if the "latency" attribute and/or the "a=measurement:latency" attribute are present and have non-zero values.

Q4S defines a PING method in order to exchange packets between the client and the server. Based on this PING exchange, the client and the server are able to calculate the round-trip time (RTT). The RTT is the sum of downlink latency (normally named "reverse latency") and uplink latency (normally named "forward latency").

At least 255 samples of RTT **MUST** be taken by the client and server. As the forward and reverse latencies are impossible to measure, the client and server will assume that both latencies are identical (symmetric network assumption). The latency will therefore be calculated as the statistical median value of all the RTT samples divided by 2.

7.3.2. Jitter

Jitter measurements will be performed if the "jitter" attribute and/or the "a=measurement:jitter" attribute are present and have non-zero values.

The jitter can be calculated independently by the client and by the server. The downlink jitter is calculated by the client taking into account the time interval between PING requests as defined by the "measurement:procedure" attribute in the first or second parameter depending on the Q4S protocol phase. The client and the server **MUST** send these PING requests at the specified intervals. The client measures the downlink jitter, whereas the server measures the uplink jitter. Note that PING responses are not taken into account when calculating jitter values.

Every time a PING request is received by an endpoint (either server or client), the corresponding jitter value is updated with the statistical jitter value, which is the arithmetic mean of the absolute values of elapsed times calculated on the first 255 packets received.

Each endpoint sends a PING periodically with a fixed interval, and each value of "elapsed time" (ET) should be very close to this interval. If a PING message is lost, the ET value is doubled. Identifying lost PING messages, however, is not an issue because all PING messages are labeled with a Sequence-Number header field. Therefore, the receiver can discard this ET value.

In order to have the first jitter sample, the receiver **MUST** wait until it receives 3 PING requests, because each ET is the time between two PINGs, and a jitter measurement needs at least two ET.

The client measures the values of RTT and downlink jitter, and the server measures RTT and uplink jitter, but all measurements are shared with the counterpart by means of the Measurements header field of the PING message.

7.3.3. Bandwidth

Bandwidth measurements will be performed if the "bandwidth" attribute and/or the "a=measurement:bandwidth" attribute is present and has non-zero values.

In order to measure the available bandwidth, both the client and the server **MUST** start sending BWIDTH messages simultaneously using the UDP control ports exchanged during the Handshake phase in the SDP message at the needed rate to verify the availability of the bandwidth constraint in each direction. The messages are sent during the period of time defined in the third parameter of the SDP "measurement:procedure default" attribute in milliseconds.

```
a=measurement:procedure default(50/50,75/75,5000,256/256,256/256)
```

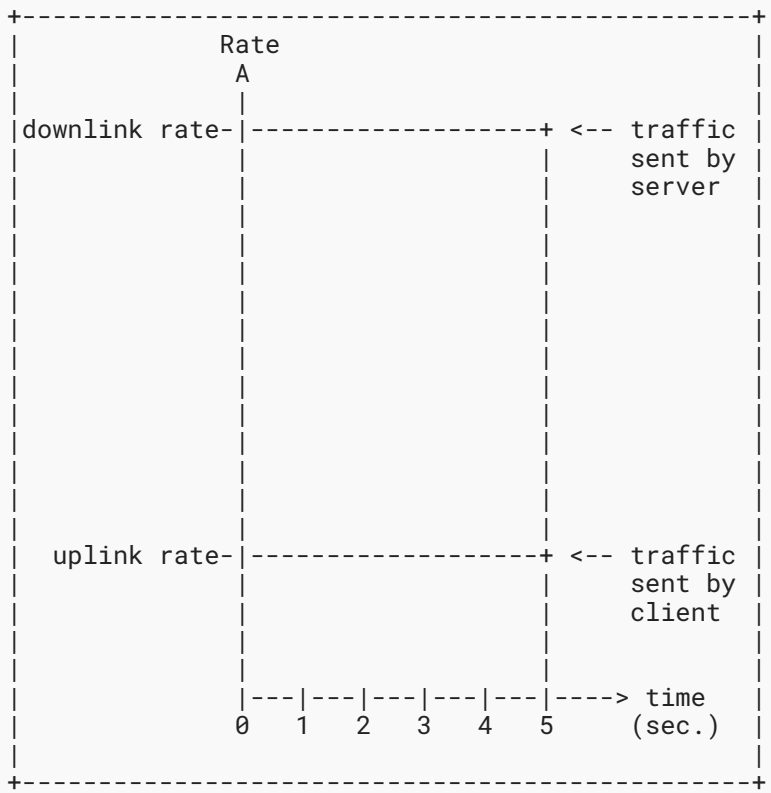


Figure 6: Bandwidth and Packet Loss Measurements

The goal of these measurements is not to identify the available bandwidth of the communication path, but to determine if the required bandwidth is available, meeting the application's constraints. Therefore, the requested bandwidth **MUST** be measured sending only the highest bitrate required by the bandwidth attribute. This is illustrated in Figure 6.

ALERTS are not expected during bandwidth measurement, but only at the end of the measurement time.

When measuring bandwidth, all BWIDTH requests sent **MUST** be 1 kilobyte in length (UDP payload length by default), they **MUST** include a Sequence-Number header field with a sequential number starting at 0, and their content **MUST** consist of randomly generated values to minimize the effect of compression elements along the path. The Sequence-Number **MUST** be incremented by 1 with each BWIDTH packet sent. If any measurement stage needs to be repeated, the sequence number **MUST** start at zero again. BWIDTH requests **MUST NOT** be answered. Examples:

```
Client message:
========================
       BWIDTH q4s://www.example.com Q4S/1.0
       User-Agent: q4s-ua-experimental-1.0
       Session-Id: 53655765
       Sequence-Number: 0
       Content-Type: text
       Content-Length: XXXX
       Measurements: l=22, j=10, pl=0.00, bw=3000

       VkZaU1FrNVZNVlZSV0doT1ZrZ (to complete up to "max-content-
                                  length" bytes UDP payload length)
========================
```

The client **MUST** send BWIDTH packets to the server to allow the server to measure the uplink bandwidth. The server **MUST** send BWIDTH packets to the client to allow the client to measure the downlink bandwidth.

```
Server message:
========================
       BWIDTH q4s://www.example.com Q4S/1.0
       Session-Id: 53655765
       Sequence-Number: 0
       Content-Type: text
       Content-Length: XXXX
       Measurements: l=22, j=7, pl=0.00, bw=200

       ZY0VaT1ZURlZVVmhyUFE9PQ (to complete up to max-content-
                               length UDP payload length)
========================
```

7.3.4. Packet Loss

Packet loss and bandwidth are measured simultaneously using the BWIDTH packets sent by both the client and the server. Because the BWIDTH packets contain a Sequence-Number header field incremented sequentially with each sent packet, lost packets can be easily identified. The lost packets **MUST** be counted during the measurement time.

7.4. Handshake Phase

The first phase consists of a Q4S BEGIN method issued from the client to the server as shown in Figure 7.

The first Q4S message **MUST** have a special URI [RFC3986], which forces the use of the Q4S protocol if it is implemented in a standard web browser.

This URI, named "Contact URI", is used to request the start of a session. Its scheme **MUST** be:

```
"q4s:" "//" host [":" port] [path["?" query]
```

Optionally, the client can send the desired quality parameters enclosed in the body of the message as an SDP document. The server **MAY** take them into account when building the answer message with the final values in the SDP body, following a request/response schema [RFC3264].

If the request is accepted, the server **MUST** answer it with a Q4S 200 OK message, which **MUST** contain an SDP body [RFC4566] with the assigned sess-id (embedded in the SDP "o=" line), the IP addresses to be used, the flow ports to be used, the measurement procedure to be followed, and information about the required quality constraints. Additionally, the "alerting-mode" and "alert-pause" time attributes may be included. Q4S responses should use the protocol designator "Q4S/1.0".

After these two messages are exchanged, the first phase is completed. The quality parameter thresholds have been sent to the client. The next step is to measure the actual quality of the communication path between the client and the server and alert if the Service Level Agreement (SLA) is being violated.

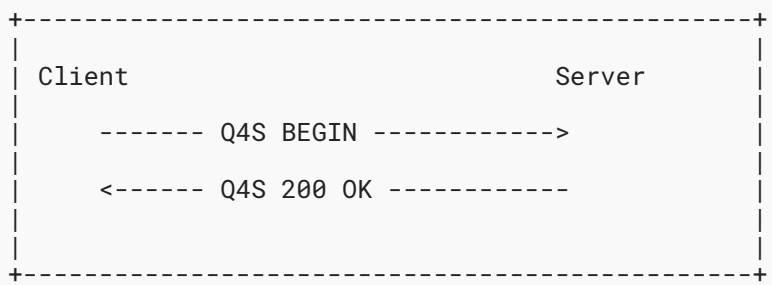


Figure 7: Handshake Phase

The following is an example of a client request and a server answer:

```
Client Request:
=========================
BEGIN q4s://www.example.com Q4S/1.0
Content-Type: application/sdp
User-Agent: q4s-ua-experimental-1.0
Content-Length: 142

(SDP not shown)
=========================

Server Answer:
=========================
Q4S/1.0 200 OK
Date: Mon, 10 Jun 2010 10:00:01 GMT
Content-Type: application/sdp
Expires: 3000
Signature: 6ec1ba40e2adf2d783de530ae254acd4f3477ac4
Content-Length: 131

(SDP not shown)
=========================
```

The header fields used are explained in Section 4.3.

7.5. Negotiation Phase

The Negotiation phase is in charge of measuring the quality parameters and verifying that the communication paths meet the required quality constraints in both directions as specified in the SDP body.

The measured parameters will be compared with the quality constraints specified in the SDP body. If the quality session is compliant with all the quality constraints, the application can start.

If the quality constraints are not met, a higher quality service level will be demanded. Depending on the scenario, this quality upgrade will be managed as follows:

In the Q4S-aware-network scenario: a Q4S-ALERT method will be triggered by the server to the client, and the client will answer with the same Q4S-ALERT method. After receiving the same Q4S-ALERT from the counterpart, no other alerts will be triggered by the server during the "alert-pause" period of time in order to allow the network to react, but measurements will continue to be taken to achieve early detection of improved network quality conditions and a fast application start.

In the Reactive scenario: an alert notification will be sent by the server stack to the Actuator, and the Actuator will answer with an alert acknowledgement. After receiving the alert acknowledgement from the Actuator, the server stack will not send other alert notifications during the "alert-pause" period of time in order to allow the Actuator to react and trigger actions on the application or on the policy server, but measurements will continue to be taken to achieve early detection of improved network quality conditions and a fast application start.

In both scenarios stated above, if after several measurement cycles, the network constraints cannot be met, the quality session is terminated. Concretely when, under all possible actions taken by Actuator, the quality remains below requirements, the session must be terminated.

The steps to be taken in this phase depend on the measurement procedure exchanged during the Handshake phase. This document only describes the "default" procedure, but others can be used, like RTP/RTCP [RFC3550].

Measurements of latency and jitter are made by calculating the differences in the arrival times of packets and can be achieved with little bandwidth consumption. The bandwidth measurement, on the other hand, involves higher bandwidth consumption in both directions (uplink and downlink).

To avoid wasting unnecessary network resources, these two types of measurements will be performed in two separate stages. If the required latencies and jitters cannot be reached, it makes no sense to waste network resources measuring bandwidth. In addition, if achieving the required latency and jitter thresholds implies upgrading the quality session level, the chance of obtaining compliant bandwidth measurements without retries is higher, saving network traffic again. Therefore, the "default" procedure determines that the measurements are taken in two stages:

Stage 0: Measurement of latencies, jitters, and packet loss

Stage 1: Measurement of bandwidths and packet loss

Notice that packet loss can be measured in both stages, as all messages exchanged include a Sequence-Number header field that allows for easy packet loss detection.

The client starts the Negotiation phase by sending a READY request using the TCP Q4S ports defined in the SDP. This READY request includes a Stage header field that indicates the measurement stage.

If either jitter, latency, or both are specified, the Negotiation phase begins with the measurement of latencies and jitters (stage 0). If none of those attributes is specified, stage 0 is skipped.

7.5.1. Stage 0: Measurement of Latencies and Jitter

The Stage 0 **MUST** start with a synchronization message exchange initiated with the client's READY message.

```
Client Request, READY message:
==========================
       READY q4s://www.example.com Q4S/1.0
       Stage: 0
       Session-Id: 53655765
       User-Agent: q4s-ua-experimental-1.0
       Content-Length: 0
==========================

Server Response:
========================
  Q4S/1.0 200 OK
       Session-Id: 53655765
       Stage:0
       Content-Length: 0
==========================
```

This triggers the exchange of a sequence of PING requests and responses that will lead to the calculation of RTT (latency), jitter, and packet loss.

After receiving a 200 OK, the client must send the first PING message, and the server will wait to send PINGs until the reception of this first client PING.

The client and server **MUST** send PING requests to each other. The Sequence-Number header field of the first PING **MUST** be set to 0. The client and server will manage their own sequence numbers.

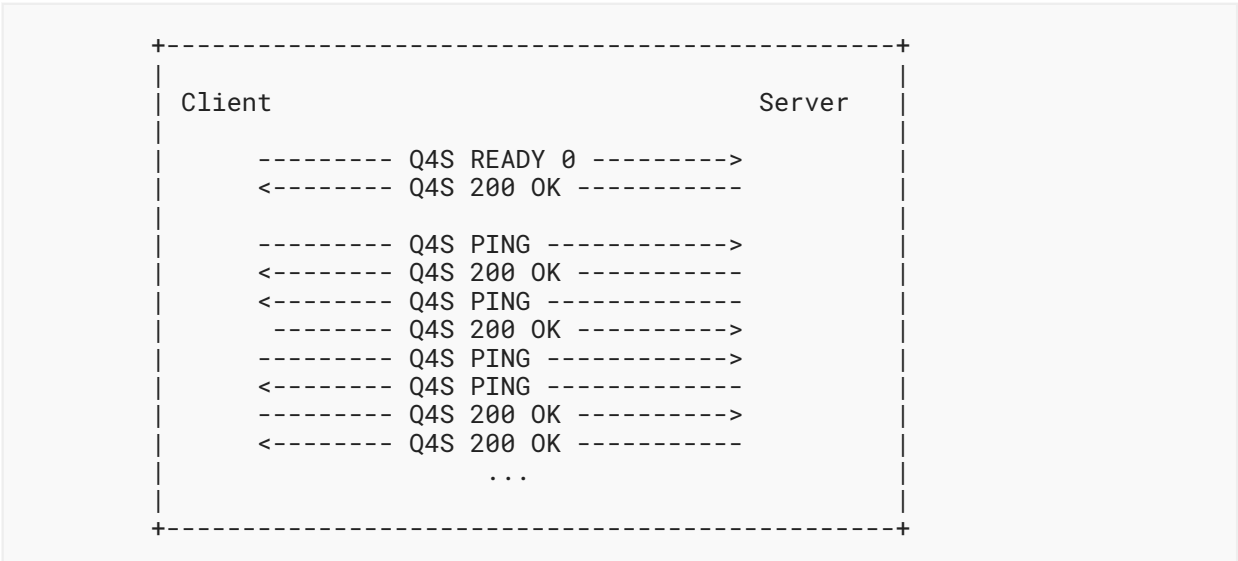


Figure 8: Simultaneous Exchange of PING Request and Responses

The following is an example of the PING request sent from the client and the server's response:

```
Client Request:
========================
       PING q4s://www.example.com Q4S/1.0
       Session-Id: 53655765
       Sequence-Number: 0
       User-Agent: q4s-ua-experimental-1.0
       Measurements: l=22, j=12, pl=0.20, bw=
       Content-Length: 0
========================

Server Response:
========================
  Q4S/1.0 200 OK
       Session-Id: 53655765
       Sequence-Number: 0
       Content-Length: 0
========================
```

The function of the PING method is similar to the ICMP echo request message [RFC0792]. The server **MUST** answer as soon as it receives the message.

Both endpoints **MUST** send Q4S PING messages with the periodicity specified in the first parameter of SDP "measurement:procedure" attribute, always using the same UDP ports and incrementing the Sequence-Number with each message.

In the following example, the value of the first parameter of the SDP "measurement:procedure" attribute is 50 milliseconds (from the client to the server) and 60 ms (from the server to the client):

```
a=measurement:procedure default(50/60,50/50,5000,256/256,256/256)
```

They **MUST NOT** wait for a response to send the next PING request. The Sequence-Number header field value is incremented sequentially and **MUST** start at zero. If this stage is repeated, the initial Sequence-Number **MUST** start at zero again.

All PING requests **MUST** contain a Measurements header field with the values of the latency, jitter, and packet loss measured by each entity up to that moment. The client will send its measurements to the server, and the server will send its measurements to the client. Example:

```
       Measurements: l=22, j=13, pl=0.10, bw=
```

Where "l" stands for latency, "j" for jitter, "pl" for packet loss, and "bw" for bandwidth. The bandwidth value is omitted, as it is not measured at this stage.

Optionally the PING request can include a Timestamp header field with the time in which the message has been sent. In the case that the header field is present, the server **MUST** include the header field in the response without changing the value.

A minimum number of PING messages **MUST** be exchanged in order to be able to measure latency, jitter, and packet loss with certain accuracy (at least 256 samples are **RECOMMENDED** to get an accurate packet loss measurement). Both the client and the server calculate the respective measured parameter values. The mechanisms to calculate the different parameters are described in Section 7.3.

At the end of this stage 0, there are three possibilities:

- The latency, jitter, and packetloss constraints are reached in both directions
- The latency, jitter, and packetloss constraints are not reached in one or both directions

In the first case, Stage 0 is finished. The client and server are ready for Stage 1: bandwidth and packet loss measurement. The client moves to stage 1 by sending a READY message that includes the header field, "Stage: 1".

If the bandwidth constraints are either empty or have a value of zero, the Negotiation phase **MUST** terminate, and both client and server may initiate the Continuity phase. In this case, client moves to the Continuity phase by sending a READY message that includes the header field, "Stage: 2".

The second case, in which one or more quality constraints have not been met, is detailed in Section 7.5.4.

7.5.2. Stage 1: Measurement of Bandwidth and Packet Loss

This stage begins in a similar way to stage 0, sending a READY request over TCP. The value of the READY message's Stage header field is 1. The server answers with a Q4S 200 OK message to synchronize the initiation of the measurements as shown in Figure 9.

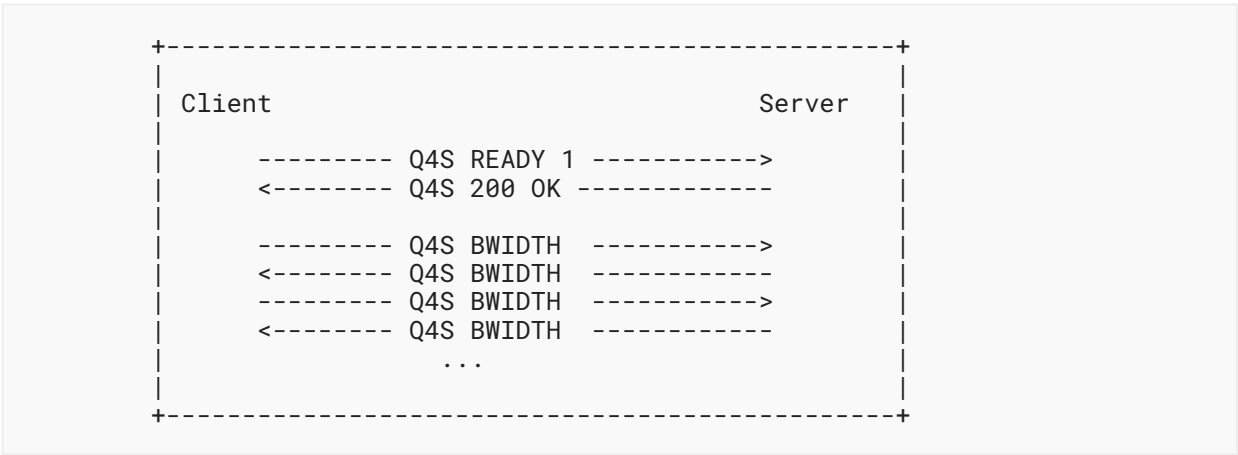


Figure 9: Starting Bandwidth and Packet Loss Measurement

```
Client Request:
========================
       READY q4s://www.example.com Q4S/1.0
       User-Agent: q4s-ua-experimental-1.0
       Stage: 1
       Session-Id: 53655765
       Content-Length: 0

========================

Server Response:
========================
  Q4S/1.0 200 OK
       Session-Id: 53655765
       Stage: 1
       Content-Length: 0

========================
```

Just after receiving the 200 OK, both the client and the server **MUST** start sending BWIDTH messages simultaneously using the UDP "q4s" ports. Section 7.3.3 describes the bandwidth measurement in detail.

At the end of this stage 1, there are three possibilities:

- The bandwidth and packetloss constraints are reached in both directions.
- The bandwidth and packetloss constraints are not reached in one or both directions.

In the first case, Stage 1 is finished. The client and server are ready for the Continuity phase. The client moves to this phase by sending a READY message that includes the header field, "Stage: 2". The server answer **MUST** be 200 OK as shown in Figure 10.

Figure 10: Trigger the Application Using HTTP URI

```
Client Request:
========================
READY q4s://www.example.com Q4S/1.0
User-Agent: q4s-ua-experimental-1.0
Stage: 2
Session-Id: 53655765
Content-Length: 0

=========================

Server Answer:
========================
Q4S/1.0 200 OK
Date: Mon, 10 Jun 2010 10:00:01 GMT
Session-Id: 53655765
Trigger-URI: http://www.example.com/app_start
Expires: 3000
Content-Type: application/sdp
Signature: 6ec1ba40e2adf2d783de530ae254acd4f3477ac4
Content-Length: 131

(SDP not shown)
=========================
```

If the Trigger-URI header field is present, the client **SHOULD** send an HTTP request to this URI.

The second case, with violated network constraints, is explained in Section 7.5.4.

7.5.3. Quality Constraints Not Reached

After finishing Stage 1 of the Negotiation phase, the client and the server have each other's measured parameter values as these have been exchanged in the Measurements header fields of the PING and BWIDTH messages. If there is one or more parameters that do not comply with the uplink or downlink application constraints required, both the server and the client are aware of it.

If there is any quality parameter that does not meet the uplink or downlink quality constraints specified in the SDP message, two scenarios are possible depending on the specified alerting mode (if not present, the default value is Reactive alerting mode):

(a) Q4S-aware-network alerting mode: the server **MUST** send a Q4S-ALERT message to the client including the digital Signature header field, and the client **MUST** answer with the same Q4S-ALERT message. The Signature header field contains the signed hash value of the SDP body in order to protect all the SDP data, and therefore it **MUST** contain the "measurement" parameters in the body.

```
Server request
=========================
Q4S-ALERT q4s://www.example.com Q4S/1.0
Host: www.example.com
User-Agent: q4s-ua-experimental-1.0
Session-Id: 53655765
Content-Type: application/sdp
Content-Length: 142

v=0
o=q4s-UA 53655765 2353687637 IN IP4 192.0.2.33
s=Q4S
i=Q4S parameters
t=0 0
a=qos-level:1/2
a=alerting-mode: Q4S-aware-network
a=alert-pause:5000
a=public-address:client IP4 198.51.100.51
a=public-address:server IP4 198.51.100.58
a=latency:40
a=jitter:10/10
a=bandwidth:20/6000
a=packetloss:0.50/0.50
a=flow:app downlink TCP/10000-20000
a=flow:app uplink TCP/56000
a=flow:q4s downlink UDP/55000
a=flow:q4s downlink TCP/55001
a=flow:q4s uplink UDP/56000
a=flow:q4s uplink TCP/56001
a=measurement:procedure default(50/50,50/50,5000,256/256,256/256)
a=measurement:latency 30
a=measurement:jitter 6/4
a=measurement:bandwidth 200/4000
a=measurement:packetloss 0.20/0.33
=========================
```

At this point, both the client and server keep on measuring but without sending new Q4S-ALERT messages during the "alert-pause" milliseconds.

(b) Reactive alerting mode: the server stack **MUST** send an alert notification to the Actuator, and the Actuator **MUST** answer with an acknowledgement to the received alert notification. The alert notification sent to the Actuator by the server stack doesn't follow Q4S message style but should have all the information the Actuator will need for the actions to be taken, which will be implementation dependent.

At this point during Negotiation phase, both the client and server keep on measuring without sending new alert notifications to the Actuator during the "alert-pause" milliseconds specified in the SDP. This way, both client and server will detect any improvement in network conditions as soon as the network reacts. The application can start as soon as the number of measurements indicated in the "measurement:procedure" attribute indicates that the quality parameters are met.

The same applies to Continuity phase: the measurement dialog between client and server must not be interrupted by any possible ALERT message.

7.5.3.1. Actuator Role

The actuator receives notifications of unmet requirements from the Q4S server stack and acts upon the application or the network policy server, according to logic out of scope of this protocol.

The Actuator logic activates mechanisms at the application level and/or the network level based on a quality level dictionary, in which the meaning of each level is implementation dependent, and each level involves different actions based on rules to keep a certain user experience quality.

The type of actions that an Actuator can take at the application level are application dependent and **MAY** involve:

- Reduction of application functionalities, such as limitation of application speed or application options.
- Reduction of application resources usage, such as reduction of frames per second in a video application or any other parameter modification in order to adapt to network conditions.

Apart from actions at the application level, the Actuator **MAY** act at the network level if a network policy server is available.

7.5.3.2. Policy Server Role

A network policy server may be part of the Reactive scenario, and it is in charge of managing network quality provision. A network policy server may implement all or some of these features (but implementation is not exclusive to):

- Server validation in terms of quality constraints
- Authentication (Signature validation) and security (blocking of malicious clients)
- Policy rules (the following rules are only examples):
  - Maximum quality level allowed for the ACP
  - Time bands allowed for providing quality sessions
  - Number of simultaneous quality sessions allowed
  - Maximum time used by allowed quality sessions
  - Etc.

If any of the policy rules fail, a Q4S-ALERT message **MUST** be answered by a 6xx error indicating the cause.

7.5.4. "qos-level" Changes

If any constraint was violated, the server **MAY** trigger a Q4S-ALERT asking for a higher "qos-level" attribute. The maximum "qos-level" allowed is 9 for both uplink and downlink.

If the "qos-level" has reached the maximum value for the downlink or uplink without matching the constraints, then a CANCEL request **MUST** be sent by the client using the TCP port determined in the Handshake phase in order to release the session. In reaction to the reception of the CANCEL request, the server **MUST** send a CANCEL request, too. If no CANCEL request is received, the expiration time cancels the session on the server side.

```
Client Request:
========================
CANCEL q4s://www.example.com Q4S/1.0
User-Agent: q4s-ua-experimental-1.0
Session-Id: 53655765
Content-Type: application/sdp
Content-Length: 142

(SDP not shown)
========================

Server Request in reaction to Client Request:
========================
CANCEL q4s://www.example.com Q4S/1.0
Session-Id: 53655765
Expires: 0
Content-Type: application/sdp
Signature: 6ec1ba40e2adf2d783de530ae254acd4f3477ac4
Content-Length: 131

(SDP not shown)
========================
```

7.6. Continuity Phase

During the Negotiation phase, latency, jitter, bandwidth, and packet loss have been measured. During the Continuity phase, bandwidth will not be measured again because bandwidth measurements may disturb application performance.

This phase is supposed to be executed at the same time as the real-time application is being used.

This document only covers the "default" procedure. The continuity operation with the "default" procedure is based on a sliding window of samples. The number of samples involved in the sliding window may be different for jitter and latency than for packet loss calculations according to the fifth and sixth parameters of the "measurement:procedure" attribute. In the example, shown in Figure 11, the jitter and latency sliding window comprises 40 samples, whereas the size of the packet loss sliding window is 100 samples:

```
a=measurement:procedure default(50/50,75/75,5000,40/40,100/100)
```

In addition, the sizes of these windows are configurable per direction: uplink and downlink values may differ.

PING requests are sent continuously (in both directions), and when the Sequence-Number header field reaches the maximum value, the client continues sending PING messages with the Sequence-Number header field starting again at zero. When the server PING Sequence-Number header field reaches the maximum value, it does the same, starting again from zero.

On the client side, the measured values of downlink jitter, downlink packet loss, and latency are calculated using the last samples, discarding older ones, in a sliding window schema.

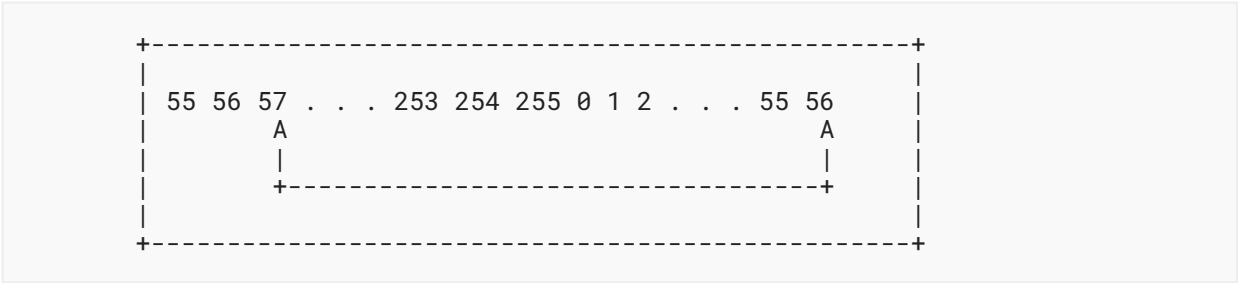


Figure 11: Sliding Samples Window

Only if the server detects that the measured values (downlink or uplink jitter, packet loss, or latency) are not reaching the quality constraints, a Q4S-ALERT is triggered and sent either to the client or to the Actuator, depending on the alerting mode, and the "alert-pause" timer is started.

In the Q4S-aware-network alerting mode shown in Figure 12, if the client receives a Q4S-ALERT message, it **MUST** answer by sending the Q4S-ALERT request message including the SDP (with its corresponding digital signature) back to the server.

Both client and server will keep performing measurements, but Q4S-ALERT messages **MUST NOT** be sent during "alert-pause" milliseconds. The operations needed to act on the network and the agents in charge of them are out of scope of this document.

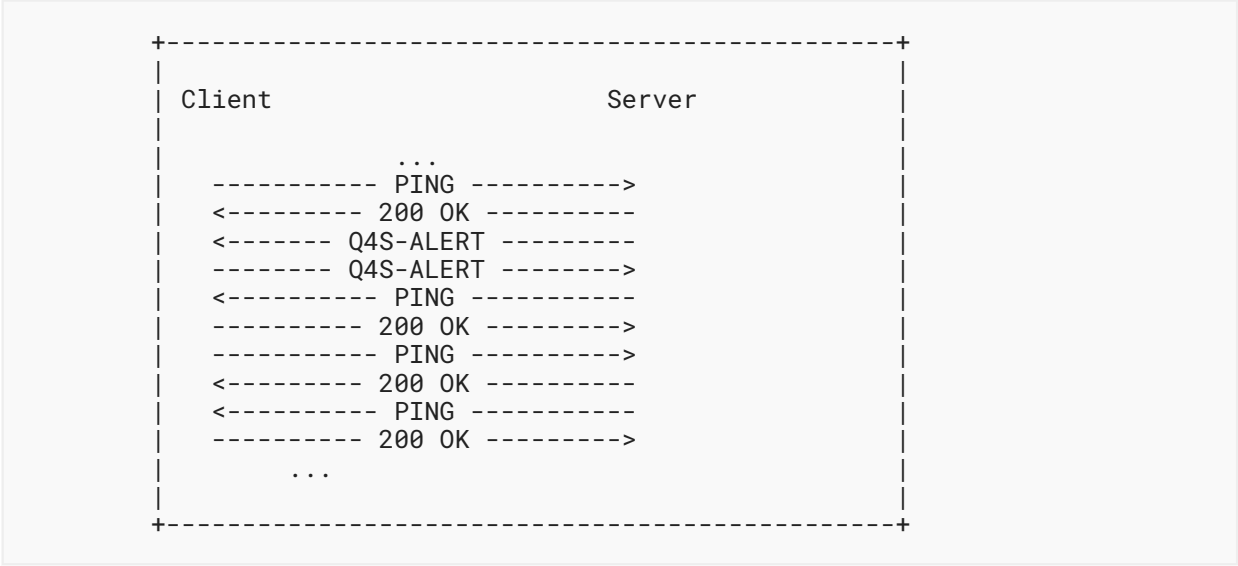


Figure 12: Continuity in Q4S-Aware-Network Alerting Mode

In the Reactive scenario shown in Figure 13, if the server detects that the measured values (downlink or uplink jitter, packet loss, or latency) are not reaching the quality constraints, an alert notification is triggered and sent to the Actuator. The Actuator **MUST** then answer to the server stack with an alert acknowledgement.

The measurement dialog between the client and the server **MUST NOT** be interrupted by any possible ALERT message.

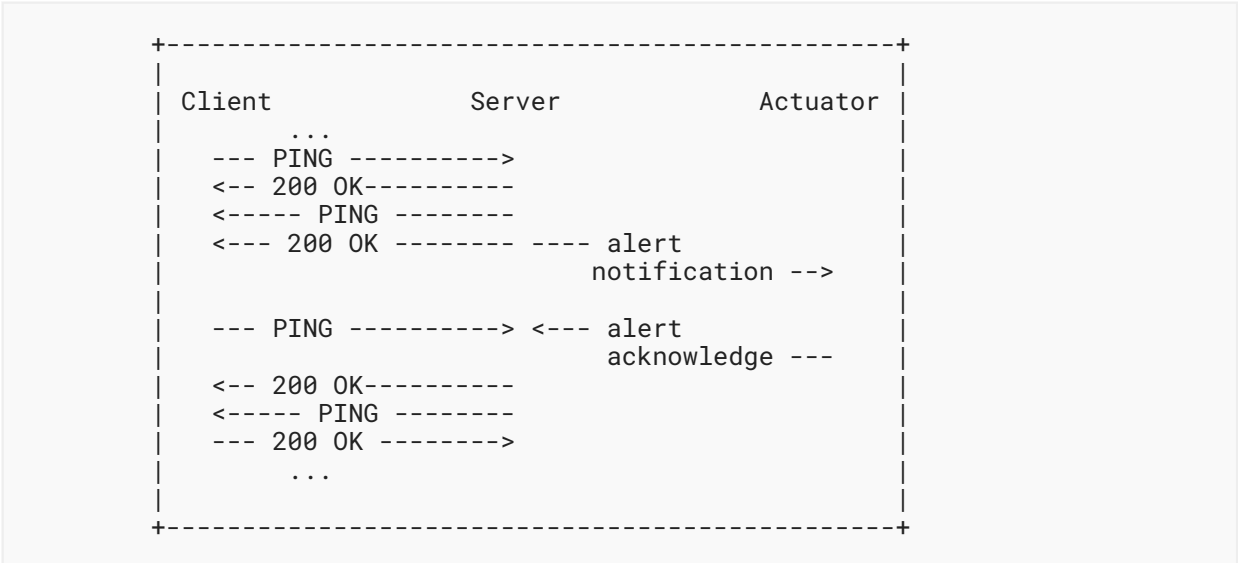


Figure 13: Continuity in Reactive Alerting Mode

7.7. Termination Phase

The Termination phase is the endpoint for the established Q4S session that is reached in the following cases:

- A CANCEL message has been received. The client sends a CANCEL message due to the network's inability to meet the required quality constraints. The client and server application will be notified by their respective Q4S stacks.
- Session expires: if after the Expires time, no client or server activity is detected, that end cancels the session.
- A BEGIN message has been received by the server. The pre-existing Q4S quality session is canceled, and a new session will be initiated.

The meaning of the Termination phase in terms of the release of resources or accounting is application dependent and out of scope of the Q4S protocol.

In the Reactive alerting mode, Q4S CANCEL messages received by the Q4S server must cause the server stack to send cancel notifications to the Actuator in order to release possible assigned resources for the session.

7.7.1. Sanity Check of Quality Sessions

A session may finish due to several reasons (client shutdown, client CANCEL request, constraints not reached, etc.), and any session finished **MUST** release the assigned resources.

In order to release the assigned server resources for the session, the Expires header field indicates the maximum interval of time without exchanging any Q4S message.

7.8. Dynamic Constraints and Flows

Depending on the nature of the application, the quality constraints to be reached may evolve, changing some or all quality constraint values in any direction.

The client **MUST** be able to deal with this possibility. When the server sends an SDP document attached to a response (200 OK or Q4S-ALERT, etc.), the client **MUST** take all the new received values, overriding any previous value in use.

The dynamic changes on the quality constraints can be a result of two possibilities:

- The application communicates to the Q4S server a change in the constraints. In this case, the application requirements can evolve, and the Q4S server will be aware of them.
- The application uses TCP flows. In that case, in order to guarantee a constant throughput, the nature of TCP behavior forces the use of a composite constraint function, which depends on RTT, packet loss, and a window control mechanism implemented in each TCP stack.

TCP throughput can be less than actual bandwidth if the Bandwidth-Delay Product (BDP) is large, or if the network suffers from a high packet loss rate. In both cases, TCP congestion control algorithms may result in a suboptimal performance.

Different TCP congestion control implementations like Reno [RENO], High Speed TCP [RFC3649], CUBIC [CUBIC], Compound TCP (CTCP) [CTCP], etc., reach different throughputs under the same network conditions of RTT and packet loss. In all cases, depending on the RTT-measured value, the Q4S server could dynamically change the packetloss constraints (defined in the SDP) in order to make it possible to reach a required throughput or vice versa (using "measurement:packetloss" to change dynamically the latency constraints).

A general guideline for calculating the packet loss constraint and the RTT constraint consists of approximating the throughput by using a simplified formula, which should take into account the TCP stack implementation of the receiver, in addition to the RTT and packet loss:

```
Th= Function( RTT, packet loss, ...)
```

Then, depending on RTT-measured values, set dynamically the packet loss constraint.

It is possible to easily calculate a worst-case boundary for the Reno algorithm, which should ensure for all algorithms that the target throughput is actually achieved, except that high-speed algorithms will then have even larger throughput if more bandwidth is available.

For the Reno algorithm, the Mathis formula may be used [RENO] for the upper bound on the throughput:

```
Th <= (MSS/RTT)*(1 / sqrt{p})
```

In the absence of packet loss, a practical limit for the TCP throughput is the receiver_window_size divided by the RTT. However, if the TCP implementation uses a window scale option, this limit can reach the available bandwidth value.

7.9. "qos-level" Upgrade and Downgrade Operation

Each time the server detects a violation of constraints, the alert mechanism is triggered, the "alert-pause" timer is started, and the "qos-level" is increased. When this happens repeatedly, and the "qos-level" reaches its maximum value (value 9), the session is canceled. But when the violation of constraints stops before reaching "qos-level" maximum value, the recovery mechanism allows for the "qos-level" upgrade gradually.

This downgrade and upgrade of "qos-level" is explained with the following example:

1. A Q4S session is initiated successfully with "qos-level=0".
2. During the Continuity phase, violation of constraints is detected; the "qos-level" is increased to 1, a Q4S-ALERT is sent by the server to the client, and an "alert-pause" timer is started.
3. The "alert-pause" timer expires, and still a violation of constraints is detected; the "qos-level" is increased to 2, a Q4S-ALERT is sent by the server to the client, and an "alert-pause" timer is started.

4. The "alert-pause" timer expires, but the violation of constraints has stopped; the "recovery-pause" timer is started.
5. The "recovery-pause" timer expires, and no violation of constraints has been detected. Meanwhile, the "qos-level" is decreased to 1, a Q4S-RECOVERY is sent by the server to the client, and the "recovery-pause" timer is started again.
6. The "recovery-pause" timer expires again, and no violation of constraints has been detected. Meanwhile, the "qos-level" is decreased to 0, and a Q4S-RECOVERY is sent by the server to the client. The "recovery-pause" timer is not started this time as the "qos-level" has reached its initial value.

When the network configuration allows for the possibility of managing Q4S flows and application flows independently (either is a network-based QoS or a Q4S-aware network), the "qos-level" downgrade process could be managed more efficiently using a strategy that allows for carrying out "qos-level" downgrades excluding application flows from SDP dynamically. The Q4S flows would be downgraded to allow for measurements on a lower quality level without interference of the application flows. A Q4S client **MUST** allow this kind of SDP modification by the server.

Periodically (every several minutes, depending on the implementation) a Q4S-ALERT could be triggered, in which the level is downgraded for Q4S flows, excluding application flows from the embedded SDP of that request.

This mechanism allows the measurement at lower levels of quality while application flows continue using a higher "qos-level" value.

- If the measurements in the lower level meet the quality constraints, then a Q4S-RECOVERY message to this lower "qos-level" may be triggered, in which the SDP includes the application flows in addition to the Q4S flows.
- If the measurements in the lower level do not meet the constraints, then a new Q4S-ALERT to the previous "qos-level" **MUST** be triggered, in which the SDP includes only the Q4S flows.

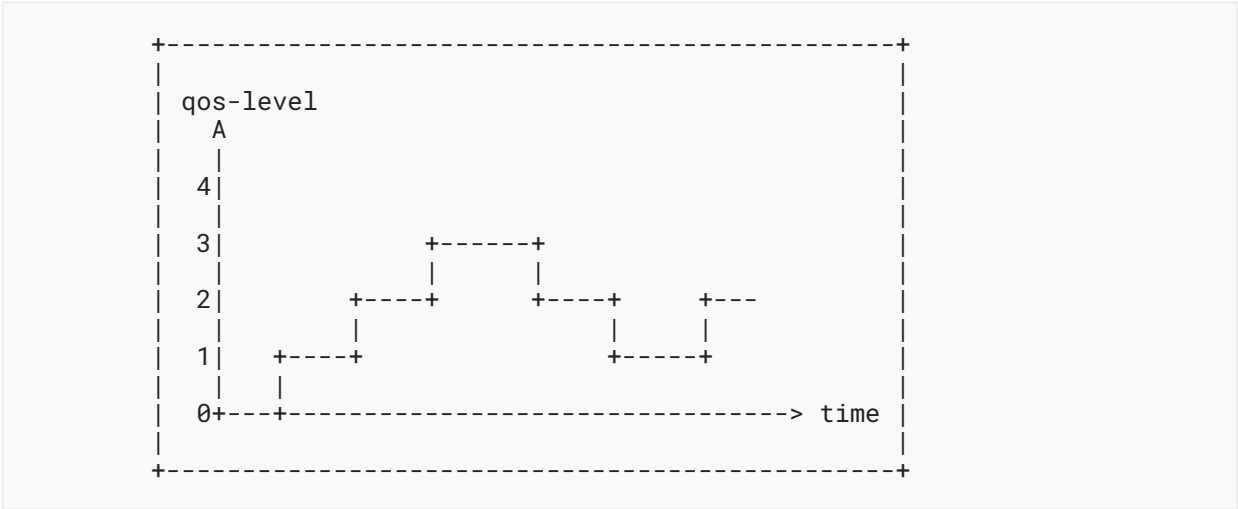


Figure 14: Possible Evolution of "qos-level"

This mechanism, illustrated in Figure 14, avoids the risk of disturbing the application while the measurements are being run in lower levels. However, this optional optimization of resources **MUST** be used carefully.

The chosen period to measure a lower "qos-level" is implementation dependent. Therefore, it is not included as a "measurement:procedure" parameter. It is **RECOMMENDED** to use a large value, such as 20 minutes.

8. General User Agent Behavior

8.1. Roles in Peer-to-Peer Scenarios

In order to allow peer-to-peer applications, a Q4S User Agent (UA) **MUST** be able to assume both the client and server role. The role assumed depends on who sends the first message.

In a communication between two UAs, the UA that first sends the Q4S BEGIN request to start the Handshake phase shall assume the client role.

If both UAs send the BEGIN request at the same time, they will wait for a random time to restart again as shown in Figure 15.

Otherwise, an UA may be configured to act only as server (e.g., content provider's side).

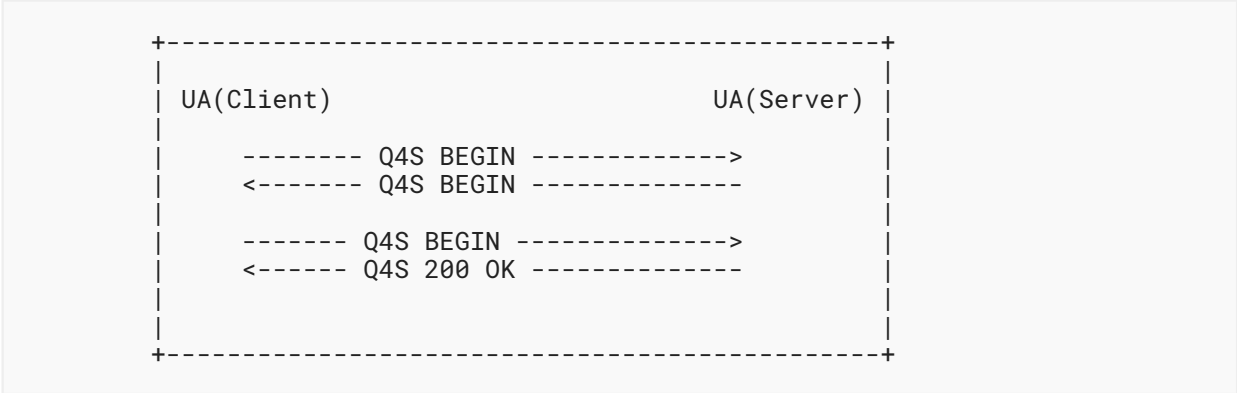


Figure 15: P2P Roles

8.2. Multiple Quality Sessions in Parallel

A Q4S session is intended to be used for an application. This means that for using the application, the client **MUST** establish only one Q4S session against the server. Indeed, the relation between the Session-Id and the application is 1 to 1.

If a user wants to participate in several independent Q4S sessions simultaneously against different servers (or against the same server), it can execute different Q4S clients to establish separately different Q4S sessions, but it is **NOT RECOMMENDED** because:

- The establishment of a new Q4S session may affect other running applications over other Q4S sessions during bandwidth measurement.
- If the Negotiation phase is executed separately before running any application, the summation of bandwidth requirements could not be met when the applications are running in parallel.

8.3. General Client Behavior

A Q4S client has different behaviors. We will use letters X, Y, and Z to designate each different behavior (follow the letters in Figure 16 and their descriptions below).

X) When it sends messages over TCP (methods BEGIN, READY, Q4S-ALERT, Q4S-RECOVERY, and CANCEL), it behaves strictly like a state machine that sends requests and waits for responses. Depending on the response type, it enters into a new state.

When it sends UDP messages (methods PING and BWIDTH), a Q4S client is not strictly a state machine that sends messages and waits for responses because of the following:

Y) During the measurement of latency, jitter, and packet loss, the PING requests are sent periodically, not just after receiving the response to the previous request. In addition, the client **MUST** answer the PING requests coming from the server, therefore the client assumes temporarily the role of a server.

Z) During the bandwidth and packet loss measurement stage, the client does not expect to receive responses when sending BWIDTH requests to the server. In addition, it **MUST** receive and process all server messages in order to achieve the downlink measurement.

The Q4S-ALERT and CANCEL may have a conventional answer if an error is produced, otherwise the corresponding answer is formatted as a request message.

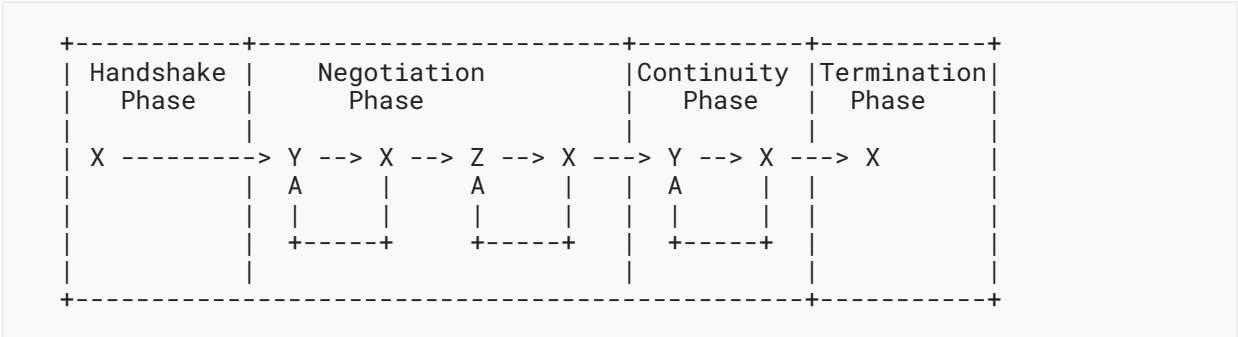


Figure 16: Phases and Client Behaviors

8.3.1. Generating Requests

A valid Q4S request formulated by a client **MUST**, at a minimum, contain the following header fields:

If no SDP is included: the header fields Session-Id and Sequence-Number are mandatory.

If SDP is included: the Session-Id is embedded into the SDP, therefore the inclusion of the Session-Id header field is optional, but if present, must have the same value. Measurements are embedded into the SDP only for Q4S-ALERT messages in order to be signed.

At any time, if the server sends new SDP with updated values, the client **MUST** take it into account.

8.4. General Server Behavior

If a server does not understand a header field in a request (that is, the header field is not defined in this specification or in any supported extension), the server **MUST** ignore that header field and continue processing the message.

The role of the server is changed at Negotiation and Continuity phases, in which the server **MUST** send packets to measure jitter, latency, and bandwidth. Therefore, the different behaviors of the server are (follow the letters in Figure 17 and their descriptions below):

R) When the client sends messages over TCP (methods BEGIN, READY Q4S-ALERT, Q4S-RECOVERY, and CANCEL), it behaves strictly like a state machine that receives messages and sends responses.

When the client begins to send UDP messages (methods PING and BWIDTH), a Q4S server is not strictly a state machine that receives messages and sends responses because of the following:

S) During the measurement of latency, jitter, and packet loss, the PING requests are sent periodically by the client and also by the server. In this case, the server behaves as a server answering client requests but also behaves temporarily as a client, sending PING requests toward the client and receiving responses.

T) During bandwidth and packet loss measurement, the server sends BWIDTH requests to the client. In addition, it **MUST** receive and process client messages in order to achieve the uplink measurement.

The Q4S-ALERT and CANCEL may have a conventional answer if an error is produced, otherwise the corresponding answer is formatted as a request message.

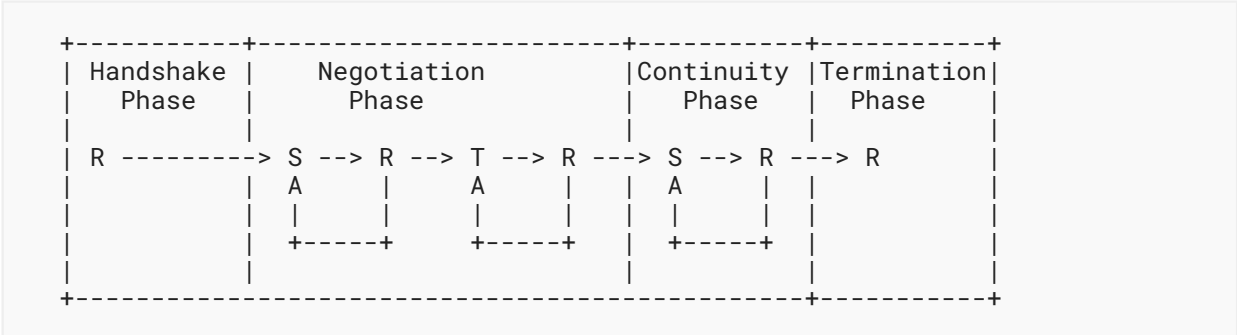


Figure 17: Phases and Server Behaviors

9. Implementation Recommendations

9.1. Default Client Constraints

To provide a default configuration, it would be good if the client had a configurable set of quality headers in the implementation settings menu. Otherwise, these quality headers will not be present in the first message.

Different business models (out of scope of this proposal) may be achieved: depending on who pays for the quality session, the server can accept certain client parameters sent in the first message, or force billing parameters on the server side.

9.2. Latency and Jitter Measurements

Different client and server implementations may send a different number of PING messages for measuring, although at least 255 messages should be considered to perform the latency measurement. The Stage 0 measurements may be considered ended only when neither the client nor server receive new PING messages after an implementation-dependent guard time. Only after, the client can send a "READY 1" message.

In execution systems, where the timers are not accurate, a recommended approach consists of including the optional Timestamp header field in the PING request with the time in which the message has been sent. This allows an accurate measurement of the jitter even with no identical intervals of time between PINGs.

9.3. Bandwidth Measurements

In programming languages or operating systems with limited timers or clock resolution, it is recommended to use an approach based on several intervals to send messages of 1KB (= 8000 bits) in order to reach the required bandwidth consumption, using a rate as close as possible to a constant rate.

For example, if the resolution is 1 millisecond, and the bandwidth to reach is 11 Mbps, a good approach consists of sending:

```
1 message of 1KB every 1 millisecond +

1 message of 1KB every 3 milliseconds +

1 message of 1KB every 23 milliseconds
```

The number of intervals depends on the required bandwidth and accuracy that the programmer wants to achieve.

Considering messages of 1KB (= 8000 bits), a general approach to determine these intervals is the following:

(1) Compute target bandwidth / 8000 bits. In the example above, it is 11 Mbps / 8000 = 1375 messages per second.

(2) Divide the number of messages per second by 1000 to determine the number of messages per millisecond: 1375 / 1000 = 1.375. The integer value is the number of messages per millisecond (in this case, one). The pending bandwidth is now 375 messages per second.

(3) To achieve the 375 messages per second, use a submultiple of 1000, which must be less than 375:

```
1000 / 2 = 500 > 375

1000 / 3 = 333 < 375
```

In this case, a message every 3 ms is suitable. The new pending target bandwidth is 375 - 333 = 42 messages per second.

(4) Repeat the same strategy as point 3 to reach the pending bandwidth. In this case, 23 ms is suitable because of the following:

```
1000 / 22 = 45 > 42

1000 / 23 = 43 > 42

1000 / 24 = 41.6 < 42
```

We can choose 24 ms, but then we need to cover an additional 0.4 messages per second (42 - 41.6 = 0.4), and 43 is a number higher than 42 but very close to it.

In execution systems where the timers are not accurate, a recommended approach consists of checking at each interval the number of packets that should have been sent at this timestamp since origin and send the needed number of packets in order to reach the required bandwidth.

The shorter the packets used, the more constant the rate of bandwidth measurement. However, this may stress the execution system in charge of receiving and processing packets. As a consequence, some packets may be lost because of stack overflows. To deal with this potential issue, a larger packet is **RECOMMENDED** (2KB or more), taking into account the overhead produced by the chunks' headers.

9.4. Packet Loss Measurement Resolution

Depending on the application nature and network conditions, a packet loss resolution less than 1% may be needed. In such cases, there is no limit to the number of samples used for this calculation. A trade-off between time and resolution should be reached in each case. For example, in order to have a resolution of 1/10000, the last 10000 samples should be considered in the packet loss measured value.

The problem of this approach is the reliability of old samples. If the interval used between PING messages is 50 ms, then to have a resolution of 1/1000, it takes 50 seconds, and a resolution of 1/10000 takes 500 seconds (more than 8 minutes). The reliability of a packet loss calculation based on a sliding window of 8 minutes depends on how fast network conditions evolve.

9.5. Measurements and Reactions

Q4S can be used as a mechanism to measure and trigger network tuning and application-level actions (i.e. lowering video bit-rate, reducing multiplayer interaction speed, etc.) in real time in order to reach the application constraints, addressing measured possible network degradation.

9.6. Instability Treatments

There are two scenarios in which Q4S can be affected by network problems: loss of Q4S packets and outlier samples.

9.6.1. Loss of Control Packets

Lost UDP packets (PING or BWIDTH messages) don't cause any problems for the Q4S state machine, but if TCP packets are delivered too late (which we will consider as "lost"), some undesirable consequences could arise.

Q4S does have protection mechanisms to overcome these situations. Examples:

- If a BEGIN packet or its corresponding answer is lost, after a certain timeout, the client **SHOULD** resend another BEGIN packet, resetting the session
- If a READY packet is lost, after a certain timeout, the client **SHOULD** resend another READY packet.
- If a Q4S-ALERT request or its corresponding answer is lost, after a certain timeout, the originator **SHOULD** resend another Q4S-ALERT packet.
- If a CANCEL request or its corresponding answer is lost, after a certain timeout, the originator **SHOULD** resend another CANCEL packet.

9.6.2. Outlier Samples

Outlier samples are those jitter or latency values far from the general/average values of most samples.

Hence, the Q4S default measurement method uses the statistical median formula for latency calculation, and the outlier samples are neutralized. This is a very common filter for noise or errors on signal and image processing.

9.7. Scenarios

Q4S could be used in two scenarios:

- client to ACP
- client to client (peer-to-peer scenario)

9.7.1. Client to ACP

One server:

It is the common scenario in which the client contacts the server to establish a Q4S session.

N servers:

In Content Delivery Networks and in general applications where delivery of contents can be achieved by different delivery nodes, two working mechanisms can be defined:

Starting mode: the end user may run Q4S against several delivery nodes and after some seconds choose the best one to start the multimedia session.

Prevention mode: during a streaming session, the user keeps several Q4S dialogs against different alternative delivery nodes. In case of congestion, the end user **MAY** change to the best alternative delivery node.

9.7.2. Client to Client

In order to solve the client-to-client scenario, a Q4S register function **MUST** be implemented. This allows clients to contact each other for sending the BEGIN message. In this scenario, the Register server would be used by peers to publish their Q4S-Resource-Server header and their public IP address to enable the assumption of the server role.

The register function is out of scope of this protocol version because different HTTP mechanisms can be used, and Q4S **MUST NOT** force any.

10. Security Considerations

10.1. Confidentiality Issues

Because Q4S does not transport any application data, Q4S does not jeopardize the security of application data. However, other certain considerations may take place, like identity impersonation and measurements privacy and integrity.

10.2. Integrity of Measurements and Authentication

Identity impersonation could potentially produce anomalous Q4S measurements. If this attack is based on spoofing of the server IP address, it can be avoided using the digital signature mechanism included in the SDP. The network can easily validate this digital signature using the public key of the server certificate.

Integrity of Q4S measurements under any malicious manipulation (such as a Man-in-the-Middle (MITM) attack) relies on the same mechanism, the SDP signature.

The Signature header field contains the signed hash value of the SDP body in order to protect all the SDP data, including the measurements. This signature not only protects the integrity of data but also authenticates the server.

10.3. Privacy of Measurements

This protocol could be supported over IPsec. Q4S relies on UDP and TCP, and IPsec supports both. If Q4S is used for application-based QoS, then IPsec is operationally valid; however, if Q4S is used to trigger network-based actions, then measurements could be incorrect unless the IPsec ports can be a target of potential action over the network (such as prioritizing IPsec flows to measure the new, upgraded state of certain application flows).

10.4. Availability Issues

Any loss of connectivity may interrupt the availability of the Q4S service and may result in higher packet loss measurements, which is just the desired behavior in these situations.

In order to mitigate availability issues caused by malicious attacks (such as DoS and DDoS), a good practice is to enable the Q4S service only for authenticated users. Q4S can be launched after the user is authenticated by the application. At this moment, the user's IP address is known, and the Q4S service may be enabled for this IP address. Otherwise, the Q4S service should appear unreachable.

10.5. Bandwidth Occupancy Issues

Q4S bandwidth measurement is limited to the application needs. It means that all available bandwidth is not measured, but only the fraction required by the application. This allows other applications to use the rest of available bandwidth normally.

However, a malicious Q4S client could restart Q4S sessions just after finishing the Negotiation phase. The consequence would be to waste bandwidth for nothing.

In order to mitigate this possible anomalous behavior, it is **RECOMMENDED** to configure the server to reject sessions from the same endpoint when this situation is detected.

11. Future Code Point Requirements

If the ideas described in this document are pursued to become a protocol specification, then the code points described in this document will need to be assigned by IANA.

11.1. Service Port

An assigned port would make possible a future Q4S-aware network capable of reacting by itself to Q4S alerts. A specific port would simplify the identification of the protocol by network elements in charge of making possible reactive decisions. Therefore, the need for a port assignment by IANA may be postponed until there is the need for a future Q4S-aware network.

Service Name: Q4S

Transport Protocol(s): TCP

Assignee:
Name: Jose Javier Garcia Aranda

Email: jose_javier.garcia_aranda@nokia.com

Contact:
Name: Jose Javier Garcia Aranda

Email: jose_javier.garcia_aranda@nokia.com

Description:

The service associated with this request is in charge of the establishment of new Q4S sessions, and during the session, manages the handoff to a new protocol phase (Handshake, Negotiation and Continuity) as well as sends alerts when measurements do not meet the requirements.

Reference: This document. This service does not use IP-layer broadcast, multicast, or anycast communication.

12. IANA Considerations

This document has no IANA actions.

13. References

13.1. Normative References

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <https://www.rfc-editor.org/info/rfc7232>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <https://www.rfc-editor.org/info/rfc7233>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/info/rfc7235>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <https://www.rfc-editor.org/info/rfc3264>.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <https://www.rfc-editor.org/info/rfc4566>.

13.2. Informative References

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <https://www.rfc-editor.org/info/rfc3550>.

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

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <https://www.rfc-editor.org/info/rfc792>.

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-34, 14 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-transport-34>.

[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>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RENO] Mathis, M., Semke, J., Mahdavi, J., and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", ACM SIGCOMM Computer Communication Review, pp. 67-82, DOI 10.1145/263932.264023, July 1997, <https://doi.org/10.1145/263932.264023>.

[RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <https://www.rfc-editor.org/info/rfc3649>.

[CUBIC] Rhee, I., Xu, L., and S. Ha, "CUBIC for Fast Long-Distance Networks", Work in Progress, Internet-Draft, draft-rhee-tcpm-cubic-02, 26 August 2008, <https://datatracker.ietf.org/doc/html/draft-rhee-tcpm-cubic-02>.

[CTCP] Sridharan, M., Tan, K., Bansal, D., and D. Thaler, "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks", Work in Progress, Internet-Draft, draft-sridharan-tcpm-ctcp-02, 11 November 2008, <https://datatracker.ietf.org/doc/html/draft-sridharan-tcpm-ctcp-02>.

Acknowledgements

Many people have made comments and suggestions contributing to this document. In particular, we would like to thank:

Victor Villagra, Sonia Herranz, Clara Cubillo Pastor, Francisco Duran Pina, Michael Scharf, Jesus Soto Viso, and Federico Guillen.

Additionally, we want to thank the Spanish Centre for the Development of Industrial Technology (CDTI) as well as the Spanish Science and Tech Ministry, which funds this initiative through their innovation programs.

Contributors

Jacobo Perez Lajo
Nokia Spain
Email: jacobo.perez@nokia.com

Luis Miguel Diaz Vizcaino
Nokia Spain
Email: Luismi.Diaz@nokia.com

Gonzalo Munoz Fernandez
Nokia Spain
Email: gonzalo.munoz_fernandez.ext@nokia.com

Manuel Alarcon Granero
Nokia Spain
Email: manuel.alarcon_granero.ext@nokia.com

Francisco Jose Juan Quintanilla
Nokia Spain
Email: francisco_jose.juan_quintanilla.ext@nokia.com

Carlos Barcenilla
Universidad Politecnica de Madrid

Juan Quemada
Universidad Politecnica de Madrid
Email: jquemada@dit.upm.es

Ignacio Maestro
Tecnalia Research & Innovation
Email: ignacio.maestro@tecnalia.com

Lara Fajardo Ibañez
Optiva Media
Email: lara.fajardo@optivamedia.com

Pablo López Zapico
Optiva Media
Email: Pablo.lopez@optivamedia.com

David Muelas Recuenco
Universidad Autonoma de Madrid
Email: dav.muelas@uam.es

Jesus Molina Merchan
Universidad Autonoma de Madrid
Email: jesus.molina@uam.es

Jorge E. Lopez de Vergara Mendez
Universidad Autonoma de Madrid
Email: jorge.lopez_vergara@uam.es

Victor Manuel Maroto Ortega
Optiva Media
Email: victor.maroto@optivamedia.com

Authors' Addresses

Jose Javier Garcia Aranda
Nokia
María Tubau 9
28050 Madrid
Spain
Phone: +34 91 330 4348
Email: jose_javier.garcia_aranda@nokia.com

Mónica Cortés
Nokia
María Tubau 9
28050 Madrid
Spain
Email: monica.cortes_sack@nokia.com

Joaquín Salvachúa
Universidad Politecnica de Madrid
Avenida Complutense 30
28040 Madrid
Spain
Phone: +34 91 0672134
Email: Joaquin.salvachua@upm.es

Maribel Narganes
Tecnalia Research & Innovation
Parque Científico y Tecnológico de Bizkaia
Astondo Bidea, Edificio 700
E-48160 Derio Bizkaia
Spain
Phone: +34 946 430 850
Email: maribel.narganes@tecnalia.com

Iñaki Martínez-Sarriegui
Optiva Media
Edificio Europa II,
Calle Musgo 2, 1G,
28023 Madrid
Spain
Phone: +34 91 297 7271
Email: inaki.martinez@optivamedia.com