

Internet Draft

Intended status: Informational
Expires: June 2019

J.J. Garcia Aranda
Nokia
M. Cortes
J. Salvachua
Univ. Politecnica de Madrid
M. Narganes
Tecnalia
I. Martinez Sarriegui
Optiva

December 5, 2018

The Quality for Service Protocol
draft-aranda-dispatch-q4s-07.txt

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on **June 5, 2019**.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This memo describes an application level protocol for the standard communication of e2e QoS compliance information using a protocol based on Hypertext Transfer Protocol (HTTP), which forms the basis for the World Wide Web, and Session Description Protocol (SDP). Quality for Service Protocol (Q4S) provides a mechanism for latency, jitter, bandwidth and packet loss negotiation and monitoring, alerting 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 draft, it is application dependent (e.g. increase quality, reduce bit-rate) or even network dependent (e.g. change connection's quality profile).

Table of Contents

1	Introduction.....	5
1.1	Scope.....	6
1.2	Motivation.....	7
1.3	Summary of Features.....	8
1.4	Differences with OWAMP/TWAMP.....	9
2	Terminology.....	10
3	Overview of Operation.....	10
4	Q4S messages.....	22
4.1	Requests.....	22
4.2	Responses.....	23
4.3	Header Fields.....	25
4.3.1	Common Q4S Header Fields.....	25
4.3.2	Specific Q4S Request Header Fields.....	25
4.3.3	Specific Q4S Response Header Fields.....	27
4.4	Bodies.....	27
4.4.1	Encoding.....	27
5	Q4S method definitions.....	28
5.1	BEGIN.....	28
5.2	READY.....	28

5.3	PING.....	29
5.4	BWIDTH.....	29
5.5	Q4S-ALERT.....	29
5.6	Q4S-RECOVERY.....	30
5.7	CANCEL.....	31
6	Response codes.....	31
6.1	100 Trying.....	31
6.2	Success 2xx.....	31
6.2.1	200 OK.....	31
6.3	Redirection 3xx.....	32
6.4	Request Failure 4xx.....	32
6.4.1	400 Bad Request.....	32
6.4.2	404 Not Found.....	32
6.4.3	405 Method Not Allowed.....	32
6.4.4	406 Not Acceptable.....	32
6.4.5	408 Request Timeout.....	33
6.4.6	413 Request Entity Too Large.....	33
6.4.7	414 Request-URI Too Long.....	33
6.4.8	415 Unsupported Media Type.....	33
6.4.9	416 Unsupported URI Scheme.....	33
6.5	Server Failure 5xx.....	33
6.5.1	500 Server Internal Error.....	33
6.5.2	501 Not Implemented.....	33
6.5.3	503 Service Unavailable.....	34
6.5.4	504 Server Time-out.....	34
6.5.5	505 Version Not Supported.....	34
6.5.6	513 Message Too Large.....	34
6.6	Global Failures 6xx.....	34
6.6.1	600 session does not exist.....	35
6.6.2	601 quality level not allowed.....	35
6.6.3	603 Session not allowed.....	35
6.6.4	604 authorization not allowed.....	35
7	Protocol.....	35
7.1	Protocol Phases.....	35
7.2	SDP Structure.....	37
7.2.1	"qos-level" attribute.....	38
7.2.2	"alerting-mode" attribute.....	39
7.2.3	"alert-pause" attribute.....	39
7.2.4	"recovery-pause" attribute.....	39
7.2.5	"public-address" attributes.....	40
7.2.6	"latency" attribute.....	40
7.2.7	"jitter" attribute.....	40
7.2.8	"bandwidth" attribute.....	40
7.2.9	"packetloss" attribute.....	40
7.2.10	"flow" attributes.....	41
7.2.11	"measurement" attributes.....	42
7.3	Measurements.....	43

7.3.1	Latency.....	44
7.3.2	Jitter.....	44
7.3.3	Bandwidth.....	45
7.3.4	Packet loss.....	47
7.4	Handshake Phase.....	48
7.5	Negotiation phase.....	49
7.5.1	Stage 0: Measurement of latencies and jitters.....	51
7.5.2	Stage 1: Measurement of bandwidth and packet loss.....	54
7.5.3	Quality constraints not reached.....	57
7.5.3.1	Actuator role.....	60
7.5.3.2	Policy server role.....	61
7.5.4	QoS Level changes.....	61
7.6	Continuity phase.....	62
7.7	Termination Phase.....	65
7.8	Dynamic constraints and flows.....	66
7.9	Qos-level upgrade and downgrade operation.....	67
8	General User Agent behavior.....	69
8.1	Roles in peer to peer scenarios.....	69
8.2	Multiple Quality sessions in parallel.....	70
8.3	General client behavior.....	71
8.3.1	Generating requests.....	72
8.4	General server behavior.....	72
9	Implementation Recommendations.....	73
9.1	Default client constraints.....	73
9.2	Latency and Jitter measurements.....	73
9.3	Bandwidth measurements.....	74
9.4	Packet loss measurement resolution.....	75
9.5	Measurements and reactions.....	76
9.6	Instability treatments.....	76
9.6.1	Loss of control packets.....	76
9.6.2	Outlier samples.....	76
9.7	Scenarios.....	77
9.7.1	Client to ACP.....	77
9.7.2	Client to client.....	78
10	Security Considerations.....	78
10.1	Confidentiality Issues.....	78
10.2	Integrity of measurements and authentication.....	78
10.3	Privacy of measurements.....	78
10.4	Availability issues.....	79
10.5	Bandwidth occupancy issues.....	79
11	IANA Considerations.....	79
11.1	Service Port.....	79
11.2	Protocol parameters.....	80
12	References.....	84
12.1	Normative References.....	84
12.2	Informative References.....	85
13	Acknowledgments.....	86

14 Contributors.....	87
15 Authors' Addresses.....	88

1 Introduction

The World Wide Web (WWW) is a distributed hypermedia system which has gained widespread acceptance among Internet users. Although WWW browsers support other, preexisting Internet application protocols, the native and primary protocol used between WWW clients and servers is the HyperText Transfer Protocol (HTTP) (RFC 7230[1] to RFC 7235). 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 each other and exchange information with certain quality constraints.

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

- o Latency: The time a message takes to travel from source to destination. It may be approximated to $RTT/2$ (Round trip time), assuming the networks are symmetrical. In this context we will consider the statistical median formula.
- o Jitter: latency variation. There are some formulas to calculate Jitter, and in this context we will consider the arithmetic mean formula.
- o Bandwidth: bit rate of communication. To assure quality, a protocol must assure the availability of the bandwidth needed by the application.
- o Packet loss: The percentage of packet loss is closely related to bandwidth and jitter. Affects bandwidth because a high packet loss implies sometimes 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 like a larger time between arrivals, causing a jitter growth.

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

Q4S provides a mechanism for quality monitoring based on an HTTP syntax and SDP in order to be easily integrated in 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 and in real-time; raising alerts if quality parameters are below a given pre-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).

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. It means that Q4S is designed to be used jointly with other transport protocols such as RTP, TCP, QUIC, HTTP, etc.

Some existent transport protocols are focused in real-time media transport and certain connection metrics are available (case of RTP/RTCP). Other protocols such as QUIC provides low connection latencies as well as advanced congestion control. These protocols transport data efficiently and provides lot of functionalities. However, there is not any other quality measurement protocol apart from Q4S.

Q4S enable applications to become reactive under e2e network quality changes. To achieve it, an independent Q4S stack application must run in parallel to target application. Then, Q4S metrics may be used to trigger actions on 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 target application allow.

1.2 Motivation

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

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

A protocol to monitor QoS must address the following issues:

- o Must be ready to be used in conjunction with current standard protocols and applications, without forcing a change on them.
- o Must have a formal and compact way to specify quality constraints of the desired application to run.
- o Must have measurement mechanisms avoiding application disruption, minimizing network resources consumption.
- o 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.
- o After having alerted about the violation of quality constraints, must have specific messages to inform about recovery of quality constraints in corresponding directions (forward and reverse).

- o Must protect the data (constrains, measurements, QoS levels demanded from the network) in order to avoid the injection of malicious data in the measurements.

1.3 Summary of Features

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

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, and therefore does not impose any restrictions on the type of application, protocol or the type of 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.

Q4S session lifetime is composed of four phases with different purposes: Handshake, Negotiation, Continuity and Termination. Negotiation and Continuity phases perform network parameter measurements as 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 draft. Basically, Q4S defines how to transport application quality requirements and measurement results between client and server and providing monitoring and alerting too.

Q4S must be executed just before starting a client-server application which needs a quality connection in terms of latency, jitter, bandwidth and/or packet loss. Once 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's 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 with OWAMP/TWAMP

OWAMP(RFC 4656) [19] and TWAMP(RFC 5357) [20] are two protocols to measure network quality in terms of RTT, but has a different goal than Q4S. The main difference is the scope: Q4S is designed for assist reactive applications, while OWAMP/TWAMP is designed just to measure network delay.

Differences can be summarized in the following points:

- OWAMP/TWAMP is 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 required bandwidth for specific application.
- OWAMP/TWAMP does not have responsivity control (which defines the speed of protocol reactions under network quality changes), because this protocol is designed to measure network performance, not to assist reactive applications and does not detect the fluctuations of quality in certain time intervals to take reactive actions. However, responsivity control is a key feature of Q4S.
- OWAMP/TWAMP is not intended to run in parallel with reactive applications, but Q4S' goal is to run in parallel and assist reactive applications to take 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 RFC 2119 [3] RFC 8174 [13] 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 tutorial nature and does not contain any normative statements.

The first example shows the basic functions of a 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 requesting a specific service or application from the server. This first message must have a special URI (RFC 3986), 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 about 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:

- o The assigned Q4S session id.
- o The quality constraints required by the requested application.
- o The measurement procedure to use.
- o The alerting mode: 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.

- o 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.
- o b) Reactive scenario: the network is not Q4S aware. In this scenario alert notifications are sent to a specific node, called Actuator, which is in charge of taking 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 more quality for application flows.

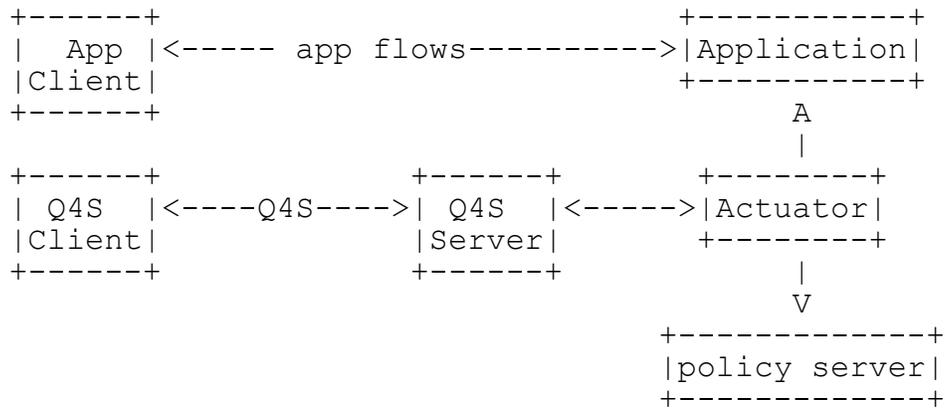


Figure 1 Reactive scenario

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

- o alert-pause: 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 fired even with violated network quality constraints in order to leave time for network reconfiguration or for application adjustments
- o recovery-pause: The amount of time the Q4S server waits before trying to recover initial qos-level. 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.

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 so as to take into account the new quality constraints in the measurement procedure.

Once the communication has been established (handshake phase is finished), the protocol will verify that the communication path between the client and the server meets the quality constraints on 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:

- o Stage 0: latency and jitter measurements
- o Stage 1: bandwidth and packet loss measurements

Stage 0 measurements are being taken through Q4S PING messages sent both 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, client can 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 fired 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 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 and 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 the Stage 0 for initiating the negotiation measurements again.

Stage 1 is optional. Its purpose is to measure the availability of application needed bandwidth. This stage can be skipped by client sending a "READY 2" message after completion of stage 0 when bandwidth requirements is set to zero kbps in the SDP. Stage 1 measurements are achieved through Q4S BWIDTH messages sent both from 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 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.

This figure depicts the message exchange in a successful scenario.

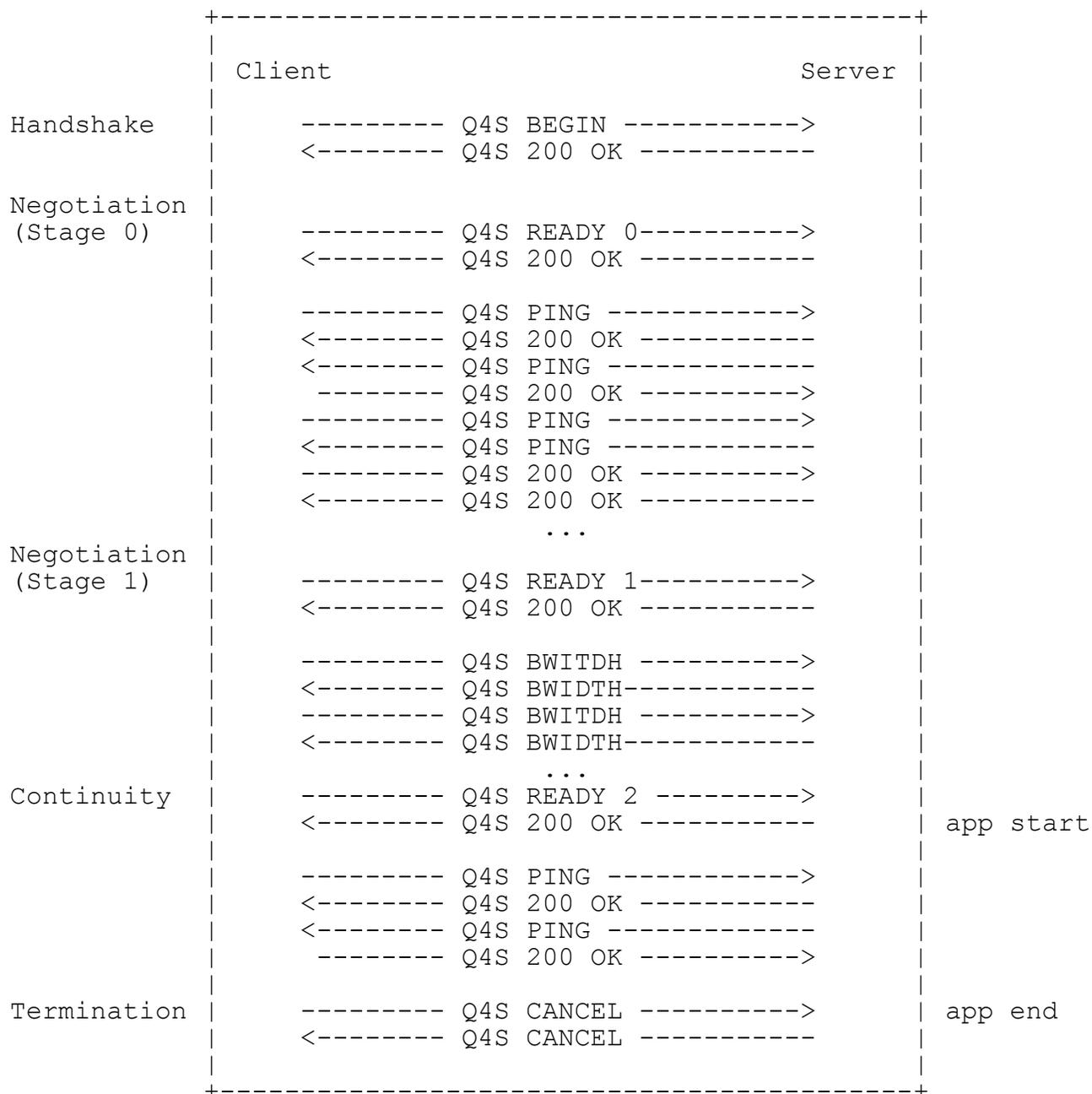
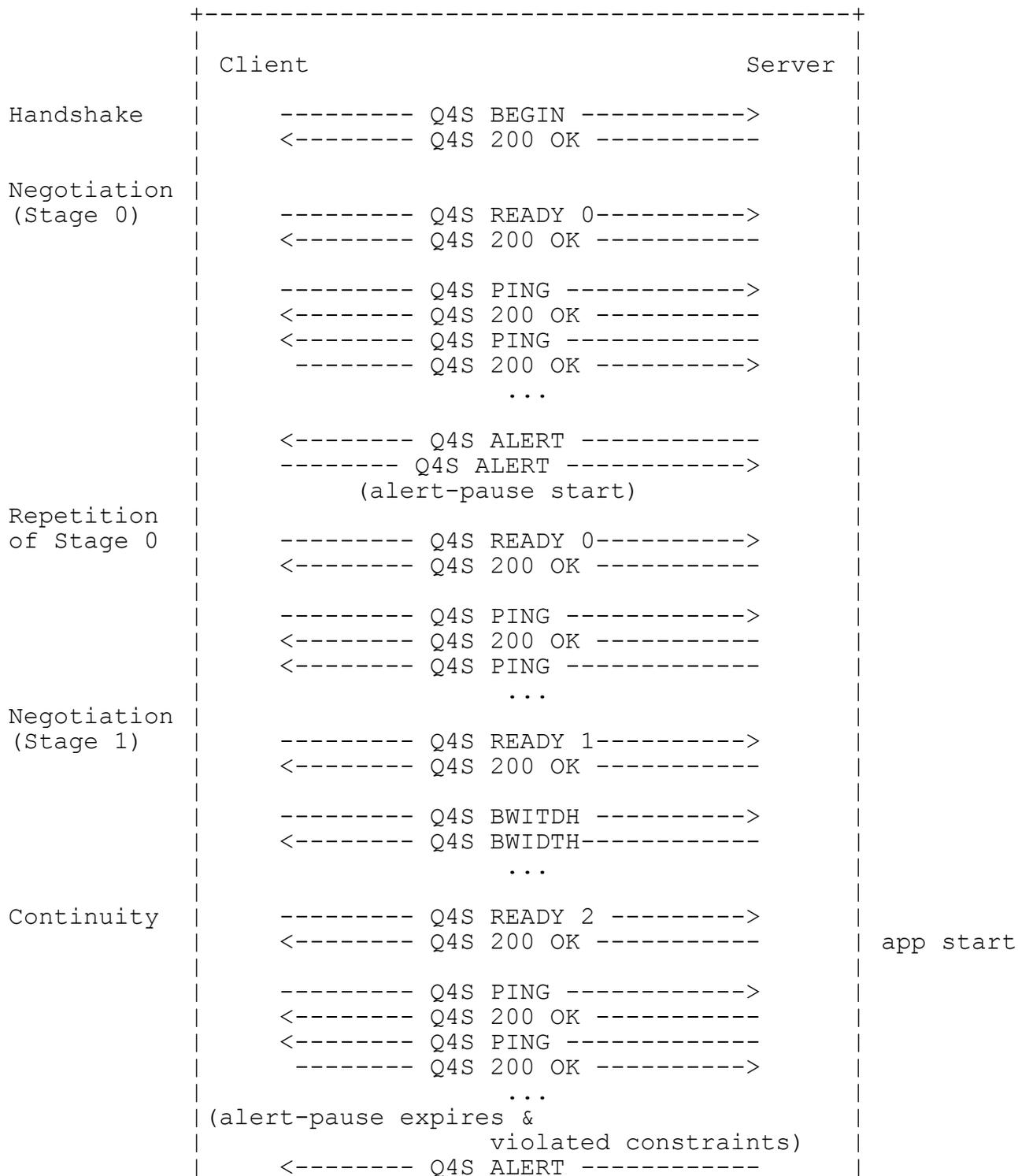


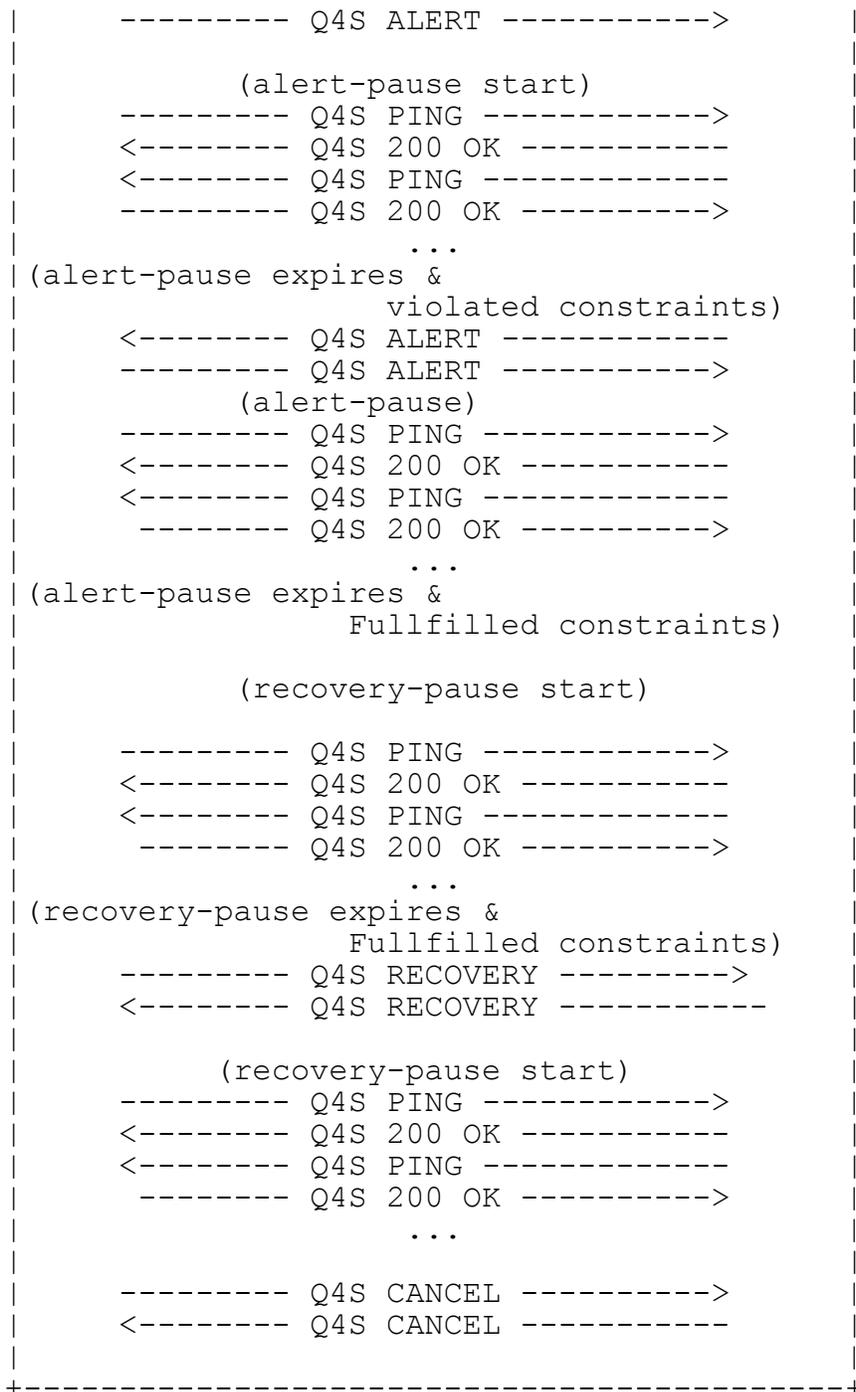
Figure 2 Successful Q4S message exchange.

Client and server measurements are included into PING and BWIDTH messages, allowing both sides of the communication 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, alerts are generated; and, later on, violation of quality constraints stops leading to the execution of the recovery process. The first example shows the Q4S-aware-network alerting mode scenario:



app start



Termination

app end

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 on 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 the following figure represents the Reactive scenario, in which alert notifications are sent from the server stack to the Actuator which is in charge of deciding either to act over application behavior and/or invoke a network policy server. The Actuator is an entity that has a pre-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 pre-defined set of different quality levels pre-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 -----          |
+-----+

```

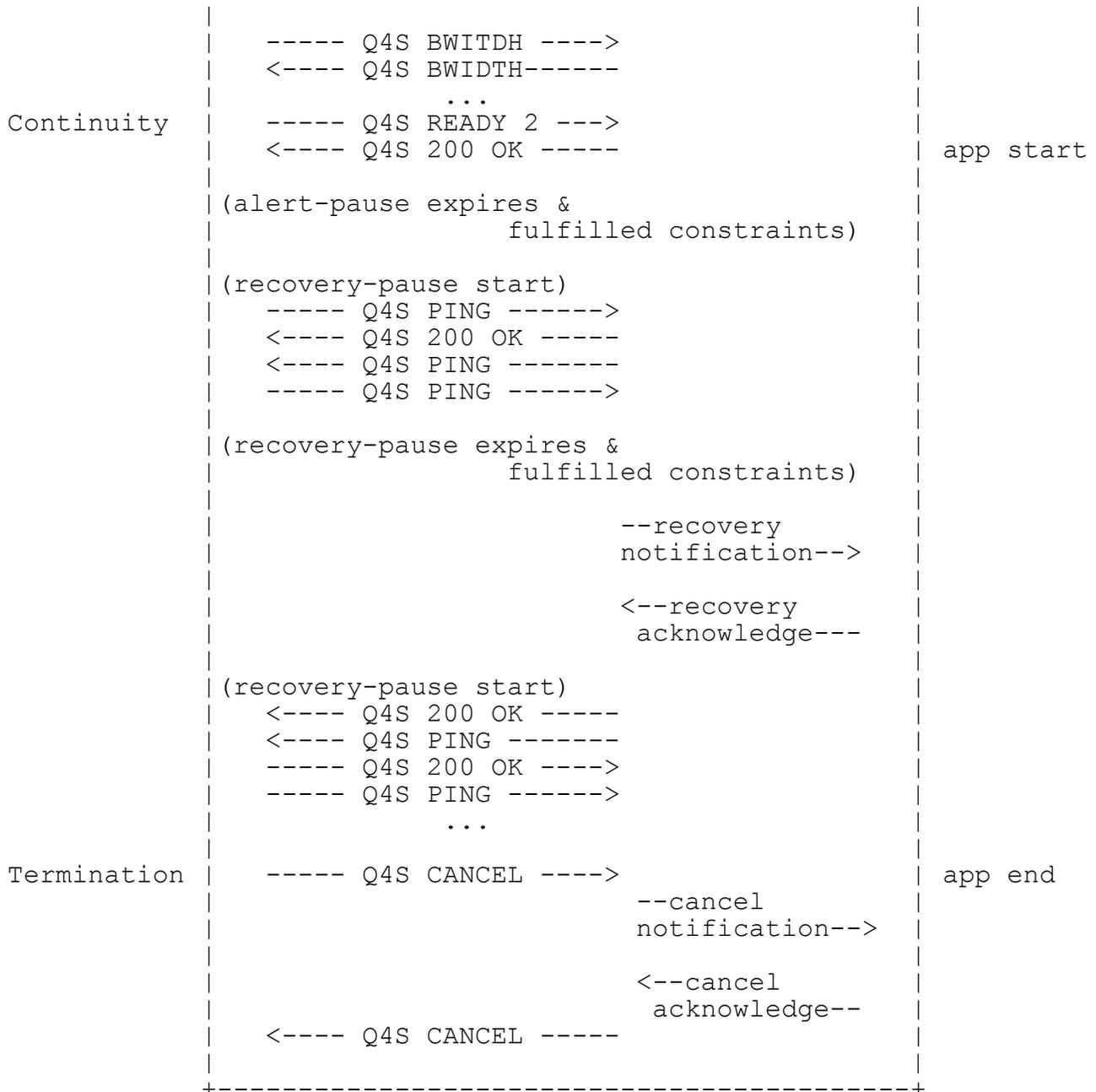


Figure 4 Reactive alerting mode.

At the end of any Negotiation phase stage, the server sends an alert notification to the Actuator if quality constraints are violated. During the period of time defined by the alert-pause parameter, 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 on 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 (RFC 3629 [11]). A Q4S message is either a request or a response.

Both Request and Response messages use the basic format of Internet Message Format (RFC 5322 [12]). 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.

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

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.

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 (LWS) 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 negotiate quality sessions, READY for synchronization of measurements, PING and BWIDTH for quality measurements purpose, CANCEL for terminating sessions, Q4S-ALERT for quality violations reporting, and Q4S-RECOVERY for quality recovery reporting.

Request-URI: The Request-URI is a Q4S URI (RFC 2396) as described in 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 upper-case. 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 described in HTTP (RFC 2616 [1]). 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. 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 not matching 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.

- o Session-Id: the value for this header is the same session id used in SDP (embedded in "o" SDP parameter) and is assigned by the server. The messages without SDP MUST include this header. If a message has and SDP body, this header is optional. The method of <session id> allocation is up to the creating tool, but it is suggested that a UTC timestamp be used to ensure uniqueness.
- o Sequence-Number: sequential and cyclic positive integer number assigned to PING and BWIDTH messages, and acknowledged in 200 OK responses.
- o Timestamp: this optional header contains the system time (with the best possible accuracy). It indicates the time in which the PING request was sent. If this header is present in PING messages, then the 200 OK response messages MUST include this value.
- o Stage: this is used in client's READY requests and 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

- o **User-Agent:** this header 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 which form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application.
- o **Signature:** this header 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 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 MD5 (RFC 1321 [6]) and RSA (RFC 8017 [7]) 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 (MD5 (<sdp>), <certificate>)

If the signature is not present, other validation mechanism MAY be implemented in order to provide assured quality with security and control.

- o **Measurements:** this header 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..9999]

Where "l" stands for latency followed by the measured value or an empty space, "j" stands for jitter followed by the measured value or an empty space, "pl" stands for packetloss followed by the measured value in % or an empty space and "bw" stands for bandwidth followed by the measured value or an empty space.

4.3.3 Specific Q4S Response Header Fields

- o 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), 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 (RFC 3261 [14]), but a compression/coding scheme will limit certain logical implementations of the way the request is parsed, thus, making the protocol concept more implementation dependent. In addition, bandwidth calculation may not be valid if compression is used. Therefore, the HTTP request header "Accept-Encoding" cannot be used in Q4S with different values than "identity" and if it is present in a request, the server MUST ignore it. In addition, the response header "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 (RFC 2616 [1] section 14.7). 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 semantics of this method is the starting of a quality session.

This method is only used during the handshake phase to retrieve the SDP containing session 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 cancelled 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 sending of PING and BWIDTH messages over UDP between clients and servers. The stage header included in this method is mandatory.

This message is only used in negotiation and continuity phases, and only just before making a measurement. Otherwise (out 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 MUST be answered in order to measure RTT and jitter, whereas BWIDTH is a long message (1 Kbyte or more) and MUST NOT be answered.

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

The Measurements header included 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 the client but also by server. Both (client and server) MUST NOT answer BWIDTH messages.

The Measurements header included 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 depends on the agreements between all involved parties.

Q4S-ALERT request does not have to be answered with a response message unless there is an error condition. However, after receiving a Q4S-ALERT request, the counterpart answers with a Q4S-ALERT request. The response to a Q4S-ALERT request is not cacheable.

This method MUST be initiated by the server in both alerting modes. In Q4S-aware-network alerting mode, the Q4S-ALERT messages are fired by the server and sent to the client, advising the network to react by itself. In Reactive alerting mode, alert notifications are triggered by the server stack and sent to the Actuator(see Figure1 "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 were being violated but they have been fulfilled during a period of time already (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 or that was decreased during violation of constraints). The Q4S-RECOVERY message does not contain any detail on the actions to be taken, which depends on the agreements between all involved parties.

Q4S- RECOVERY request MUST NOT be answered with a response message unless there is an error condition. However, after receiving a Q4S-RECOVERY request, the counterpart MUST answer with a Q4S- RECOVERY request. The response to a Q4S- RECOVERY request is not cacheable.

As for the Q4S-ALERT message, the Q4S-RECOVERY method is always initiated by the server in both alerting modes. In Q4S-aware-network alerting mode, the Q4S-RECOVERY messages are fired by the server and

sent to the client, advising the network to react by itself. In Reactive alerting mode, recovery notifications are triggered by the server stack and sent to the Actuator(see Figure1 "Reactive Scenario").

5.7 CANCEL

The semantics of CANCEL message is the release of the Q4S session id and the possible resources assigned to the session. This message could be triggered by 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. However, if the server receives a CANCEL message, it must answer with a CANCEL request message towards the client, acknowledging the reception.

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 application or at 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.

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 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 headers 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 of generating response entities that have content characteristics 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.

6.5.6 513 Message Too Large

The server was unable to process the request since 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 pair client/server

6.6.3 603 Session not allowed

The session is not allowed due to some policy (number of sessions allowed for the server is exceeded, or the time band of the Q4S-ALERT is not allowed for the pair client/server, 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 the 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 an 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 the following figure:

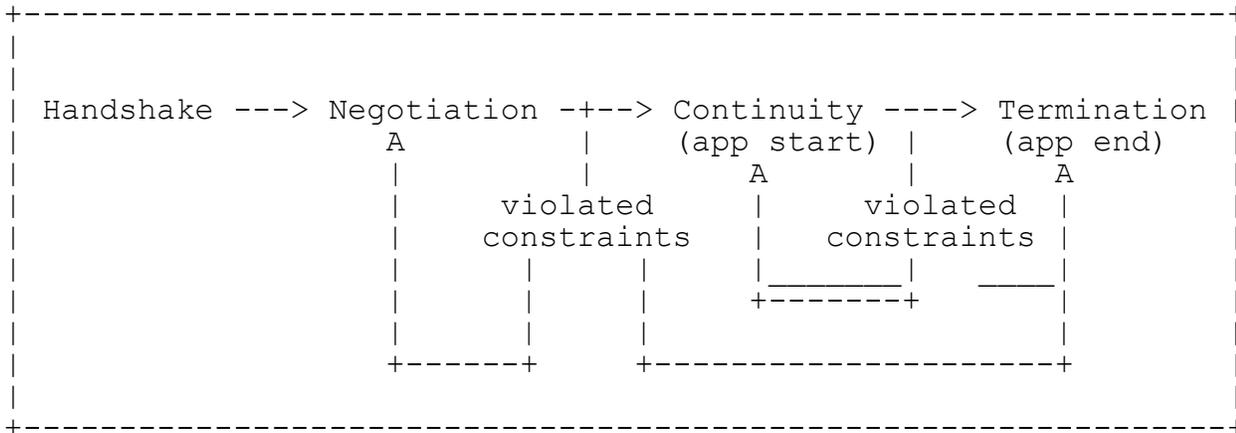


Figure 5 Session lifetime phases.

- o Handshake phase: in which the server is contacted by the client and in the answer message the quality constraints for the application is communicated embedded in an SDP.
- o 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 cancelled 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.
- o 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 Termination phase.

- o Termination phase: in which the Q4S session is terminated. The application may be closed too 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 attributes ("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>. 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 SDP message embedded 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.

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 case of constraint violation. There are two possible values:

- a) 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.
- b) 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 either network reconfigurations or application adjustments.

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).

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 his 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 network policy server or other external entities in charge of processing Q4S-ALERT messages.

7.2.6 "latency" attribute

The maximum latency (considered equal for uplink and downlink) tolerance are 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.

7.2.7 "jitter" attribute

The maximum uplink and downlink jitter tolerance are 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 "jitter" attribute is not present or has a 0 value, no jitter constraints need to be met and no measurements MAY be taken.

7.2.8 "bandwidth" attribute

The minimum uplink and downlink bandwidth are 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 "bandwidth" attribute is not present or has a 0 value, no bandwidth constraints need to be met and no measurements MAY be taken.

7.2.9 "packetloss" attribute

The maximum uplink and downlink packet loss tolerance are 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 "packetloss" attribute is not present or has a 0 value, no packetloss constraints need to be met and no measurements MAY be taken.

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 or UDP) (RFC 793 [8] and RFC 768 [9]) 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 to assume that measurements on q4s flows are the same experimented 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 specified Q4S ports (UDP and TCP) specified in the SDP.

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

If "clientListeningPort" is zero (a=flow:q4s clientListeningPort TCP/0), the client MAY choose one randomly as 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/NATP devices. If zero value or incorrect value is present, server must set the value to the received origin port in the next message with SDP (200 OK, ALERT and CANCEL messages).

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 draft 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:

- o 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 having different responsiveness values depending on the control resources used in each direction.

- o 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 having two different responsiveness values depending on the control resources used in each direction.
- o The third parameter is the time interval to be used to measure bandwidth during the negotiation phase.
- o The fourth parameter indicates the window size for jitter and latency calculations. Uplink and downlink values are separated by "/".
- o 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 latency, jitter, bandwidth and packet-loss measurement 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 messages will include the measured values in the Measurements header.

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 non-present attributes measurements MAY be omitted.

7.3.1 Latency

Latency measurements will be performed if the latency attribute and/or the application latency attribute are present and with 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, 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 application jitter attribute are present and with 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 message is received by an endpoint (either server or client), the corresponding jitter value is updated using the Statistical Jitter value calculated on the first 255 packets received using the arithmetic mean of the absolute values of elapsed times.

Each endpoint sends a PING periodically with a fixed interval, each value of "elapsed time" (ET) should be very close to this interval. If a PING message is lost, the elapsed time value is doubled. Identifying lost PING messages, however, is not an issue because all

PING messages are labeled with a Sequence-Number header. Therefore the receiver can discard this elapsed time 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 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 "Measurements" header of PING message.

7.3.3 Bandwidth

Bandwidth measurements will be performed if the bandwidth attribute and/or the application bandwidth attribute is present and with 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 using messages of 1 Kbyte or more in length. The messages are sent during the period of time defined in the third parameter of the SDP measurement default procedure attribute in millisecond units.

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

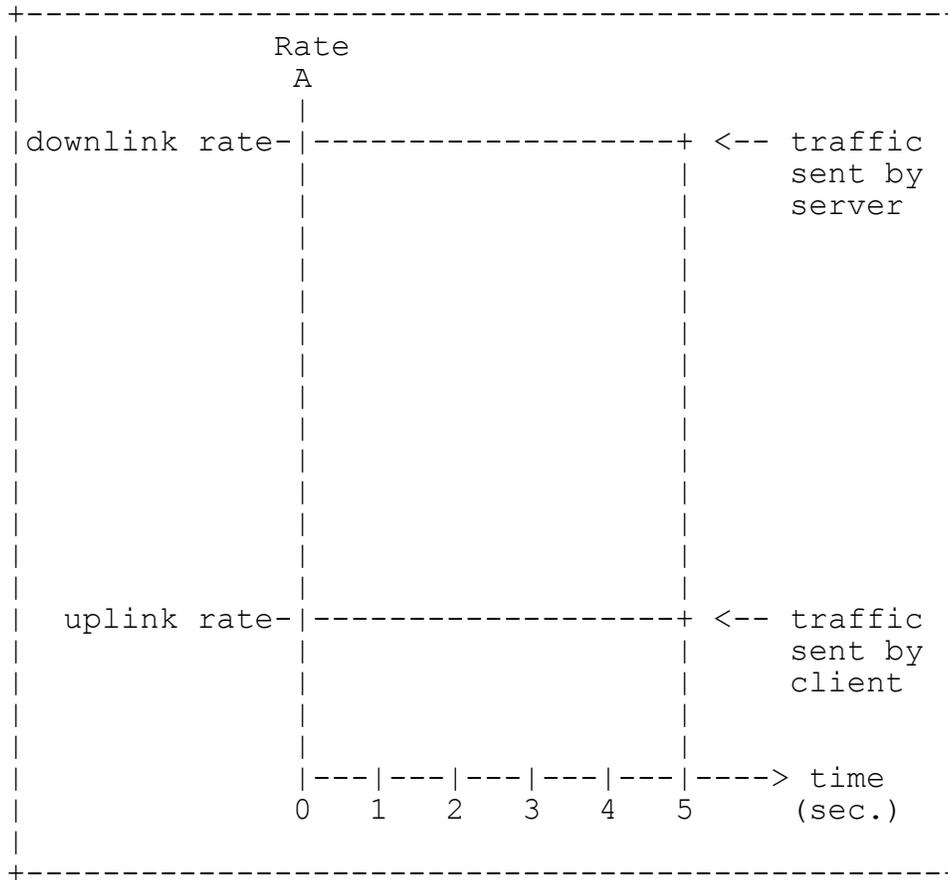


Figure 6 Bandwidth and packet loss measurements.

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 bit rate required by the bandwidth attribute.

During bandwidth measurement time, ALERTS are not expected, 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), and MUST include a Sequence-Number header with a sequential number starting at 0. 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
```

```
aaaaaaaaaaaa ( to complete 1024 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
```

```
aaaaaaaaaaaa ( to complete 1024 bytes 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 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.

The first Q4S message MUST have a special URI (RFC 3986 [4]), 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 (RFC 3464 [5]).

If the request is accepted, the server MUST answer it with a Q4S 200 OK message, which MUST contain an SDP body (RFC 4566 [2]) with the assigned session id (embedded in the "o" SDP parameter), 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 parameters 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 SLA is being violated.

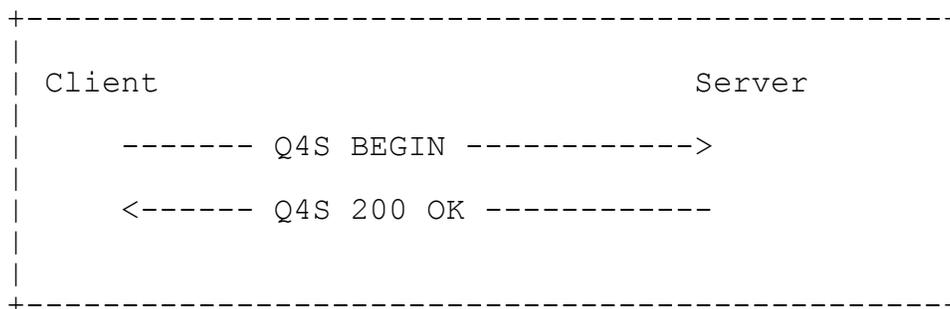


Figure 7 Handshake phase.

Example of Client Request and 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 headers 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 on 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.

- o 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.
- o 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 (RFC 3550 [10]).

Measurements of latency and jitter are done calculating the differences in 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:

- o Stage 0: Measurement of latencies, jitters and packet loss
- o 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 that allows for easy packet loss detection.

The client starts the negotiation phase sending a READY request using the TCP Q4S ports defined in the SDP. This READY request includes a "Stage" header 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 are specified, stage 0 is skipped.

7.5.1 Stage 0: Measurement of latencies and jitters

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 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.

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

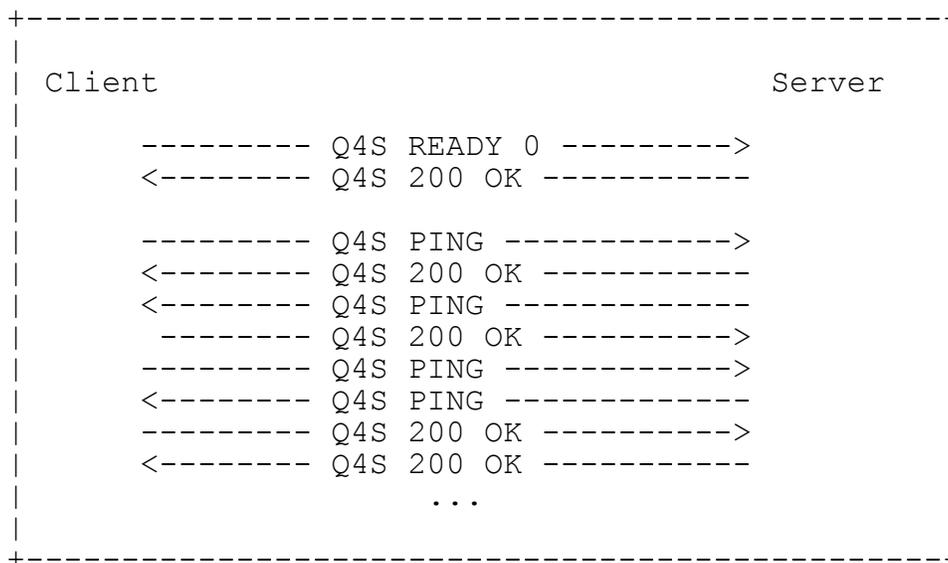


Figure 8 Simultaneous exchange of PING request and responses.

This 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. 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, using always the same UDP ports and incrementing the Sequence-Number with each message.

In the following example, the SDP measurement procedure attribute, this value is 50 milliseconds (from the client to the server) and 60ms (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 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, 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 his measurements to the client. Example:

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

Where l stands for latency, j for jitter, pl for packetloss 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, with the time in which the message has been sent. In case the header is present, the server MUST include the header 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 a 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:

- o The latency, jitter and packet loss constraints are reached in both directions
- o The latency, jitter and packet loss constraints are not reached in one or both directions

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

If the bandwidth constraints are empty or with value zero, the negotiation phase MUST terminate and both client and server may initiate the Continuity Phase. In this case client moves to Continuity phase by sending a READY message including the header "Stage: 2".

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

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. This READY message "Stage" header value is 1. The server answers with a Q4S 200 OK message to synchronize the initiation of the measurements.

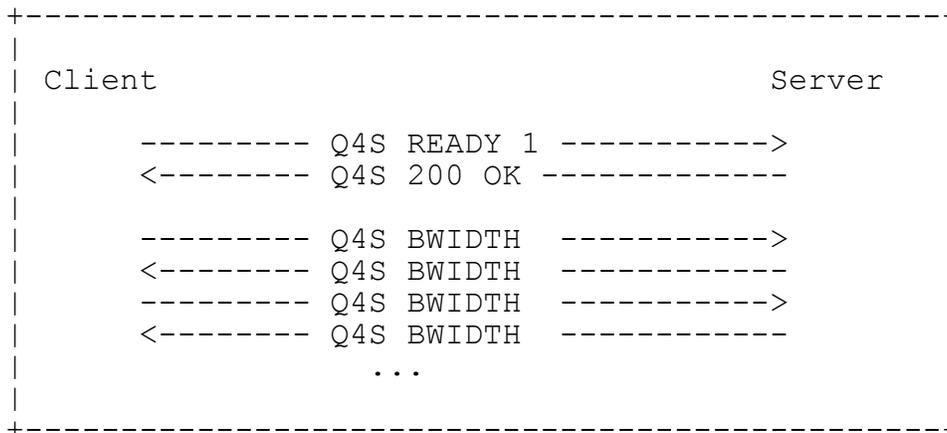


Figure 9 Starting bandwidth and packet loss measurement

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:

- o The bandwidth and packet loss constraints are reached in both directions
- o The bandwidth and packet loss constraints are not reached in one both directions.

In the first case, Stage 1 is finished. Client and server are ready for Continuity phase. The client moves to this phase by sending a READY message including the header "Stage: 2". The server answer MUST be 200 OK.

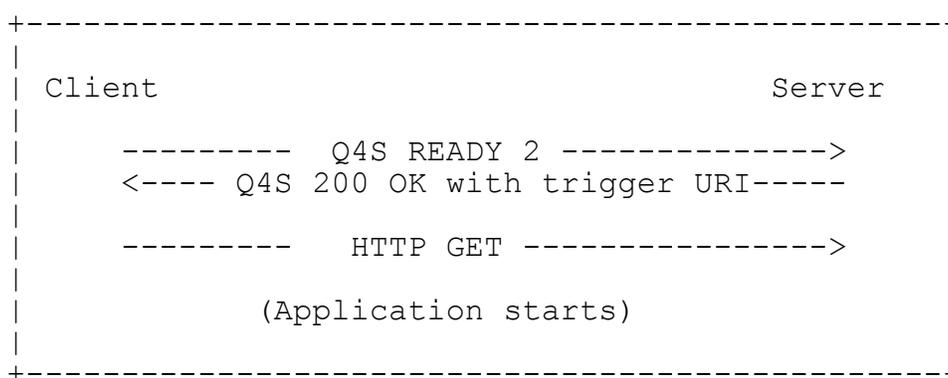


Figure 10 Trigger the application using HTTP URI

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 is present, the client SHOULD send an HTTP request to this URI.

The second case, with violated network constraints is explained in 7.5.4 Quality constraints not reached.

7.5.3 Quality constraints not reached

After finishing Stage 1 of the Negotiation phase, the client and the server have each other measured parameter values as these have been exchanged in the "Measurements" headers 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, 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, and the client MUST answer with the same Q4S-ALERT message. The Signature header contains the signed hash value of the SDP body in order to protect all the SDP the 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 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 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.

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

Actuator receives notifications of unmet requirements from Q4S server stack, and act over the application or over the network policy server, according with certain logic out of scope of this protocol.

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

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

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

Apart from actuate at application level, the actuator MAY act at 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. Network policy server may implement all or some of these features (but not exclusive to):

- o Server validation in terms of quality constraints.
- o Authentication (Signature validation) and security (block malicious clients)
- o Policy rules (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, client or server (depending on alerting mode) MAY trigger a Q4S-ALERT asking for higher qos-level attribute. The maximum qos-level allowed is 9, both uplink and downlink.

If the qos-level has reached the maximum value for 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 at 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 draft only covers the default procedure. The continuity operation with 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 this example, 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 reaches the maximum value, the client continues sending PING messages with the Sequence-Number header starting again at zero. When the server PING Sequence-Number header 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.

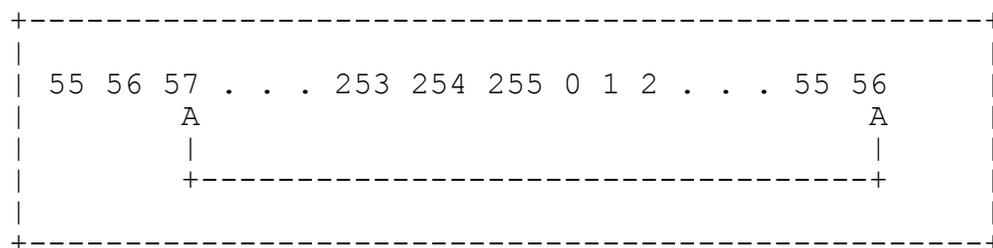


Figure 11 Sliding samples window

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

Both client and server will keep performing measurements but no other Q4S ALERT message MUST 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 draft.

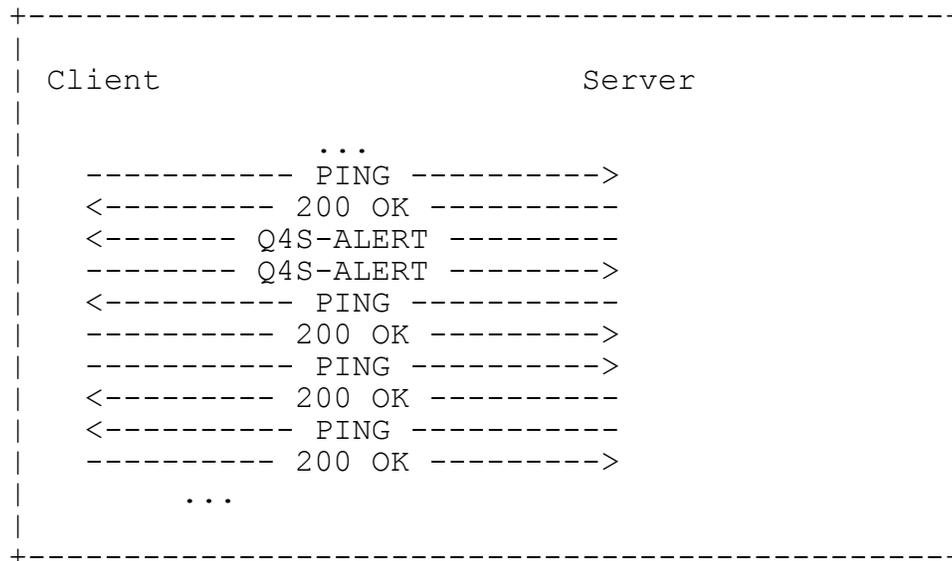


Figure 12 Continuity in QoS-aware-network alerting mode

In the Reactive scenario, 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.

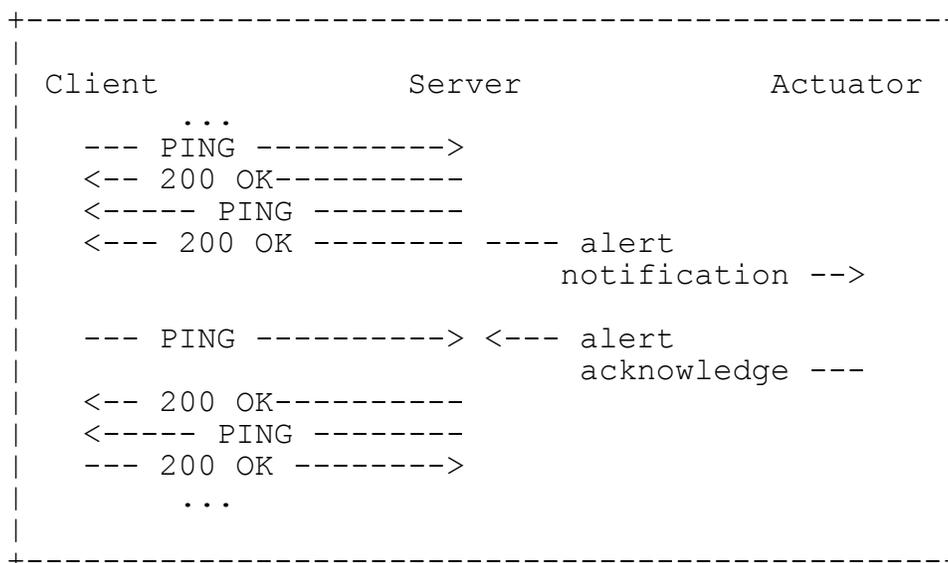


Figure 13 Continuity in Reactive alerting mode

7.7 Termination Phase

The Termination phase is end point 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 impossibility of the network to meet the required quality constraints. The client and server application will be notified by the respective Q4S stack.
- 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 cancelled and a new session will be initiated.

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

In Reactive alerting mode, Q4S CANCEL messages received by the Q4S server must cause the sending of cancel notifications sent from the server stack 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 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 as a result of two possibilities:

- o 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.
- o 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 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 [15], High Speed TCP (RFC 3649 [16]), CUBIC [17], Compound TCP (CTCP [18]), 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 change dynamically the packetloss constraints (defined in SDP) in order to make possible to reach a required throughput or vice versa (use packetloss measurement to change dynamically latency constraints).

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

$$Th = \text{Function}(\text{RTT}, \text{packet loss}, \dots)$$

Then, depending on RTT measured values, set dynamically the packetloss 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 a larger throughput, if more bandwidth is available.

For the Reno algorithm, the Mathis' formula may be used [16] for the upper bound on the throughput:

$$Th \leq (\text{MSS}/\text{RTT}) * (1 / \sqrt{p})$$

In absence of packet loss, a practical limit for the TCP throughput is the receiver window size divided by the round-trip time. 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 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 cancelled. But when the violation of constraints stops before reaching qos-level maximum value, the recovery mechanism allows for the qos-level upgrade gradually.

Following, this downgrade and upgrade of qos-level is explained with an example:

1. A Q4S session is initiated successfully with qos-level=0.

2. During the continuity phase, violation of constraints is detected; qos-level is increased to 1, a Q4S-ALERT is sent by the server to the client and alert-pause timer is started.
3. Alert-pause timer expires and still violation of constraints is detected; qos-level is increased to 2, a Q4S-ALERT is sent by the server to the client and alert-pause timer is started.
4. Alert-pause timer expires but violation of constraints has stopped; recovery-pause is started.
5. Recovery-pause timer expires, and no violation of constraints has been detected meanwhile; qos-level is decreased to 1, a Q4S-RECOVERY is sent by the server to the client and recovery-pause timer is started again.
6. Recovery-pause timer expires again and no violation of constraints has been detected meanwhile; qos-level is decreased to 0 and a Q4S-RECOVERY is sent by the server to the client; recovery-pause timer is not started this time as 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 app 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 modifications 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 to measure at lower levels of quality while application flows continue using a higher qos level value.

- o 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 Q4S flows.

- o 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.

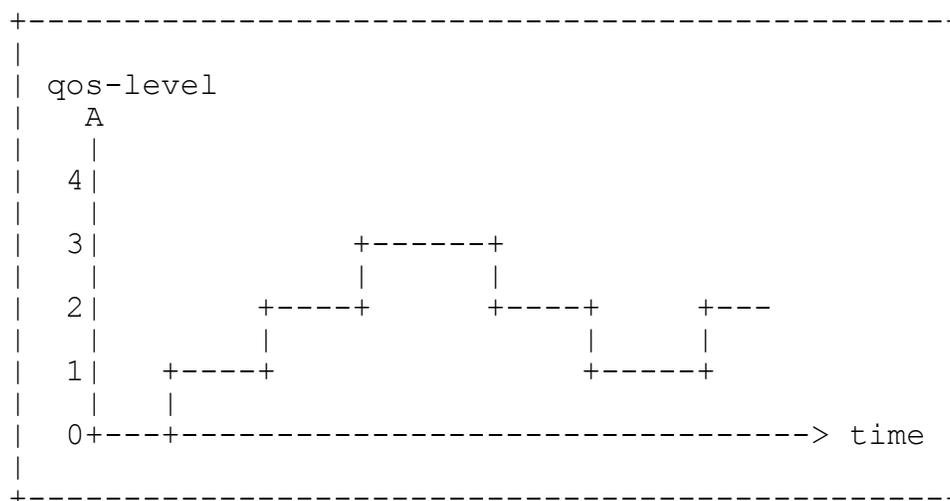


Figure 14 Possible evolution of qos-level

This mechanism 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 client and server role. The role assumed depends on who sends the first message.

In a communication between two UAs, the UA that sends the Q4S BEGIN request in the first place, for starting the handshake phase, shall assume the client role.

If both UASs send the BEGIN request at the same time, they will wait for a random time to restart again.

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

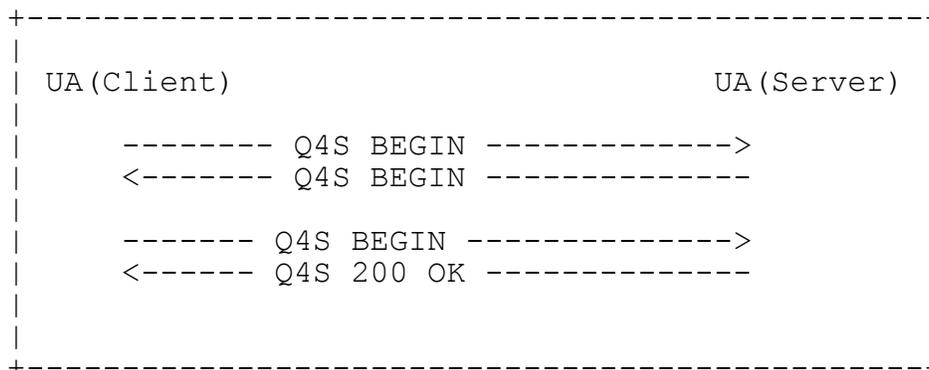


Figure 15 P2P roles.

8.2 Multiple Quality sessions in parallel

A Q4S session is intended to be used for an application. It means that for using the application, the client MUST establish only one Q4S session against the server. Indeed, the relation between session-id and 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) can execute different Q4S clients to establish separately different Q4S sessions but it is not recommended, because:

- o The establishment of a new Q4S session may affect other running applications over other Q4S sessions during bandwidth measurement.
- o 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,Z to designate each different behavior (follow the letter bullets in the figure 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 in 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:

Y) At latency, jitter and packet loss measurement, the PING requests are sent periodically, not 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) At 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.

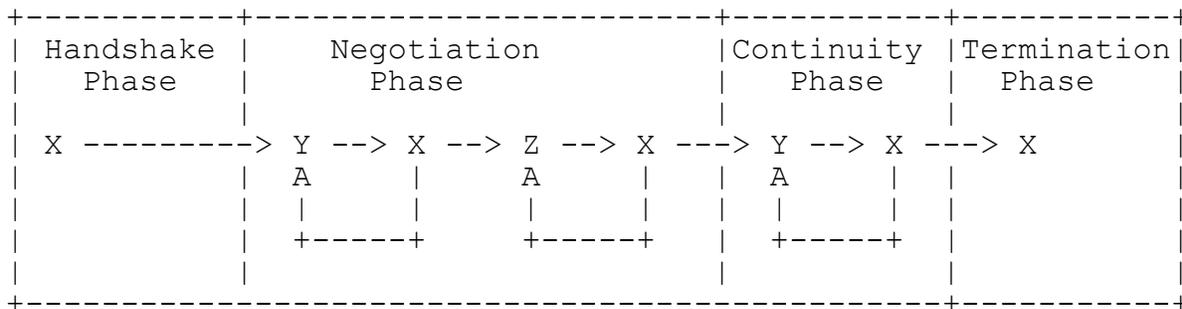


Figure 16 Phases & client behaviors.

8.3.1 Generating requests

A valid Q4S request formulated by a Client MUST, at a minimum, contains the following header fields:

- o If no SDP is included: the header Session-Id and Sequence-Number are mandatory.
- o If SDP is included: Session-Id is embedded into SDP, therefore the inclusion of Session-Id header 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 a new SDP with updated values, 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 server MUST send packets to measure jitter, latency and bandwidth. Therefore, the different behaviors of server are (follow the letter bullets in the figure 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:

S) At latency, jitter and packet loss measurement, the PING requests are sent periodically by the client but 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) At 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.

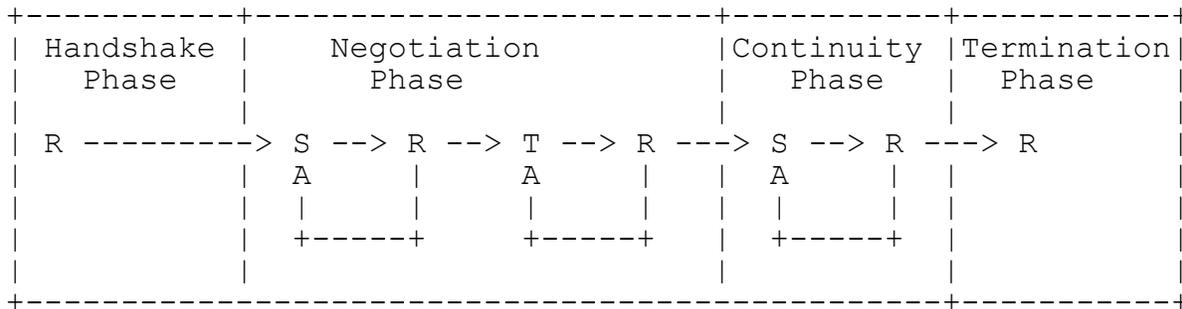


Figure 17 Phases & server behaviours.

9 Implementation Recommendations

9.1 Default client constraints

To provide a default configuration, it would be good that 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 only may be considered ended when neither client nor server receive new PING messages after an implementation-dependent guard time. Only after, client can send a "READY 1" message.

In execution systems, where the timers are not accurate, a recommended approach consists of including the optional header "Timestamp" 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 11Mbps, 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 required bandwidth and accuracy that the programmer wants to achieve.

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

- 1) Compute Target bandwidth / 8000 bits. In the example above is $11\text{Mbps}/8000 = 1375$ 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 sub-multiple 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:

$$1000/22 = 45 > 42$$

$$1000/23 = 43 > 42$$

$$1000 / 24 = 41.6 < 42$$

We can choose 24 ms but then we need to cover 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 packets are used, the more constant is 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 application nature and network conditions, a packet loss resolution less than 1% may be needed. In such case, there is no limit to the number of samples used for this calculation. A tradeoff 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 packetloss measured value.

The problem of this approach is the reliability of old samples. If the interval used between PING messages is 50ms, 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 for measure and trigger network tuning and application level actions (i.e. lowering video bit-rate, reduce 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 lost, some undesirable consequences could arise.

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

- o If a BEGIN packet is lost or its corresponding answer, after a certain timeout, the client SHOULD resend another BEGIN packet, resetting the session
- o If a READY packet is lost, after a certain timeout, the client SHOULD resend another READY packet.
- o If a QOS ALERT request is lost or its corresponding answer, after a certain timeout, the originator SHOULD resend another Q4S-ALERT packet.
- o If CANCEL request is lost or its corresponding answer, 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 Q4S default measurement method uses the statistical median formula for latency calculation, the outlier samples are neutralized. This is a very common filtering for noise or errors on signal and image processing.

9.7 Scenarios

Q4S could be used in two scenarios:

- o client to ACP (Application content provider)
- o client to client (peer to peer scenario)

9.7.1 Client to ACP

One server:

It is the common scenario in which client contact 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

- o Starting mode: End-user may run Q4S against several delivery nodes and after some seconds choose the best one to start the multimedia session
- o Prevention mode: During streaming session, user keeps several Q4S dialogs against different alternative delivery nodes. In case of congestion, 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 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 make possible the assumption of 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

Hence 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 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 MITM attack) relay on the same mechanism, the SDP signature.

The Signature header 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 relays on UDP and TCP, and IPsec supports both. If Q4S is used for application-based QoS, then IPsec is operationally valid but if Q4S is used to trigger network-based actions, then measurements could be wrong, unless IPsec ports be considered at any potential action over the network (such as prioritization of certain application flows).

10.4 Availability issues

Any loss of connectivity may interrupt the availability of Q4S service, and results into 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 Q4S service only for authenticated users. Q4S can be launched after user is authenticated by the application. At this moment, his IP address is known and the Q4S service may be enabled for this IP address. Otherwise 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 normally the rest of available bandwidth.

However, a malicious Q4S client could re-starts 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 end-point when this situation is detected.

11 IANA Considerations

11.1 Service Port

IANA is requested to assign a specific port for Q4S TCP control flow mechanism:

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 pass to a new protocol stage (handshake, negotiation and continuity) as well as inform of alerts when measurements do not meet the requirements.

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

11.2 Protocol parameters

IANA is requested to allocate the SDP parameters at "Foo-bar TLV Types" belonging to "The Foo Protocol Parameters" registry, from the range 0x42 to 0xfe.

This is the list of attribute field names to register:

Attribute name: qos-level
Type of attribute: session level
Subject to the charset attribute: NO
Explanation of purpose: defines the current QoS profile in uplink and downlink for the communication between client and server. The exact meaning of each level is implementation dependent but in general, a higher qos-level value corresponds to a better quality network profile.
Appropriate attribute values: [0..9] "/" [0..9]

Attribute name: alerting-mode
Type of attribute: session level
Subject to the charset attribute: NO
Explanation of purpose: defines the receiver of the Q4S alerts sent by the server. In Q4S-aware-network alerting mode, Q4S alerts are sent to the client. In this case the network is supposed to be Q4S aware, and reacts by itself to these alerts. In Reactive alerting mode, Q4S alerts sent to the network policy server. In this case the network is not Q4S aware and a specific node (policy server) is supposed to be in charge of achieving network tuning mechanisms. The "alerting-mode" attribute is optional, and its default value, when it is not present, is "Reactive".
Appropriate attribute values: <"Q4S-aware-network"|"Reactive">

Attribute name: alert-pause
Type of attribute: session level
Subject to the charset attribute: NO
Explanation of purpose: interval of time in milliseconds that the server must wait between Q4S-ALERT messages in order to allow network tuning operations. Measurements are not affected by this pause.
Appropriate attribute values: [0..60000]

Attribute name: public-address
Type of attribute: session level
Subject to the charset attribute: NO
Explanation of purpose: contains the public IP address of the client or the server.
Appropriate attribute values:<"client"|"server"><"IP4"|"IP6"> <value of IP address>

Attribute name: latency
Type of attribute: session level
Subject to the charset attribute: NO

Explanation of purpose: defines the latency constraints in milliseconds for the communication between client and server.
Appropriate attribute values: [0..9999]

Attribute name: jitter

Type of attribute: session level

Subject to the charset attribute: NO

Explanation of purpose: defines the jitter constraints in milliseconds in uplink and downlink for the communication between client and server.

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

Attribute name: bandwidth

Type of attribute: session level

Subject to the charset attribute: NO

Explanation of purpose: define the bandwidth constraints in kbps in uplink and downlink for the communication between client and server.

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

Attribute name: packetloss

Type of attribute: session level

Subject to the charset attribute: NO

Explanation of purpose: define the packet loss tolerance constraints in 100% in uplink and downlink for the communication between client and server.

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

Attribute name: application

Type of attribute: session level

Subject to the charset attribute: NO

Explanation of purpose: define the quality parameter tolerance constraints with uplink and downlink values for the communication between the client and the server for four different parameters: latency, jitter, bandwidth and packetloss.

Attribute values:

<"latency:"> [0..9999]

<"jitter:"|"bandwidth:"> [0..99999] "/" [0..99999]

<"packetloss:" [0.00 ..100.00] "/" [0.00 ..100.00]

Attribute name: flow

Type of attribute: session level

Subject to the charset attribute: NO

Explanation of purpose: define a flow between a client and a server. The flow involves purpose (application or q4s -control-), destination port (server or client) protocol (UDP or TCP) and port or range or ports

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

Attribute values:

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

Attribute name: measurement

Type of attribute: session level

Subject to the charset attribute: NO

Explanation of purpose: define the procedure to measure the quality and the different values for each measurement

```
Attribute values: "procedure/" <procedure> |
                  "latency "[0..9999] "/" [0..9999] |
                  "jitter "[0..9999] "/" [0..9999] |
                  "bandwidth "[0..99999] "/" [0..99999] |
                  "packetloss "[0.00..100.00] "/" [0.00..100.00]
```

If the attribute value is "procedure", the rest of the line MUST contain the name of the procedure and optional parameters, separated by ",".

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

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

where:

- o 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 having different responsiveness values depending on the control resources used in each direction.
- o 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 having two different responsiveness values depending on the control resources used in each direction.
- o The third parameter is the time interval to be used to measure bandwidth during the negotiation phase.

- o The fourth parameter indicates the window size for jitter and latency calculations. Forward and reverse values are separated by "/".
- o The fifth parameter indicates the window size for packet loss calculations. Forward and reverse values are separated by "/".

Other procedure names are allowed, but at least "default" procedure implementation is mandatory in client and servers.

12 References

12.1 Normative References

- [1] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.
- [2] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 4566, July 2006.
- [3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [4] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 3986, January 2005.
- [5] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with SDP", RFC 3264, June 2002.
- [6] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.
- [7] Moriarty, K., Johnsson, J., B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications version 2.2", RFC 8017, November 2016.
- [8] Defense Advanced Research Projects Agency, "Transmission Control Protocol", RFC 793, September 1981.
- [9] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

- [10] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V. "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.
- [11] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 3629, November 2003.
- [12] Resnick, P., "Internet Message Format", RFC 5322, October 2008
- [13] Leiba, S., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", RFC 8174, May 2007

12.2 Informative References

- [14] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A. Peterson, J., Sparks, R., Handley, M. and Schooler, E. , "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [15] Mathis, M., Semke, J., Mahdavi, J., Ott, T., "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", Computer Communications Review, 27(3), July 1997.
- [16] Floyd, S., "HighSpeed TCP for a Large Congestion Windows", RFC 3649, December 2003.
- [17] Rhee, I., Xu, L., Ha, S., "CUBIC for Fast Long-Distance Networks", Internet-draft draft-rhee-tcpm-cubic-02, February 2009.
- [18] Sridharan, M., Tan, K., Bansal, D., Thaler, D., "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks", Internet-draft draft-sridharan-tcpm-ctcp-02, November, 2008.
- [19] Shalunov, S., Teitelbaum, B., Karp, A., Boote, J., and M. Zekauskas, "A One-way Active Measurement Protocol (OWAMP)", RFC 4656, September 2006.
- [20] Hedayat, K., Krzanowski, R., Morton, A., Yum, K., and J. Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)", RFC 5357, October 2008.

13 Acknowledgments

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.

14 Contributors

Jacobo Perez Lajo
Nokia Spain
Email: jacoboperez@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 Ibanez
Optiva Media
Email: lara.fajardo@optivamedia.com

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

15 Authors' Addresses

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

Monica Cortes
Universidad Politecnica de Madrid
Avenida Complutense 30
28040 Madrid
Spain
Email: cortesm@dit.upm.es

Joaquin Salvachua
Universidad Politecnica de Madrid
Avenida Complutense 30
28040 Madrid
Spain
Phone: +34 91 0672134
Email: jsalvachua@dit.upm.es

Maribel Narganes
Tecnalia Research & Innovation
Parque Cientifico y Tecnologico de Bizkaia
Geldo Auzoa, Edificio 700
E-48160 Derio (Bizkaia)
Spain
Phone: +34 946 430 850
Email: maribel.narganes@tecnalia.com

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