DISPATCH Working Group

Internet Draft
Intended status: Standards Track
Expires: December 2011

J.J. Garcia Aranda
J. Perez Lajo
L.M. Diaz Vizcaino
Alcatel-Lucent
C. Barcenilla
M. Cortes
J. Salvachua
J. Quemada
Univ. Politecnica de Madrid

June 27, 2011

The Quality for Service Protocol
draft-aranda-dispatch-q4s-01.txt

Status of this Memo

Copyright Notice

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 an 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 dependant (e.g. increase quality, reduce bit-rate) or even network dependant (e.g. change connection's quality profile).

Table of Contents

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 2616 [1]). 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:

- 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.

- Jitter: latency variation. There are some formulas to calculate Jitter, and in this context we will consider the statistical variance formula.

- Bandwidth: To assure the quality, a protocol MUST assure the availability of bandwidth needed by the application.

- 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.

Q4S provides a mechanism for quality monitoring based on HTTP 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 which have 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 an alert if quality parameters are below a given pre-negotiated threshold. Q4S describes when these alerts 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, in order to minimize the impact on them. To perform the measurements, two control flows are created on both communication paths (forward and reverse direction).

1.1. Motivation

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

- Enable real-time services and applications to verify whether network resources achieve a certain QoS level.
- Monitoring 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.
- Monitoring also applies to Peer to Peer (P2P) real-time applications
- Enable ISPs to offer QoS to any ACP or final user application in an accountable way
- 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:

- Must be ready to be used in conjunction with current standard protocols and applications, without forcing a change on them.
- Must have a formal and compact way to specify quality constraints of the desired application to run.
- Must have measurement mechanisms avoiding application disruption.
- 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.
- Must protect the data (constrains, measurements, QoS levels demanded from the network) in order to avoid the injection of malicious data in the measurements.

1.2. Summary of Features

Quality for Service is a message-oriented communication protocol that can be used in conjunction with any other application-level protocol.

The benefits in quality enhancement 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 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 constraints. It will raise alerts if the new constraints are violated.

Q4S session lifetime is composed of four phases with different purposes. Two 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 ~~SLA~~ and measurement results between client and server and provides, as well, monitoring and alerting.

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.

During the lifetime of the application, the protocol periodically renews the session measurements and alerts if the measured values of quality parameters do not meet the negotiated application requirements.

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.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [3].

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 message exchange is depicted in Figure 1.

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 forces 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 initial communication quality requirements in an SDP body.

This request is answered by the server with an Q4S 200 OK method letting the client know that it accepts the request. This message MUST contain an SDP body with the quality constraints required by the requested application and the measurement procedure to use.

Once the communication has been established, 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. This 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.

Measurements are being taken and communicated to each other through Q4S PING messages sent both from the client and the server. All Q4S PING requests will be answered by Q4S 200 OK messages to allow for bidirectional measurements.

After a pre-agreed number of measurements, determined by the measurement procedure as sent by the server, have been performed, the client will send a Q4S GET message to the server containing the measured values of all quality parameters from its point of view. The server receives this message and if the values meet the necessary requirements answers with a Q4S 200 OK method.

As the communication meets the conditions, the application will start. Q4S will continue to measure the communication parameters verifying that the real-time application can keep executing under quality conditions. This will be done through the Q4S PING message exchange on both connection paths.

Once the client wants to terminate the communication it sends a Q4S CANCEL message, which will be acknowledged by the server with another Q4S CANCEL message.

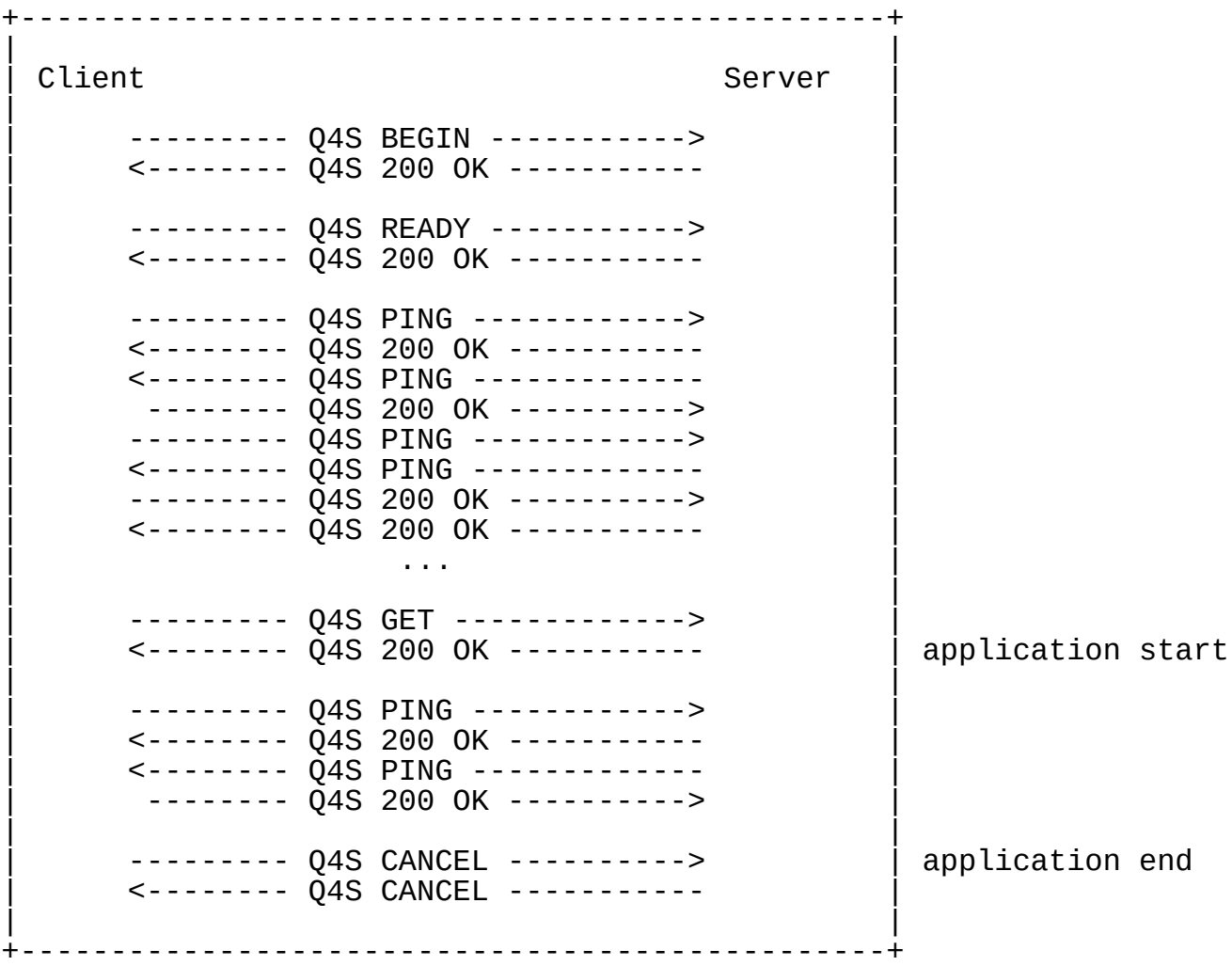

Figure 1 Basic Q4S message exchange.

The second example shows the behavior of Q4S protocol in the event of the loss of quality for a given parameter constraint: detection of a the violation of specific parameter constraints after continuous measurements of network parameters, the raising of an alert, the termination in case of non recovery of the quality communication. This message exchange is depicted in Figure 2.

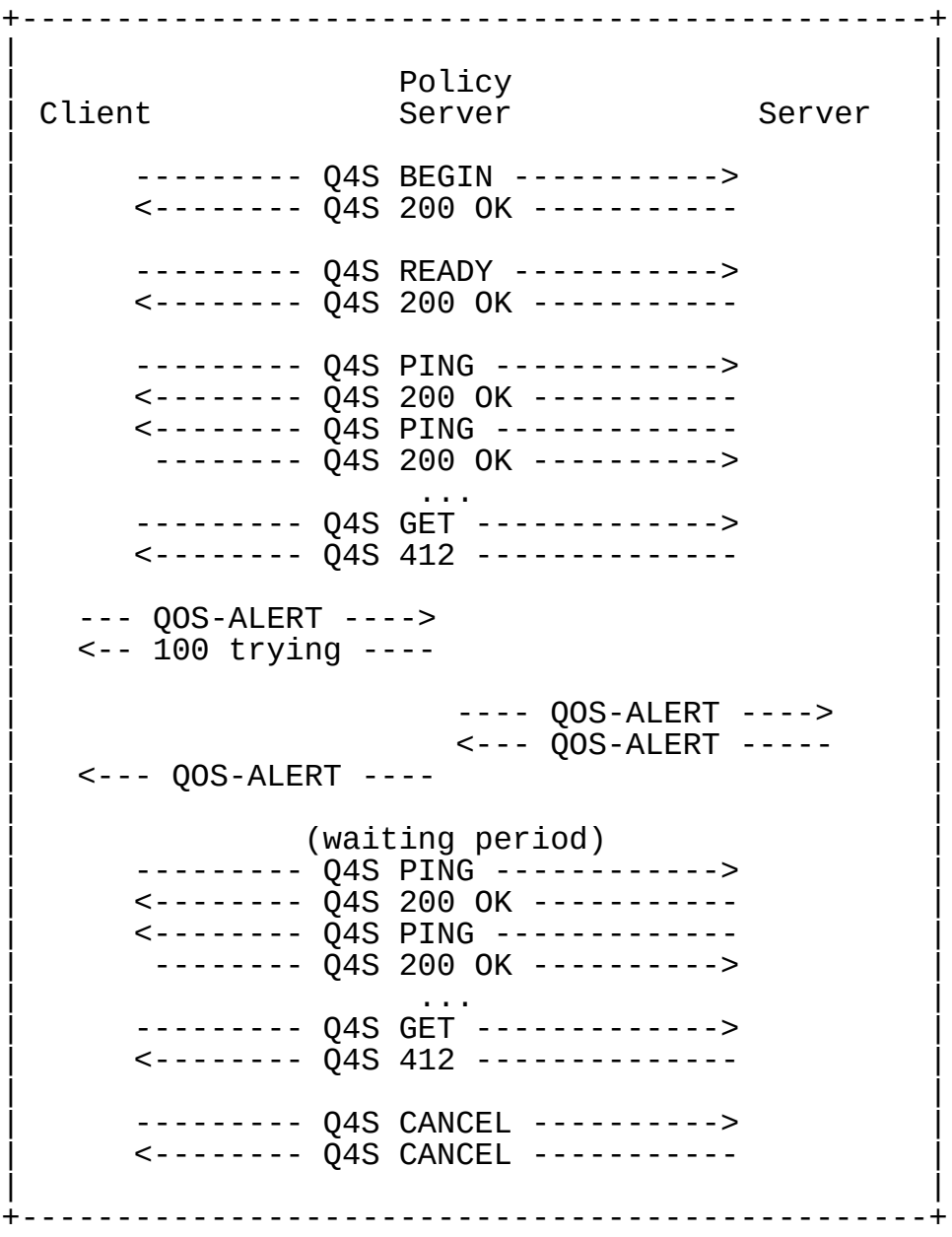

Figure 2 Q4S message exchange with quality constraints not reached.

The initiation of the communication happens in the same way as in the first example establishing communication from the client to the server through a Q4S BEGIN message answered by the server if ready

with a Q4S 200 OK. In this case, the server indicates the presence of a policy server, which will receive the alerts. The policy server is an entity that can act on the network. It has a set of different quality levels defined and pre-agreed upon between the ACD and the ISP.

Then the measurements are taken in order to verify if the communication conditions meet the quality constraints of the application through an exchange of Q4S PING requests and Q4S 200 OK responses. The client sends the server the measured values in a Q4S GET message and waits for the answer with the actions to take.

In this case, some or many of the quality constraints are not met and the server sends a Q4S 412 message indicating that a pre-condition as not been met.

The client will then send a Q4S QOS-ALERT message to the policy server. The policy server responds to the client with a Q4S 100 trying message and notifies the server of the received alert. The policy server will verify the authorization credentials and raise the quality of service level acting on the network elements as per SLA agreement between the ACP and the ISP, which are out of scope of this protocol description.

After a delay, the client and server will initiate the quality parameter measurements through more Q4S PING and Q4S 200 OK messages. If the quality of the communication cannot be met, the client closes the connection with a Q4S CANCEL message acknowledged by the server with another Q4S CANCEL.

4. 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.

4.1. 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 shown in Figure 3.

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

- 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 match the constraints. This phase is iterative until quality constraints are reached or the session is cancelled after a number of measurement cycles without meeting the quality constraints. Just after reaching the quality requirements, Q4S provides a simple optional mechanism to start the application, which will benefit from quality connection from the beginning, using HTTP.

- Continuity phase: in which quality is continuously measured. If quality becomes degraded, an alert shall be issued. New measurements may follow up to a negotiated maximum before moving to Termination phase if constraints cannot be met. In this phase the measurements MUST avoid disturbing application by consuming network resources.

- Termination phase: in which the session is terminated.

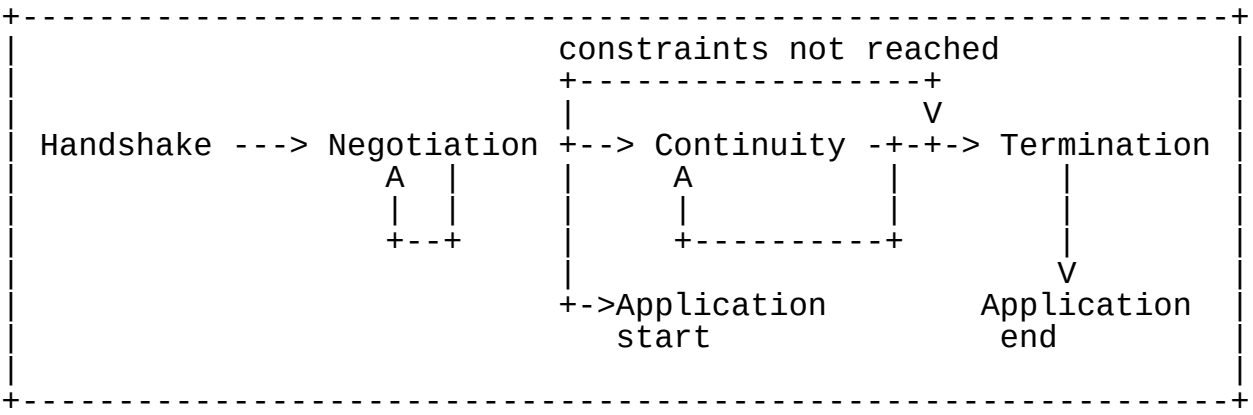

Figure 3 Session lifetime phases.

4.2. 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 this parameter and 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.

```
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=public-address:client IP4 192.0.2.33
a=public-address:server IP4 198.51.100.58
a=latency:40/35
a=jitter:10/10
a=bandwidth:20/6000
a=packetloss:5/5
a=flow:data downlink TCP/10000-20000
a=flow:control downlink UDP/55000
a=flow:control downlink TCP/55001
a=flow:data uplink TCP/56000
a=flow:control uplink UDP/56000
a=flow:control uplink TCP/56001
a=measurement:procedure default,50/50,75/75,,0
a=measurement:latency 10000/10000
a=measurement:jitter 10000/10000
a=measurement:bandwidth 0/0
a=measurement:packetloss 0/0
```

4.2.1. 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 protocol lifetime raising or lowering the value as necessary following the network measurements and the application needs.

4.2.2. 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.

4.2.3. latency attribute

The maximum uplink and downlink latency tolerance are specified in the “latency” attribute, expressed in milliseconds.

4.2.4. jitter attribute

The maximum uplink and downlink jitter tolerance are specified in the “jitter” attribute, expressed in milliseconds.

4.2.5. bandwidth attribute

The minimum uplink and downlink bandwidth are specified in the “bandwidth” attribute, expressed in kbps.

4.2.6. packetloss attribute

The maximum uplink and downlink packet loss tolerance are specified in the “packetloss” attribute expressed in percentage.

4.2.7. flow attributes

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

Several “flow” attributes can be defined. The goal is to monitor each flow to verify that the quality constraints are met. These flows identify the direction (uplink or downlink), the protocol (TCP or UDP) (RFC 761 [8] and RFC 768 [9]) and the ports that are going to be used by the application data and, of course, by the Q4S control flows (for quality measurements), because the quality measurements MUST be achieved over the same quality session for each direction. All defined flows will be considered within the same quality profile, which is determined by the qos-level attribute in each direction.

During negotiation phase the specified control ports will be used for Q4S messages, and this is the reason to separate application data ports from Q4S control ports, otherwise they could collide.

The control should involve two UDP flows, one uplink and one downlink, and two TCP flows, again one uplink and one downlink. Application data MAY consist of many flows, depending on the nature of the application. The handshake phase takes place through the Contact URI, using TCP port 80 for example. However, the negotiation

phase will take place on the specified control ports (UDP and TCP) using the Session URI.

The “downlink port” is a port in which the client listens for server requests and MUST be used as origin port of client responses. The “uplink port” is a port in which server is listening for incoming messages from the client and MUST be used as origin port of server responses.

If the server’s “downlink” port is null (a=flow:control downlink TCP/0), the client May choose one randomly as per OS standard rules. “Downlink” 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.

Figure 4 Downlink flow.

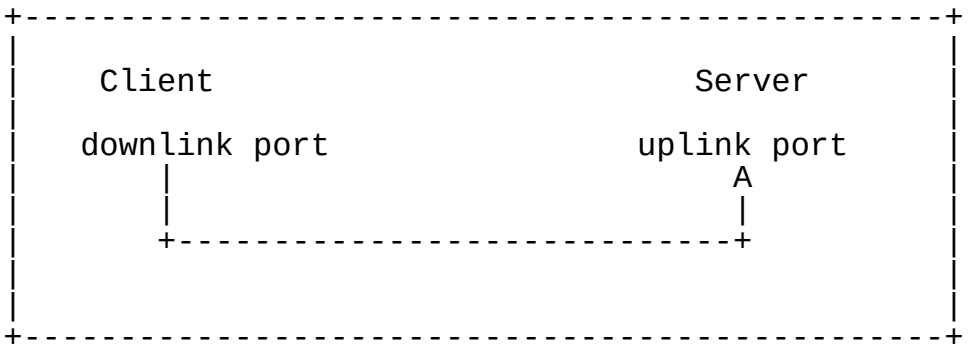

Figure 5 Uplink flow.

4.2.8. 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,50/50,75/75,5000,1,40/80,100/256

In the “default” procedure, the meaning of these parameters is:

- 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.
- The second parameter is the time interval (in milliseconds) between PING requests during the continuity phase. Forward and reverse values are separated by “/”. This allows having two different responsiveness values depending on the control resources used in each direction.
- The third parameter is the time interval to be used to measure bandwidth during the negotiation phase. If not present, a default value of 5000 ms MUST be assumed. Forward and reverse values are separated by “/”.

- The fourth parameter indicates the operation mode during the continuity phase. Two values are defined: 0 means “normal” mode and 1 means “sliding window” mode. If not present, normal mode (default value of 0) will be assumed. These modes of operation will be described in [4.7].

- The fifth parameter is only applicable in sliding window mode. It indicates the window size for jitter and latency calculations. If not present, a value of 256 MUST be assumed. Forward and reverse values are separated by “/”.

- The sixth parameter is only applicable in sliding window mode. It indicates the window size for packet loss calculations. If not present, a value of 256 MUST be assumed. Forward and reverse values are separated by “/”.

There are four more measurement attributes:

```
a=measurement:latency 10000/10000
a=measurement:jitter 10000/10000
a=measurement:bandwidth 0/0
a=measurement:packetloss 0/0
```

The latency, jitter, bandwidth and packetloss measurement attributes contain the values measured for each of these quality parameters in uplink and downlink directions. Quality parameters values in these measurement attributes provide a snapshot of the quality level reached in each measurement stage.

They can be omitted during the initial protocol phases as no measurements have been taken, but they MUST be filled in when sending GET requests and 412 responses.

4.3. Measurements

This section describes the way quality parameters are measured as defined by the “default” procedure.

4.3.1. Latency

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. As the forward and reverse latencies are unknown the client and server will assume that the network is symmetric. The latency will therefore be calculated as the average value of all the RTT samples divided by 2.

4.3.2. Jitter

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 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 statistical variance formula:

Jitter Statistical = SquareRootOf(SumOf((ElapsedTime[i]-Average)^2)/(ReceivedPacketCount-1))

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.

After 255 samples the client has the values of RTT and downlink jitter and the server has RTT and uplink jitter.

4.3.3. Bandwidth

In order to measure the available bandwidth, both the client and the server MUST start sending DATA 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 in length. The messages are sent during the period of time defined in the third parameter of the SDP measurement procedure attribute in millisecond units. If this parameter is not present, a value of 5 seconds SHOULD be used by default.

a=measurement:procedure default,50/50,75/75,5000,0

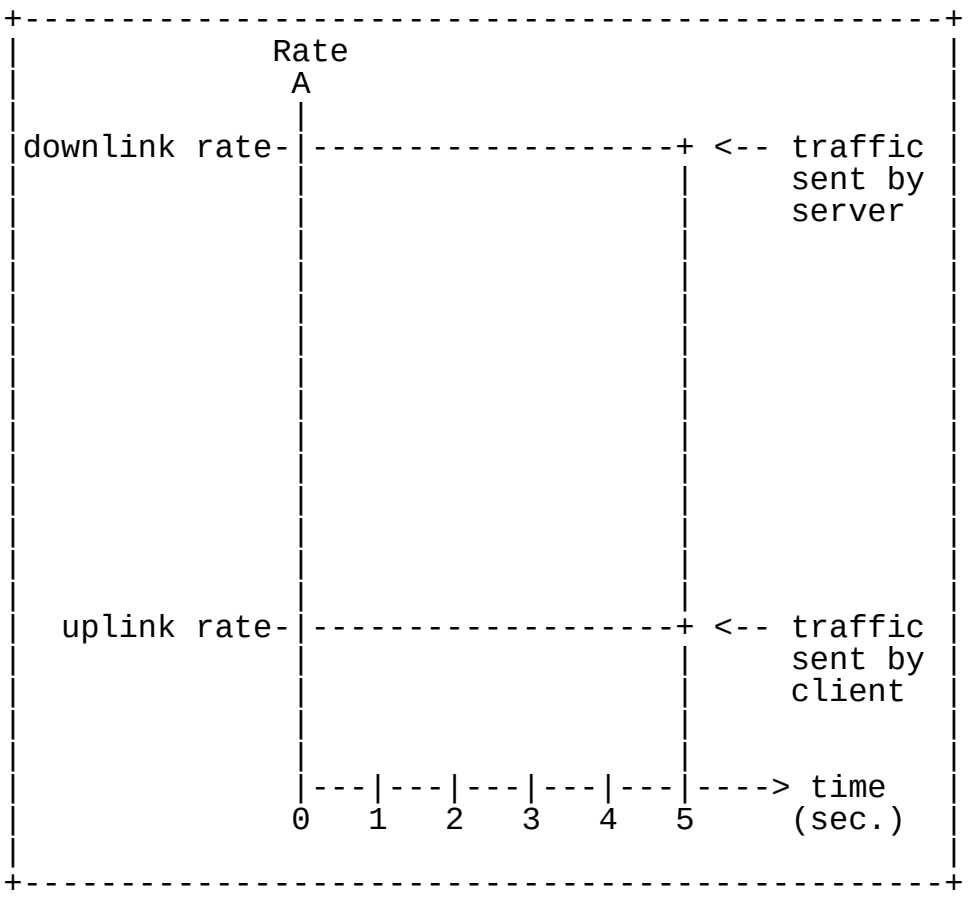

Figure 6 Bandwidth and packet loss measurements.

The goal of these measurements is not to identify the bandwidth of the Internet connection but to determine if the required bandwidth is available, meeting the application's constraints. Therefore, the requested bandwidth MUST be measured sending only that bit rate.

When measuring bandwidth, all DATA 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 sent DATA packet. If any measurement stage needs to be repeated, the values MUST start at zero again. DATA requests MUST NOT be answered. Examples:

```
Client message:
=========================
       DATA 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

       aaaaaaaaaaaaa ( to complete 1024 bytes UDP payload length)
=========================
```

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

```
server message:
=========================
       DATA q4s://www.example.com Q4S/1.0
       Session-Id: 53655765
       Sequence-Number: 0
       Content-Type: text
       Content-Length: XXXX

       aaaaaaaaaaaaa ( to complete 1024 bytes UDP payload length)
=========================
```

When the measurement time interval is over, the client and the server have a collection of server messages and can calculate the downlink and uplink bandwidth respectively.

4.3.4. Packet loss

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

4.4. 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) and the server can take them into account when it builds the answer with the final values, following a request / response schema (RFC 3464 [5]). The description of these quality parameters is attached in an SDP document.

If the request is accepted, the server MUST answer with a Q4S 200 OK message, and in the body of the answer message, an SDP document MUST be included (RFC 4566 [2]), with information about the required quality constraints. Q4S responses should use the protocol designator “Q4S/1.0”.

After these two messages are exchanged, the first phase is completed. The quality parameters have been sent to the client. Next step is to measure the quality of the communication path between the client and the server and alert if the SLA is being violated.

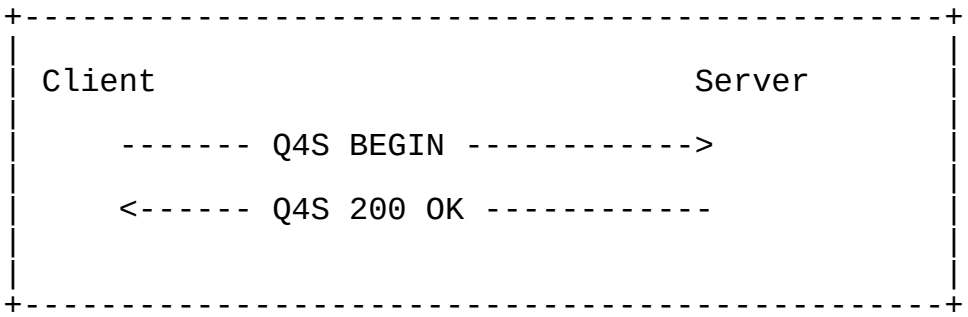

Figure 7 Handshake phase.

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
Q4S-Resource-Server: \
q4s://www.example.com/example/util/agent?num=666
Q4S-Policy-Server: q4s://www.qosmanager.com/agent
Signature: 6ec1ba40e2adf2d783de530ae254acd4f3477ac4
Content-Length: 131

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

The “Expires” header purpose is to provide a sanity check and enables the server to close inactive sessions. If the client does not send a new request before the expiration time, the server can close the session.

The “Signature” header contains a digital signature that can be used by the network to validate the SDP, preventing security attacks.

The signature is an optional header generated by the server using a hash and encryption method such as MD5 (RFC 1321 [6]) and RSA (RFC 2437 [7]), but it depends on the certificate used by the server. 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.

The optional response header “Q4S-Resource-Server” contains the Session URI, which is in charge of this session. This URI MUST be invoked by the client in all later requests. Example:

```
Q4S-Resource-server: \
q4s://www.example.com/example/util/agent?num=666
```

If this header is not present, the client will continue sending all requests to the original Contact URI, but if it is present, its use is mandatory.

The last optional response header is “Q4S-Policy-Server” which contains the “Policy Server URI” towards which client MUST send the later QOS-ALERT messages. If the “Q4S-Policy-Server” header is present, the "Q4S-Resource-server" header is mandatory, as the policy server MUST notify the Q4S server about the Q4S-ALERT received from the client and this information in not available to the policy server except through the "Q4S-Resource-server" header.

During the next phases of the protocol, the client role will perform a mix of client and server role. Hence, the client can specify a “Q4S-Resource-Client” header in the BEGIN request, indicating the Resource Client URI, a relative URI in charge of the server requests when client receives requests from the server. Example:

```
Q4S-Resource-Client: /example/useragent
```

This URI MUST be relative because user agents may not have an associated domain, or its IP address is unknown.

4.5. Negotiation phase

The negotiation phase is in charge of measuring the quality parameters and verifying that the communication paths meet the required quality constraints. If the quality session is compliant with the minimum quality constraints the application can start. If not, a higher quality service level will be demanded and the results on the network parameters will be measured again. If after some time the quality constraints cannot be met the quality session is terminated due.

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:

- Stage 0: Measurement of latencies, jitters and packet loss
- Stage 1: Measurement of bandwidths and packet loss

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

The client starts the negotiation phase sending a READY request using the TCP control 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.

4.5.1. Stage 0: Measurement of latencies and jitters

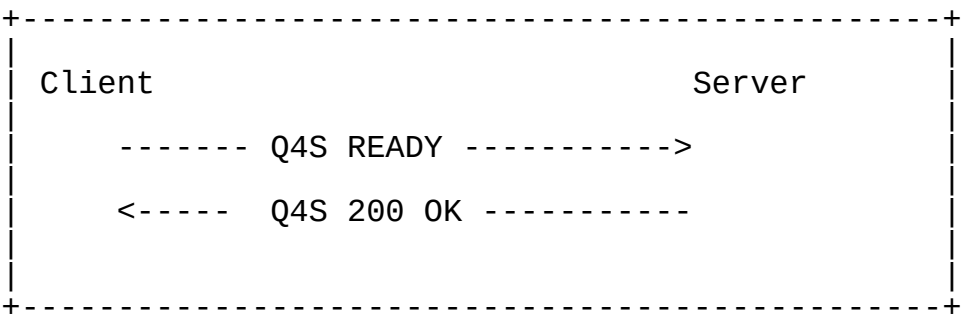

Figure 8 Beginning of Stage 0 of the Negotiation phase.

The Stage 0 MUST start with a synchronization message exchange initiated with the client's READY message. This allows the synchronization of negotiation phases in multiple quality sessions enabling the possibility to repeat a successful stage.

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 as described in section 4.3.

The client MUST send its PING requests using the UDP control flow ports defined in the SDP negotiated during the handshake phase. The downlink port is set as destination and the uplink port is set as origin (according to the example given in the SDP structure, from

client UDP port 56000 to server UDP port 55000). The Sequence-Number header MUST be set to 0.

At the same time the server must begin to do exactly the same, using the downlink UDP control ports to send PING requests towards the client.

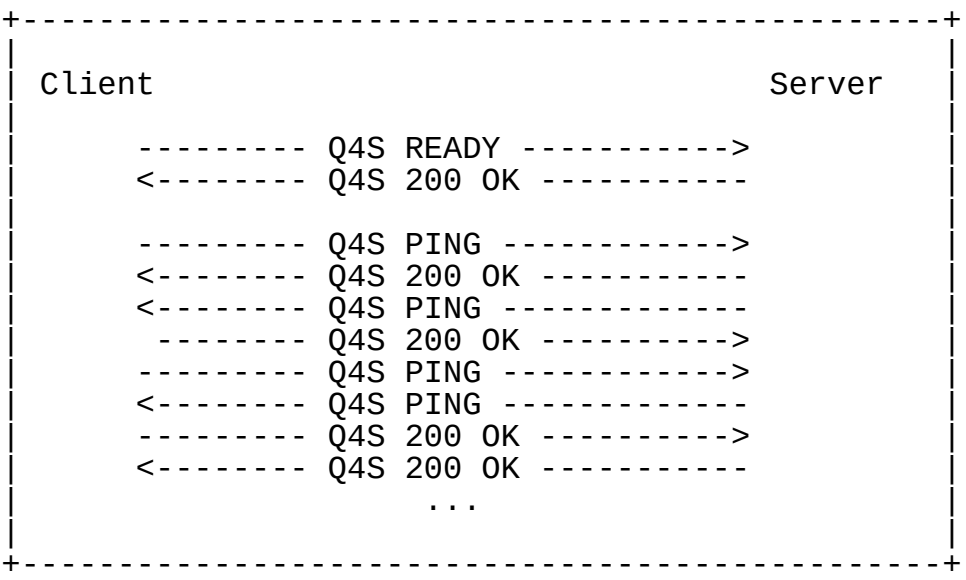

Figure 9 Simultaneous exchange of PING request and responses.

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
       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 control ports and incrementing the Sequence-Number with each message and MUST NOT wait for a response to send the next PING request. The “Sequence-Number” header value is a sequential integer number and MUST start at zero. If this stage is repeated, the initial Sequence-Number MUST start again at zero.

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 “Measurements” header, with the values of the jitter and packet loss measured by each entity. The client will send its measurements to the server and the server his measurements to the client. Example : Measurements: 13, 1

- a “Qualimeter” header, with the value in percentage of experienced versus desired quality.

In 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,50/60,50/50,5000,0

A minimum of 256 PING messages MUST be exchanged in order to be able to measure latency, jitter and packet-loss. Both the client and the server calculate the respective measured parameter values. The mechanisms to calculate the different parameters are described in section 4.3.

Then the client MUST send a GET request to the server using the TCP uplink control port exchanged in the handshake phase in order to communicate the measured parameters to the server. This message MUST always be sent, independently of the measurement procedure used. The GET request contains a body with updated downlink values for the latency, jitter and packet loss measurement attributes.

An example of a GET request can be seen below.

```
Client Request:
=========================
GET q4s://www.example.com Q4S/1.0
Host: www.example.com
User-Agent: q4s-ua-experimental-1.0
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:0/0
a=public-address:client IP4 192.0.2.33
a=public-address:server IP4 198.51.100.58
a=latency:40/35
a=jitter:10/10
a=bandwidth:20/6000
a=packetloss:5/5
a=flow:data downlink TCP/10000-20000
a=flow:control downlink UDP/55000
a=flow:control downlink TCP/55001
a=flow:data uplink TCP/56000
a=flow:control uplink UDP/56000
a=flow:control uplink TCP/56001
a=measurement:procedure default,50/50,75/75,5000,0
a=measurement:latency 40/40
a=measurement:jitter 0/10
a=measurement:bandwidth 0/0
a=measurement:packetloss 0/2
=========================
```

When the server receives this message, it compares the latency value (RTT/2) with its own measurements, in order to avoid inconsistencies and giving priority to the latency value measured by server.

At this point there are two possibilities

- The latency, jitter and packet loss constraints are reached
- The latency, jitter and packet loss constraints are not reached

In the first case, the server verifies that all three parameters are within acceptable values and then MUST answer with an

acknowledgement, a QOS 200 OK message. This message contains an SDP body with the server's measured data.

```
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)
=========================
```

It means that the client and the server have finalized Stage 0 and are ready for Stage 1, bandwidth and packet loss measurement.

If the bandwidth constraints are empty or with value zero, the negotiation phase MUST terminate and both client and server initiate the Continuity Phase.

The second case, in which some constraint has not been met is detailed in section 4.5.3. Constraints not reached.

4.5.2. Stage 1: Measurement of bandwidth and packet loss

This stage begins in a similar way to stage 0, sending a READY request over TCP control ports. This READY message "Stage" header value is 1. The server answers with a Q4S 200 OK message to synchronize the initiation of the measurements.

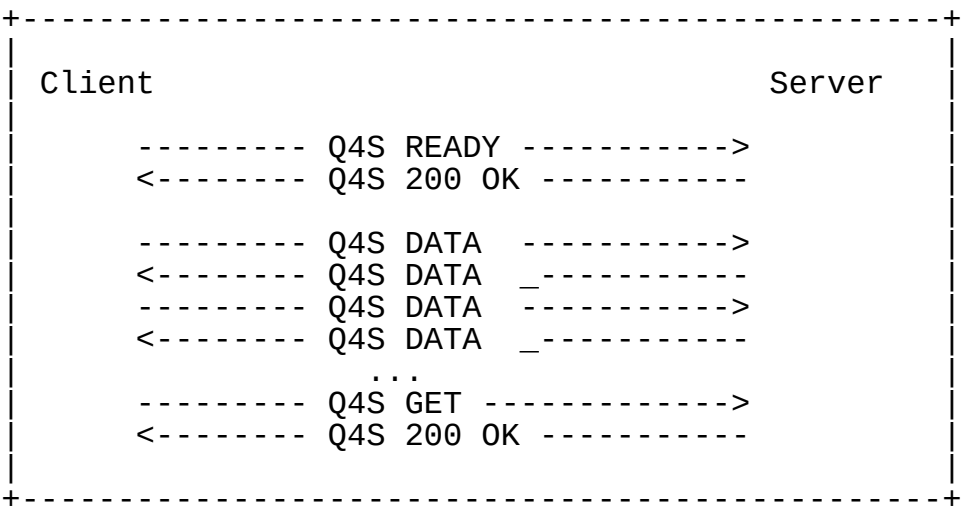

Figure 10 Starting bandwidth and packet loss measurement

```
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 DATA messages simultaneously using the UDP control ports. Section 4.3.3 describes the bandwidth measurement in detail.

After the measurements have been performed the client MUST send a GET message to the server using the uplink TCP control port including in the body of the message the SDP data with the parameter measurement attributes filling the downlink fields of the bandwidth and packet loss.

```
Client Request:
=========================
GET 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:1/1
a=public-address:client IP4 192.0.2.33
a=public-address:server IP4 198.51.100.58
a=latency:40/35
a=jitter:10/10
a=bandwidth:20/6000
a=packetloss:5/5
a=flow:data downlink TCP/10000-20000
a=flow:control downlink UDP/55000
a=flow:control downlink TCP/55001
a=flow:data uplink TCP/56000
a=flow:control uplink UDP/56000
a=flow:control uplink TCP/56001
a=measurement:procedure default,50/50,50/50,5000,0
a=measurement:latency 30/30
a=measurement:jitter 6/4
a=measurement:bandwidth 0/4000
a=measurement:packetloss 0/3
============================
```

At this point there are two possibilities:

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

The second case, with violated constraints is explained in 4.6.3 Constraints not reached.

If measurements match the constraints, the negotiation phase is successful, the client and server have verified that all constraints are met and the application can be started. An optional simple mechanism, based on HTTP, is defined to trigger the application using the “Trigger-URI” header.

The server answer MUST be 200 OK, and MAY include the URI for triggering the application using an optional “Trigger-URI” header.

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

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

The application SHOULD be triggered using an URI, by means of an HTTP request, specified in the Q4S header “Trigger-URI”. Other mechanisms, such as including a “Location” header in the Q4S message, to force redirection is not recommended because these mechanisms are achieved without parsing the body of the message.

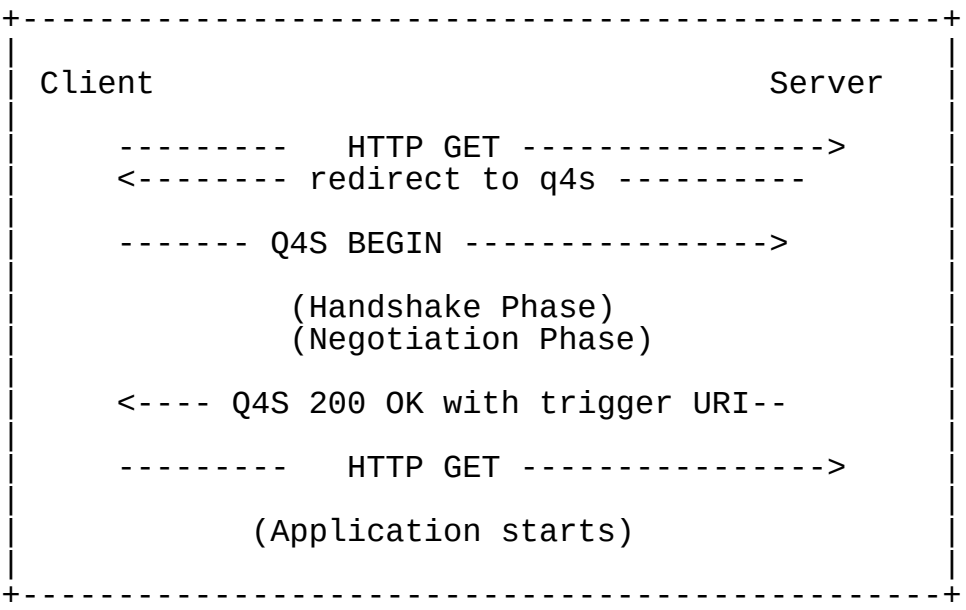

Figure 11 Trigger the application using HTTP URI

Figure 12 shows a usage example; an integration of HTTP and Q4S is shown. First, the client contacts the server using HTTP, a redirection to a Q4S URI is achieved and the User Agent starts the Q4S handshake phase. After negotiation phase succeeds, the client trigger the application using the URI indicated in the Q4S 200 OK message.

4.5.3. Constraints not reached

After a measurement period the client sends the measured parameters in the SDP body of a GET request to the server.

If there is any parameter that does not comply with the uplink or downlink quality constraints required, the server MUST answer the client's GET request with a 412 message (a precondition setting required by the client or server has not been met) indicating in the method type the parameter or parameters that violate the constraints. This message MUST contain an SDP body with all data, the client's and the server's parameter measurements.

The 412 message MUST include two additional headers: Cause and Signature headers. The Cause: header includes information about the direction and the parameter that did not meet the constraints. The Signature header contains the signed hash value of the SDP body in order to protect all the SDP the data.

Server’s 412 Answer:

```
========================
Q4S/1.0 412 downlink_bandwidth
Date: Mon, 10 Jun 2010 10:00:01 GMT
Content-Type: application/sdp
Expires: 3000
Cause:downlink_bandwidth
Signature: 6ec1ba40e2adf2d783de530ae254acd4f3477ac4
Content-Length: 131

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=public-address:client IP4 192.0.2.33
a=public-address:server IP4 198.51.100.58
a=latency:40/35
a=jitter:10/10
a=bandwidth:20/6000
a=packetloss:5/5
a=flow:data downlink TCP/10000-20000
a=flow:control downlink UDP/55000
a=flow:control downlink TCP/55001
a=flow:data uplink TCP/56000
a=flow:control uplink UDP/56000
a=flow:control uplink TCP/56001
a=measurement:procedure default,50/50,50/50,5000,0
a=measurement:latency 30/30
a=measurement:jitter 6/4
a=measurement:bandwidth 200/4000
a=measurement:packetloss 2/3
========================
```

In the body of the 412 message, the server MAY also rise the “qos-level” SDP session-level attribute of the affected direction (uplink or downlink). The maximum qos-level allowed is 9, both uplink and downlink. If this value is reached without meeting the constraints,

the Q4S protocol initiates the Termination phase, quality session is aborted using the CANCEL method.

After a Q4S 412 message the client MUST send a QOS-ALERT request to acknowledge the SLA violation (using TCP control port). Notice that the server's signature header is present in the client QOS-ALERT, in order to allow integrity validation. An example of a QOS-ALERT message follows.

```
Client Request:
=========================
QOS-ALERT q4s://www.example.com Q4S/1.0
Host: www.example.com
User-Agent: q4s-ua-experimental-1.0
Signature: 6ec1ba40e2adf2d783de530ae254acd4f3477ac4
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=public-address:client IP4 192.0.2.33
a=public-address:server IP4 198.51.100.58
a=latency:40/35
a=jitter:10/10
a=bandwidth:20/6000
a=packetloss:5/5
a=flow:data downlink TCP/10000-20000
a=flow:control downlink UDP/55000
a=flow:control downlink TCP/55001
a=flow:data uplink TCP/56000
a=flow:control uplink UDP/56000
a=flow:control uplink TCP/56001
a=measurement:procedure default,50/50,50/50,5000,0
a=measurement:latency 30/30
a=measurement:jitter 6/4
a=measurement:bandwidth 200/4000
a=measurement:packetloss 2/3
=========================
```

If during the handshake phase the optional header Q4S-policy-server is included in the server response, the QOS-ALERT message MUST be sent to the policy server, otherwise, the client will send this

message to the server directly. The scenario with an existing policy server will be explained in 4.6.3.1.

Upon receiving the QOS-ALERT request from the client, the server will acknowledge the alert issuing another QOS-ALERT request towards the client. This message MUST include a new header: “Guard-time” shown in the example below.

```
Server Answer:
=========================
QOS-ALERT q4s://www.example.com Q4S/1.0
Date: Mon, 10 Jun 2010 10:00:01 GMT
Content-Type: application/sdp
Expires: 3000
Cause: latency
Guard-time: 5000
Signature: 6ec1ba40e2adf2d783de530ae254acd4f3477ac4
Content-Length: 131

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=public-address:client IP4 192.0.2.33
a=public-address:server IP4 198.51.100.58
a=latency:40/35
a=jitter:10/10
a=bandwidth:20/6000
a=packetloss:5/5
a=flow:data downlink TCP/10000-20000
a=flow:control downlink UDP/55000
a=flow:control downlink TCP/55001
a=flow:data uplink TCP/56000
a=flow:control uplink UDP/56000
a=flow:control uplink TCP/56001
a=measurement:procedure default,50/50,50/50,5000,0
a=measurement:latency 30/30
a=measurement:jitter 6/4
a=measurement:bandwidth 200/4000
a=measurement:packetloss 2/3
=========================
```

At this point, both client and server wait for “Guard-Time” seconds as specified by the header before attempting to re-initiate the measurements that caused the alert. This time SHOULD be set such that enough time has been given to allow the server to take any actions notifying the application or allowing for the network to recover. (5 seconds should be enough, but this depends on each case). After “Guard-Time” seconds the measurement process MUST be initiated by the client sending a READY request.

If the client does not obey the “Guard-time”, and sends a READY message before this time elapses, the server MUST wait and not answer the READY message until the guard time has elapsed.

If during the measurement process some interference disturbs or affects the measurement results, it is better to repeat the process again rather than alerting of an SLA violation. This is always possible by sending current values of parameter “qos-level” without changes, and in this case a header Guard-time can be set to “0”. It is a good practice to repeat the measurements before reporting a violation.

An example of a message exchange with several guard-times is shown below.

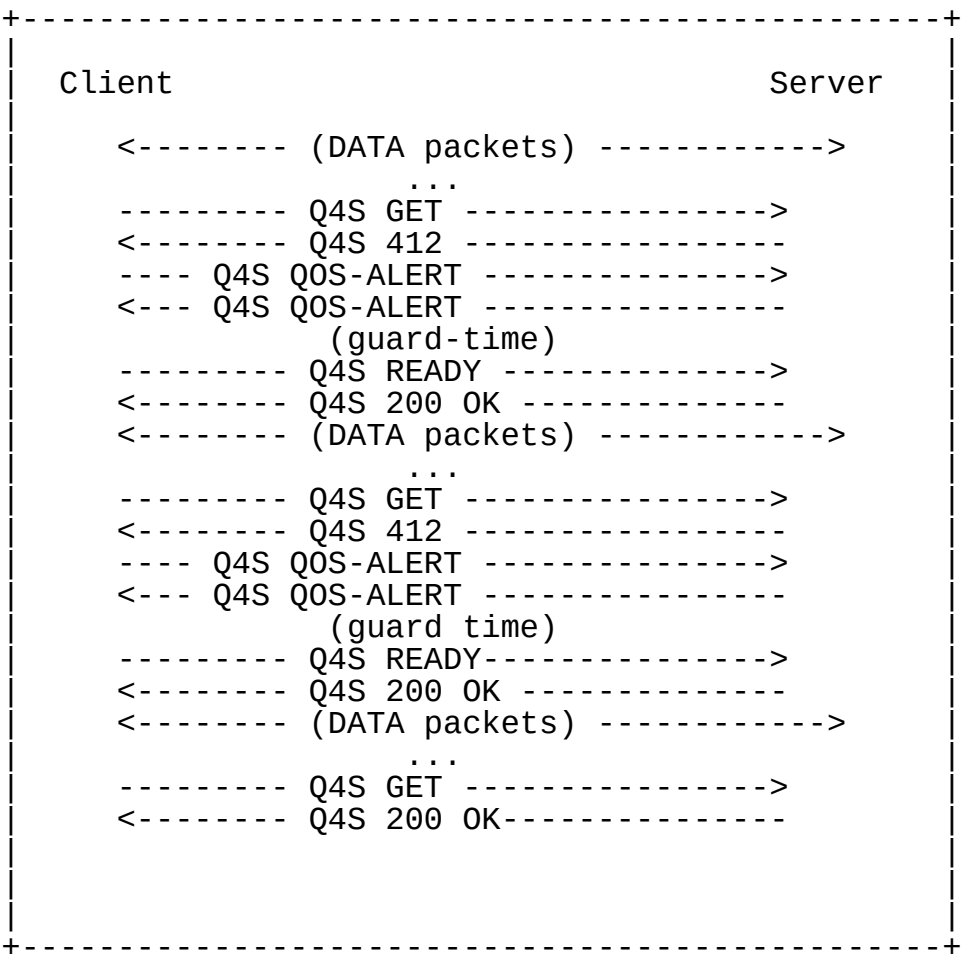

Figure 12 Several measurements with alerts and final success.

4.5.3.1. Policy server is present

If during handshake phase the optional header Q4S-Policy-Server is included in the server response, the QOS-ALERT request MUST be sent to the policy server, which can implement all or some of these features (but not exclusive to):

- Client and server validation in terms of SLA.
- Authentication (Signature validation) and security (block malicious clients)

- Policy rules (following rules are only examples):
  - Maximum quality level allowed for the ACP
  - Time bands allowed for provide quality sessions for the ACP
  - Number of simultaneous quality sessions allowed
  - Maximum time used by quality sessions allowed
  - Etc.

With policy server, the QOS-ALERT message sent by the client MUST contain the URIs of the server and the client to be contacted later by the policy server. Therefore the following headers MUST be included in the client request: “Q4S-Resource-server” and “Q4S-Resource-client”

Depending on the results of the operations achieved by the policy server, the client could receive different types of errors or CANCEL messages.

The flows of messages in this case are in the following figure:

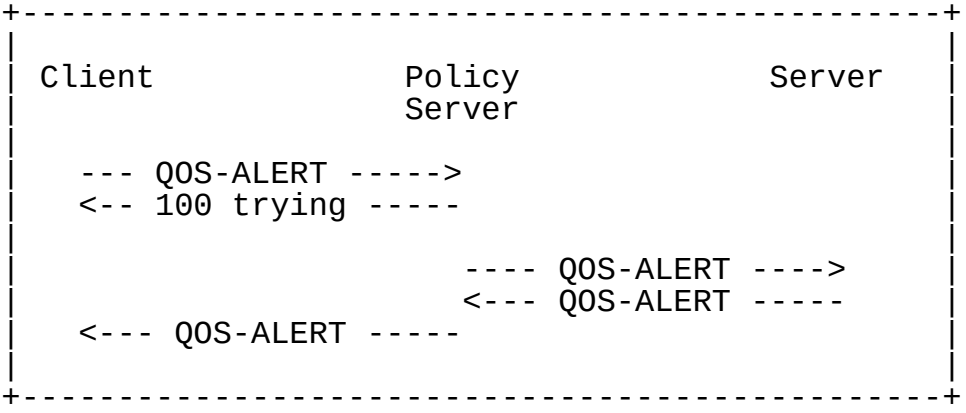

Figure 13 Policy server.

If the validation or authentication of the QOS-ALERT operation fails, the policy server will send a CANCEL request to the client without contacting the server.

If any of the policy rules fail, the server will send a 6XX error to the client, indicating the rule that is not satisfied.

Only if the validation, authentication and policy checking are successful, the server is contacted by the policy server and the QOS-ALERT message is forwarded to it.

4.5.4. QoS Level changes

If any constraint was violated, the server may raise by one the qos-level attribute of the 412 message sent to the client. 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 receipt 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
Host: www.example.com
User-Agent: q4s-ua-experimental-1.0
Signature: 6ec1ba40e2adf2d783de530ae254acd4f3477ac4
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
Date: Mon, 10 Jun 2010 10:00:01 GMT
Expires: 0
Content-Type: application/sdp
Signature: 6ec1ba40e2adf2d783de530ae254acd4f3477ac4
Content-Length: 131

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

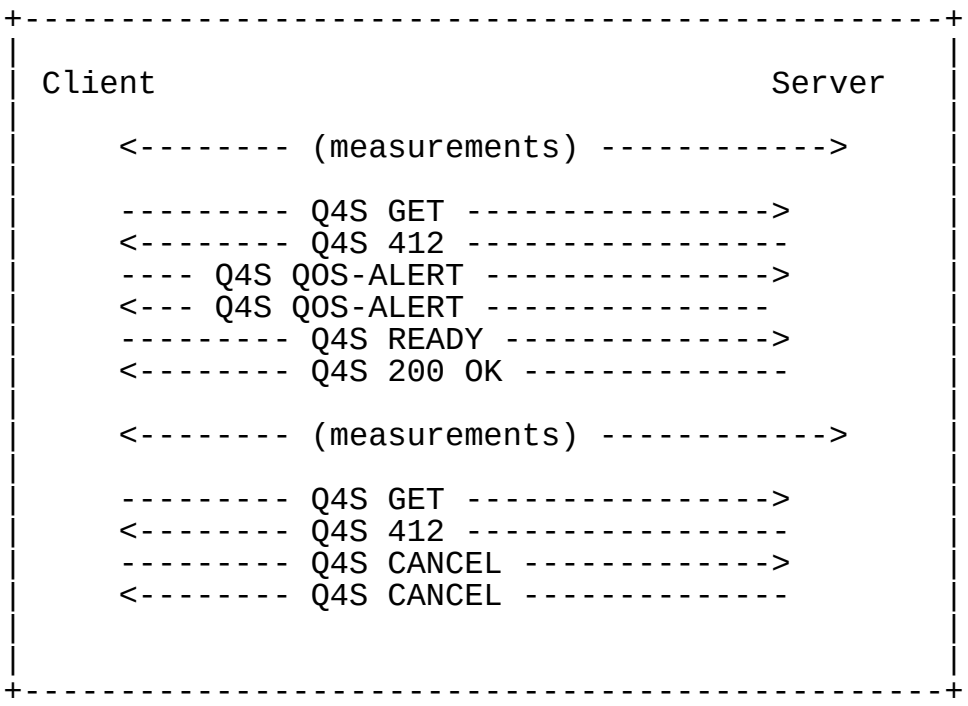

Figure 14 Failed negotiation phase.

4.5.4.1. QoS Level increments without changes in network behaviour

If the qos-level has not reached the maximum value (9) but after 3 QOS-ALERT messages (with increments in qos-level) the network remains with the same quality values, the client and the server MUST assume that the network can not reach the desired quality and abort the session in order to save resources (time and traffic). To do that, the client MUST send a CANCEL request and the server MUST react to it sending a CANCEL request too.

If the client does not send a CANCEL request but a request using a different method, the server MUST react to it sending a CANCEL request.

4.6. Continuity phase

During the negotiation phase, latency, jitter, bandwidth and packet loss have been measured. During 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.

In the default measurement procedure, two working modes are defined for this phase: normal and sliding window modes. This draft only covers the default procedure.

4.6.1.1. Normal mode

The server can force the use of normal mode by setting the fourth parameter of “procedure” SDP attribute to 0. If this parameter is not set, the default value is assumed (zero), and normal mode will be used.

Example:

```
a=measurement:procedure default,50/50,50/50,5000,0
```

Considering that network conditions can change, the client may periodically check network conditions against negotiated constraints. The maximum interval expected between network testing is indicated in the Q4S Expires header.

However, the measurements can be carried out periodically in a smaller period of time than “Expires” header value. Intense interactive applications, like arcade videogames, the period to repeat the measurements may be very small (even zero), in order to measure continuously the quality and assure the best reaction time. To reach the best reaction time, the use of the sliding window mode is recommended.

To start the continuity phase, the client sends a Q4S READY method, using the TCP control port specified in the handshake phase, indicating the new Stage header value for continuity phase (value 2).

```
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 Response:
========================
  Q4S/1.0 200 OK
       Session-Id: 53655765
       Stage: 2
       Content-Length: 0
========================
```

After these messages are exchanged, latency, jitter and packet loss measurement are started. The measurement procedure is identical to the one carried out in the negotiation phase and is explained in the Measurements section 3.2, except for the time elapsed between salmples. If the default measurement method is being used, it is recommended to use a larger interval for PING messages than the one used in the negotiation phase, but the same number of samples will be taken to check quality. The goal of incrementing the interval of PING messages is to minimize the load of the server, which would be running lots of connections in parallel.

The interval used for this phase is indicated in the second parameter of the attribute line for the procedure. In this example, the interval is 75 milliseconds.

```
a=measurement:procedure default,50/50,75/75,5000,0
```

A value larger than the one used in the negotiation phase is recommended.

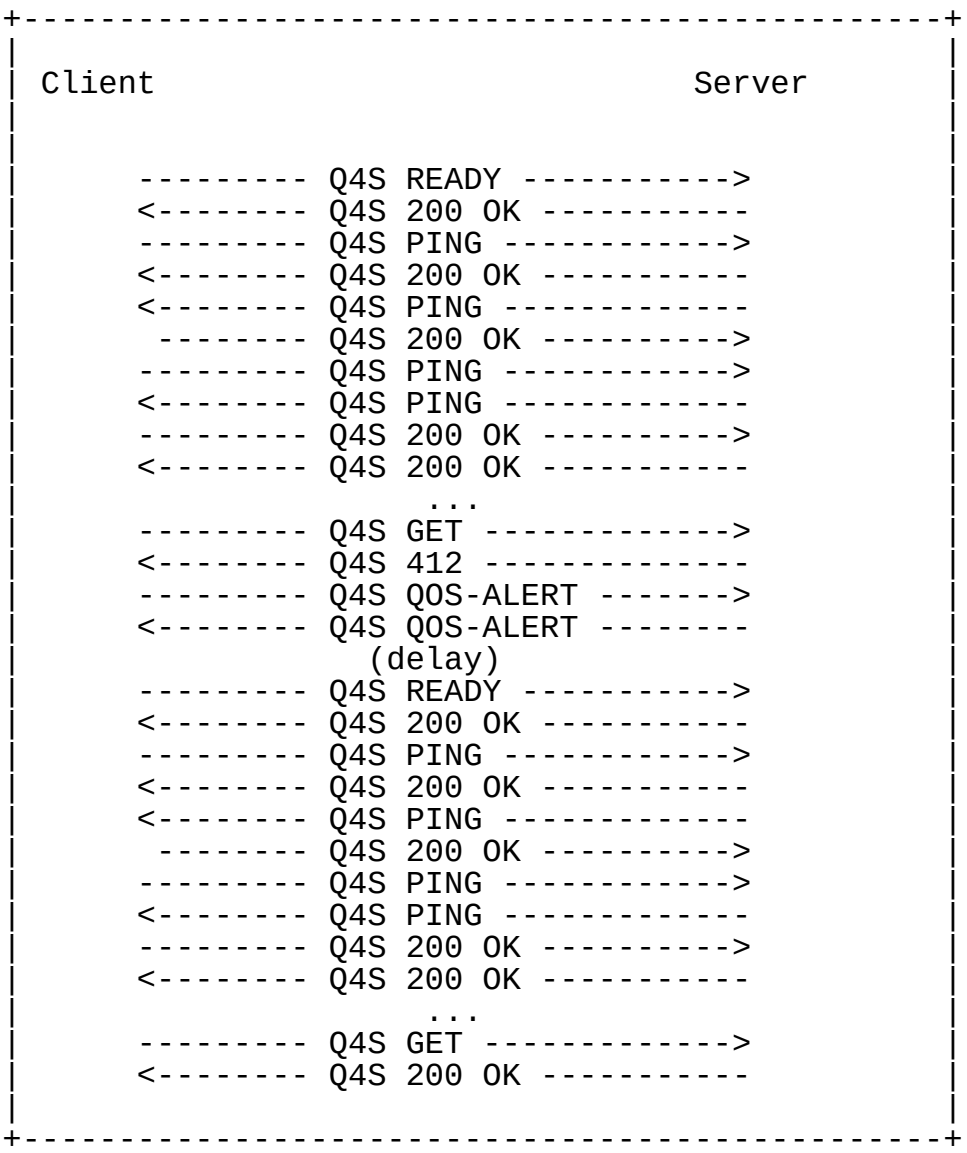

Figure 15 Continuity.

4.6.1.2. Sliding window mode

In order to improve the reaction time when network conditions degrade quickly, the server can force the use of the sliding window mode by setting the fourth parameter of the “procedure” SDP attribute to 1.

Example:

a=measurement:procedure default,50/50,50/50,5000,1

The sliding window mode applies a sliding window of samples instead cycles of samples. The number of samples involved in the sliding window may be different for jitter and latency calculations than for packet-loss calculations according to the fifth and sixth parameters of the measurement 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,50/50,75/75,5000,1,40/40,100/100

In addition, the sizes of these windows are configurable per direction.

In the sliding window mode, PING requests are sent continuously (in both directions) and when the Sequence-Number header reaches the value of 255, the client MUST NOT send a GET message for instructions, but continues sending PING messages with the Sequence-Number header starting again at zero. When the server PING Sequence-Number header reaches 255, 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.

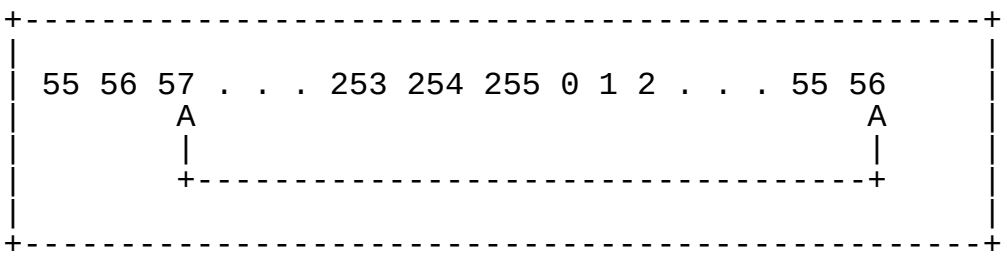

Figure 16 Sliding samples window

Only when the client detects that the measured values (downlink jitter, downlink packet loss and latency) are not reaching the constraints, a GET request is sent to the server.

When the server receives the GET request, it stops sending PING requests and answers the GET request just received. If the response code is 412, then a QOS-ALERT will be requested by the client, exactly in the same way as described in normal mode.

On the other hand, if the server detects that the measured values (uplink jitter, uplink packet loss and latency) are not reaching the constraints, it MUST choose between the following alternatives:

- The server stops sending PING request to the client. In this case the client MUST notice this lack of PING requests using a timeout (Expire header) at reception. If so, the client reacts stopping the PING requests to the server and sending a GET request for instructions, exactly in the same way as described in normal mode.

- It continues sending PING requests but all of them with Sequence-Number set to -1 till a client GET request is received. Then the server stops sending PING messages and answers the GET request with the corresponding 412 error, exactly in the same way as described in normal mode. The client reacts sending this GET request when it receives a PING request with Sequence-Number header set to -1. This behavior allows the shortest reaction time under degraded network conditions.

Both alternatives MUST be implemented by the Q4S client.

In Sliding-window mode, the optional header "measurements" of the PING message may be quite useful, because it allows the server to obtain all measurements without needing to receive a GET message from client. In this mode, the server may raise alerts directly, independently of which direction didn’t meet a quality constraint, and send them to a policy server without client intervention.

To avoid further alerts from the client, the SDP message sent by server to client may have high thresholds. The following diagram represents this scenario

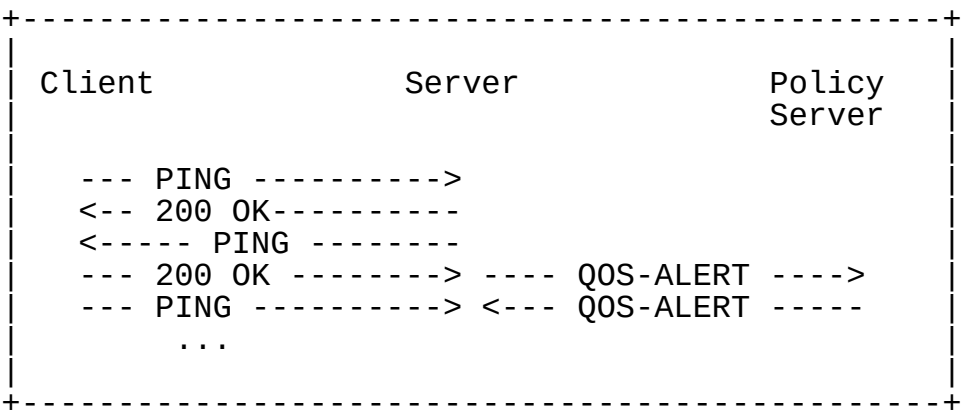

Figure 17 Direct Alert sending by the server

4.7. Termination Phase

The Termination phase is not a phase itself but an end point for the established Q4S session. This phase 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 application has stopped running and alerts the server about its termination so that the server can terminate the Q4S session.
- 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.

4.8. Dynamic constraints and flows

Depending on the nature of the application, the constraints to be reached may evolve, changing some or all 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 reply (200 OK, or 412, etc), the client MUST assume all the new received values, overriding any previous value in use.

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

- The application communicates to the Q4S a change in the constraints. In this case the application requirements can evolve and the Q4S server will be aware of them.

- The application uses TCP flows. In that case, in order to guarantee a constant throughput, the nature of TCP behavior forces the use of a composite constraint function, which depends on RTT, packet loss and 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 [14], High Speed TCP (RFC 3649 [15]), CUBIC [16], Compound TCP (CTCP [17]), 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 viceversa (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= Function(RTT, packet loss, ...)

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 [15] for the upper bound on the throughput:

Th <= (MSS/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.

4.9. Qos-level downgrade operation

During the continuity phase it might be desirable to downgrade the current qos-level SDP parameter.

The strategy to carry out downgrades must include the possibility to exclude specific data flows from SDP dynamically. A Q4S client MUST allow this kind of SDP modifications by server.

Periodically (every several minutes, depending on the implementation) the server could force a QOS-ALERT, in which the level is downgraded for control flows, excluding application data flows from the embedded SDP of that request. To set the new SDP, the server MUST include the modified SDP in the 412 error message.

This mechanism allows to measure at lower levels of quality while application flows continue using a higher qos level value.

- If the measurements in the lower level meet the constraints, then a new QOS-ALERT to this lower qos-level can be forced by the server, in which the SDP includes the application data flows in addition to control flows.

- If the measurements in the lower level do not meet the constraints, then a new QOS-ALERT to the previous qos-level MUST be forced by the server, in which the SDP includes only the control flows.

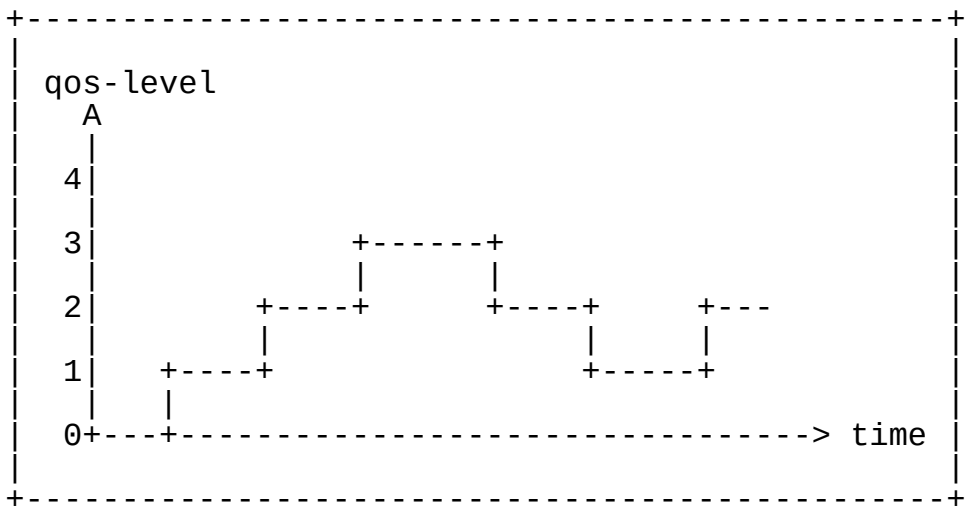

Figure 18 Possible evolution of qos-level

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

The chosen period to measure a lower qos level is implementation dependant. Therefore it is not included as a measurement procedure parameter. It is recommended to use a large value, such as 20 minutes.

4.10. Sanity check of Quality sessions

A session may finish by 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 that a client can wait to repeat the continuity phase (in normal mode).

5. 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.

5.1. 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 five methods: GET for getting information and sending quality reports, PING and DATA for quality measurements purpose, CANCEL for terminating sessions, and QOS-ALERT for querying ISPs for quality upgrades.

Request-URI: The Request-URI is a Q4S URI (RFC 2396) as described in 2.2.1 It Normally indicates the user or service to which this request is being addressed to, but in the Q4S context, there are some methods whose URI only reflects the service on the server side, but nothing more. This is the case of the QOS-ALERT method, because the real address of a QoS upgrade

request is the network, and therefore in this case the URI only reflects the server address. In addition the CANCEL method has the same treatment, and in the ECHO and DATA methods invoked by the server to the client the meaning of the URI is only the URI of the service, but not the destination of the request. 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.

5.2. 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 and CANCEL requests do not have to be responded.

5.3. 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.

5.3.1. Specific Q4S Request Header Fields

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

- Session-Id: the value for this header is the same session id used in SDP 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.

- Sequence-Number: sequential integer number assigned to PING and DATA messages.

- Timestamp: this optional header contains the system time (with the best possible accuracy). Indicates the time in which the request was sent.

- Signature: this header contains a digital signature that can be used by the network to validate the SDP. The signature is always generated by the server. It is optional.

- Q4S-Resource-Client: this optional header contains the relative URI in charge of this session at the client side. In The case of being included, it MUST appear in the GET request of handshake phase. This URI MUST be invoked by the server in all later requests. It is optional, but it should be present, it becomes mandatory for the counterpart. This URI MUST be relative because user agents can not have associated domain, in addition to ignore their public IP address.

5.3.2. Specific Q4S Response Header Fields

- Expires: the 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 can close the session. The value MUST be an integer and the measurement unit are milliseconds.

- Guard-time: A time interval in milliseconds left vacant (i.e., during which no data is sent) during the quality session. The guard time provides a safety margin before re-starting each measurement process when a QOS-ALERT has been raised. This header is optional in all messages but mandatory in the QOS-ALERT sent by the server.

- Sequence-Number: same meaning as Request Header Fields

- Timestamp: UTC time in nanoseconds. Indicates the time in which the request was sent. If the server (or a client) receives a Timestamp header in a request, MUST include the same header with the same value in the response. The purpose of this header is simplify the RTT calculation.

- Signature: same meaning as Request Header Fields

- Q4S-Resource-Server: this optional header contains the URI in charge of this session (Session URI). In case of being included, it MUST appear in the response to the BEGIN request of the handshake phase. This URI MUST be invoked by the client in all later requests. It is optional, but if present, it becomes mandatory for the counterpart.

- Q4S-Policy-Server: this optional header contains the URI towards which the client and MUST send the QOS-ALERT messages (Policy Server URI). In case this header is present, the Q4S-Resource-Server header is mandatory, and MUST be included in the QOS-ALERT messages sent by the client to the policy server. In addition, the QOS-ALERT sent to the policy server MUST contain the header Q4S-Resource-client

- Cause: This header field is a comma-separated list which contains the cause(s) for which the connection constraints were not reached after measurement process. Current defined values are:

    - Downlink_latency
    - Uplink_latency
    - Downlink_jitter
    - Uplink_jitter
    - Downlink_bw
    - Uplink_bw

5.4. 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.

5.4.1. Encoding

The body MUST NOT be compressed. This mechanism is valid for other protocols such as HTTP and SIP (RFC 3261 [13]), but a compression/coding scheme will limit certain logical implementations of the way the request is parsed, thus, making the protocol concept more implementation dependant. In addition, bandwidth calculation may not be valid if compression is used. Therefore, the HTTP request header “Accept-Encoding” can not 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 is 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.)

6. General User Agent behavior.

6.1. Roles

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).

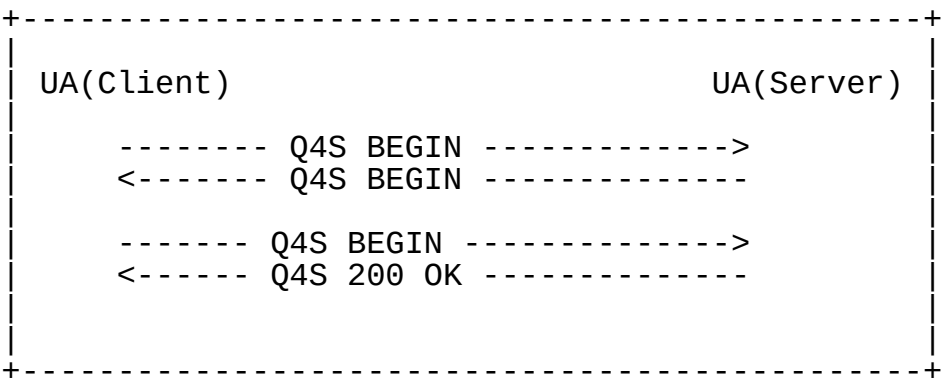

Figure 19 P2P roles.

6.2. Multiple Quality sessions in parallel

A quality session is intended to be used for a single application (or application instance). It means that for using the application, the client MUST establish only one quality session against the server. Indeed, the relation between Session-Id and application is 1 to 1.

If a user wants to raise several independent quality sessions simultaneously against different servers (or against the same server) it can execute multiple Q4S clients to establish separate quality sessions. However, this is not recommended because:

- The establishment of a new quality session may affect other running applications over other quality sessions. Thus, minimum quality level may not be achieved depending on individual requirements of each application.
- If the negotiation phase is executed separately before running any application, the quality requirements could not be assured when the applications are running in parallel.
- Flow identification (Protocol, SourceIP, Source Port + Destination IP, Destination Port) must always be unique for each application/application instance, to ensure that each one of them is using their QoS constraints.

For running different applications in parallel it is highly recommended to execute the negotiation phase of all of them simultaneously, in order to assure the quality constraints of all applications in parallel. To do that, a single User Agent MUST be used, and this User Agent MUST be able to launch several quality

session negotiations in parallel, synchronizing the beginning of each negotiation phase, and running again the negotiation phase of all applications in parallel until all of them succeed.

In order to repeat the execution of a negotiation phase that has been succeeded, both, client and server MUST allow using the READY method with a Stage header value already succeeded.

6.3. 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 GET, QOS-ALERT 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 DATA), 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 assumes the role of a server.

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

The QOS-ALERT and CANCEL do not need to be answered. However, these methods may have a conventional answer if an error is produced.

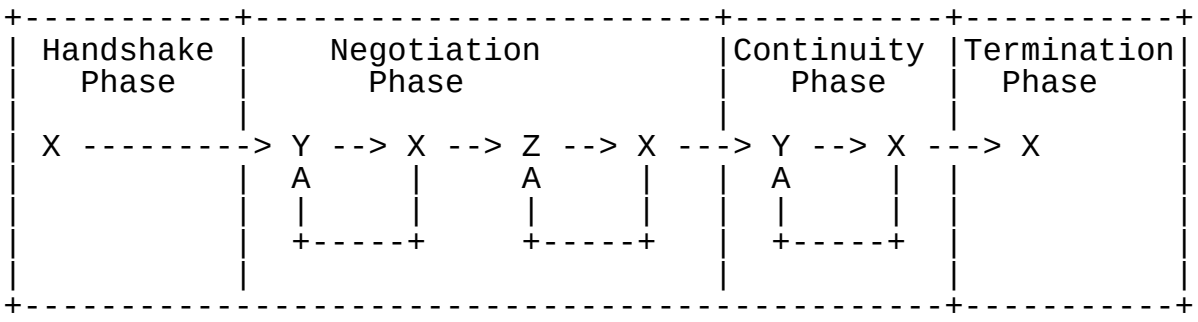

Figure 20 Phases & client behaviors.

6.3.1. Generating requests

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

If no SDP is included: This is the case of PING and DATA messages. The header Session-Id and Sequence-Number are mandatory.

If SDP is included: this is the case of GET, QOS-ALERT and CANCEL messages. Inside SDP is included Session-Id, therefore the inclusion of Session-Id header is optional.

6.4. 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 GET, QOS-ALERT 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 DATA), 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 as a client, sending PING requests toward the client and receiving responses.

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

The QOS-ALERT and CANCEL do not need to be answered. However, these methods may have a conventional answer if an error is produced.

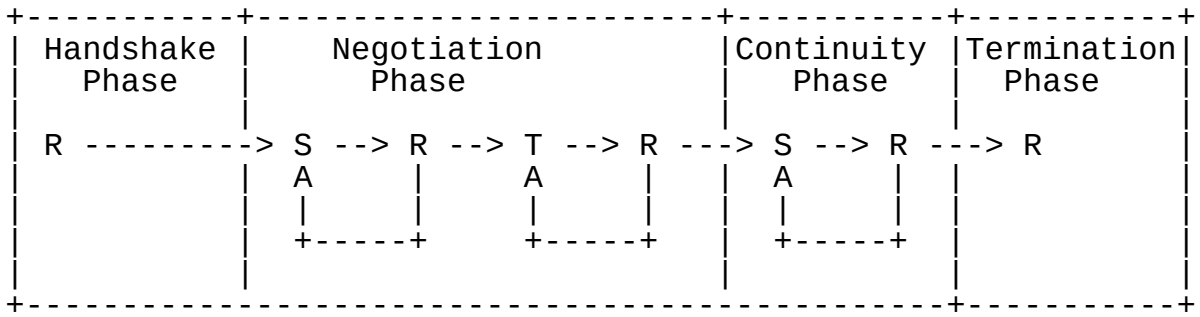

Figure 21 Phases & server behaviours.

7. 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” | “GET” | “READY” | “PING” | “DATA” |
          “QOS-ALERT” | “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.

7.1. BEGIN

The BEGIN method means request 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 all quality parameters for the desired application to run.

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

The response to a Q4S BEGIN request is not cacheable.

7.2. GET

The GET method means retrieve information from a resource identified by a Q4S URI.

In the negotiation and continuity phases, this method is used to check if the server considers the quality good enough to execute the desired application. If the measured quality is not enough, the server will return a 412 error.

The response to a Q4S GET request is not cacheable.

7.3. READY

The READY method is used to synchronize the starting time for sending of PING and DATA messages over UDP between clients and servers.

In addition, the Stage header included in this method is mandatory and allows clients to repeat a test, which is needed in scenarios with multiple quality sessions between one client and different servers.

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.

7.4. 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 control port. If a server receives this message in another port it MUST ignore it.

The fundamental difference between the PING and DATA requests is reflected in the different measurements achieved with them. PING is a short message, and MUST be answered in order to measure RTT, whereas DATA is a long message (1 Kbyte) and MUST NOT be answered.

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

7.5. DATA

This message is used only during the continuity phase to measure the bandwidth and packet loss of a session. The message MUST be sent only over UDP control port. If a server receives this message in other port it MUST ignore it.

The fundamental difference between the PING and DATA requests is reflected in the different measurements achieved with them. PING is a short message, and MUST be answered in order to measure RTT, whereas DATA is a long message (1 Kbyte) and MUST NOT be answered.

DATA is a request method that can be originated by the client but also by server. Both (client and server) MUST NOT answer DATA messages.

7.6. QOS-ALERT

This is the request message that Q4S generates when the measurements indicate that quality SLA is 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 and the SLA). The QOS-ALERT message does not contain any detail on the actions to be taken, which depends on the agreements between all involved parties.

A QOS-ALERT request does not have to be answered unless there is an error condition. However, after receiving a QOS-ALERT request, the server sends a QOS-ALERT request to the client.

This method can be initiated by the client only after a 412 error coming from server, and with enough information to build the QOS-ALERT message.

If the “Q4S-Policy-Server” header was included in the server response of the handshake phase, the QOS-ALERT message MUST be sent to the URI indicated in this header, otherwise the QOS-ALERT message MUST be sent to the server.

With policy server, the QOS-ALERT message sent by the client MUST contain the URIs of the server and the client to be contacted later by the policy server. Therefore the following headers MUST be included in the client request: “Q4S-Resource-Server” and “Q4S-Resource-Client”.

The response to a Q4S QOS-ALERT request is not cacheable.

7.7. CANCEL

Like QOS-ALERT, this message is used for communication with the network resources. The semantics in this case is the release of the special resources assigned to the session.

In the same way as QOS-ALERT, CANCEL does not need to be answered. However, if the server receives a CANCEL message, it should send a new CANCEL request towards the client acknowledging the reception.

8. Response codes

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

8.1. 100 Trying

This response indicates that the request has been received by the next-hop server (the policy 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 QOS-ALERT by the client.

8.2. 200 OK

The request has succeeded.

8.3. 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.

8.4. 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.

8.4.1. 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”.

8.4.2. 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.

8.4.3. 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.

8.4.4. 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.

8.4.5. 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

8.4.6. 412 A precondition has not been met

The server is indicating that the SLA is being violated.

8.4.7. 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.

8.4.8. 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.

8.4.9. 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.

8.4.10. 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.

8.5. Server Failure 5xx

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

8.5.1. 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.

8.5.2. 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.

8.5.3. 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).

8.5.4. 504 Server Time-out

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

8.5.5. 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.

8.5.6. 513 Message Too Large

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

8.6. Global Failures 6xx

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

8.6.1. 600 session not exist

The Session-Id is not valid

8.6.2. 601 quality level not allowed

The QOS level requested is not allowed for the pair client/server

8.6.3. 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 QOS-ALERT is not allowed for the pair client/server, etc)

8.6.4. 604 authorization not allowed

The policy server does not authorize the QOS-ALERT operation because any internal or external reason.

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 browser 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. Bandwidth measurements

In programming languages or Operating Systems with timers or limited clock resolution, it is recommended to use an approach based on several intervals to send messages of 1KB, 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.

9.3. 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.

9.4. Measurements and reactions

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

The trigger is based on message QOS-ALERT, which is always forced by the server response 412 error. A server can avoid these QOS-REQUEST messages sending 200 OK when a GET message is received from server, independently whether the constraints are met or not.

9.5. Instability treatments

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

9.5.1. Loss of control packets

Lost UDP packets (PING or DATA messages) don’t cause any problems for the Q4S state machine, but if control packets like READY, 200 OK, 412 error, or GET messages are lost, some undesirable consequences could arise.

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

- If a READY packet is lost, after a certain timeout, the client SHOULD resend another READY packet.
- If the server’s expected 200 OK answer to the client’s READY message is lost, but PING packets begin to arrive, we assume that the initial received PING packet is enough and the client SHOULD start sending PING messages.
- If a GET packet is lost, the client will not receive any response by the server. After a certain timeout, the client SHOULD resend another GET message.
- If the 412 error message is lost, the server will receive a resent GET message instead a QoS-REQUEST message.

9.5.2. Outlier samples

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

In order to get rid of some outlier samples, that due to a bad spurious sample or an error on a measurement. The recommendation is to implement a median one dimension filter. This is a very common filtering for noise or errors on signal and image processing.

This is a very simple algorithm, where we keep a small window of previous measurements. We recommend a small window of 5 to 10 values. The new value is to get the median from the sorted tuple of window stored values.

This small window is not the same as the window used for calculating the average latency or jitter, but a simple mechanism to add a new (and more reliable) sample to the list.

9.6. Scenarios

Q4S could be used in two scenarios:

- client to ACP (Application content provider)
- client to client.

9.6.1. Client to ACP

In this scenario, the policy server is optional. If it exists, the QOS-ALERT messages MUST be sent to this policy server which acts as a proxy for this type of messages and validates them (plus any other actions out of scope of this document).

In order to avoid useless load on the server, the policy server could receive the BEGIN messages of handshake phase. For this purpose, the policy server MUST know the URI of the Q4S servers.

In this scenario a client could send the BEGIN to the policy server, with an additional parameter in the URI requested, which identifies the server, like:

Q4s://www.policy.com/listofservers?id=xtiwn28821ho4

Then the Policy Server validates the request and forward the BEGIN to the Q4S server, adding the Q4S-Resource-Server to the response for the client in the 200 OK response.

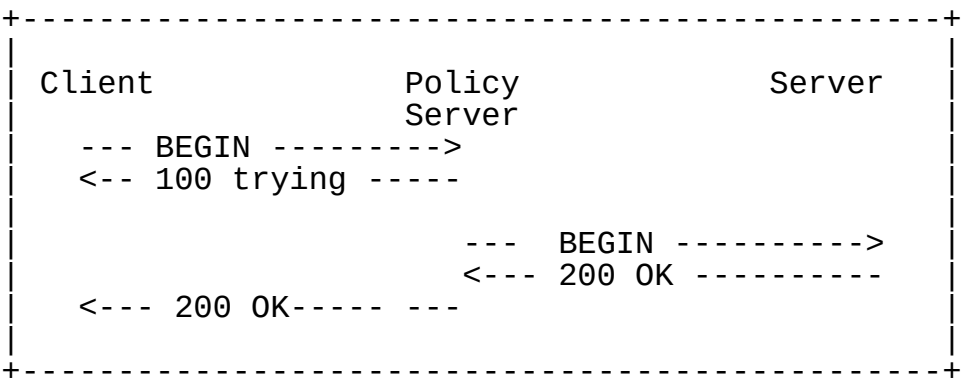

Figure 22 Policy server.

In this scenario the client MUST send further messages directly to the server without passing through policy server.

There is a possible scenario in which the policy server is contacted only by the Q4S server, enabled through the reception of the client measurements in PING messages that include the “measurements” header. In this case, the QOS-ALERT message does not cause an interruption of the sliding-window during continuity phase.

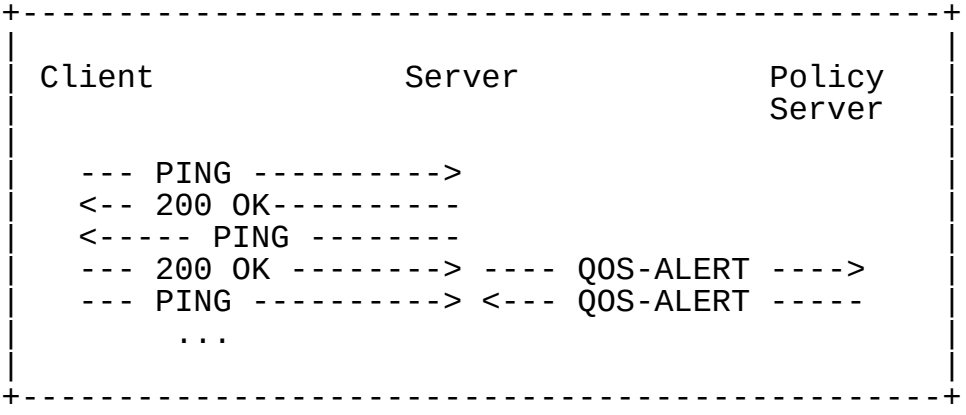

Figure 23 Alerts are sent by the server directly to the policy server

9.6.2. 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

Different types of attacks can be avoided:

- Spoofing of server IP address can be avoided using the digital signature mechanism. The network can easily validate this digital signature using the public key of the server certificate.

- The client could try to send QOS-ALERT requests constantly, trying to enter in the negotiation phase continuously. In this case, the server MUST answer a message “CANCEL”, in order to release the all levels reached and return to plain access without enhanced quality.

This protocol could be supported over IPSec to increase privacy, although it is out of scope of this proposal.

11. IANA Considerations

A specific port for Q4S TCP control flow mechanism could be assigned. It could simplify the network implementation. Other possibility is to use any other port (like 80, HTTP). In this case the network could use the protocol designator “Q4S” as the mark for distinguish and treat the packets.

Q4S uses SDP as a container for session information, in which quality attributes have been added as extended “session-level” attributes. These set of new attributes should be registered (in order to avoid the prefix “X-”). In this document, this set of attributes has been presented as registered attributes.

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 dependant but in general, a higher qos-level value corresponds to a better quality network profile.
Appropriate attribute values: [0..9] “/” [0..9]

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 in uplink and downlink for the communication between client and server. Appropriate attribute values: [0..9999] “/” [0..9999]
If there is no constraint in some direction (uplink, downlink or both) the value can be empty in that direction

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..99] “/” [0..99]

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 (data or control), direction (uplink or downlink) protocol (UDP or TCP) and port or range or ports
Attribute values:
<“control”|”data”> <“uplink”|”downlink”> <“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..255] “/” [0..255]

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|1 “,” [0..999]/[0..999] ”,” [0..255]/[0..255]]

where:

- The first parameter is the interval of time (in milliseconds) between PING messages in the negotiation phase. Forward (client to server) and reverse (server to client) values separated by “/”.

- The second parameter is the interval of time (in milliseconds) between PING messages in the continuity phase. Forward (client to server) and reverse (server to client) values separated by “/”.

- The third parameter is the time used to measure bandwidth during negotiation phase. In case of not present, a default value of 5000 ms will be assumed.

- The fourth parameter indicates the mode for continuity phase (0 means “normal” and 1 means “sliding window”). In case of not be present, normal mode (default value of 0) will be assumed.

- The fifth parameter is only applicable in sliding window mode. It indicates the window size for the jitter and latency calculation on both forward and reverse directions. If not present, a value of 256 MUST be assumed.

- The sixth parameter is only applicable in sliding window mode. It indicates the window size for packet loss calculations on both forward and reverse directions. If not present, a value of 256 MUST be assumed.

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

12. Conclusions

Q4S defines four phases with different purposes, and inside these phases the negotiated measurement procedure is used. Different measurement procedures can be used (even RTCP itself) inside Q4S. Basically, Q4S only defines how to transport SLA information and measurement results as well as providing some mechanisms for alerting. Q4S does not ask for resources. Q4S only alerts if one (or some) of SLA quality parameters are being violated. Depends on server (Application content provider) to do something with this information and return it back to a SLA-compliant state.

13. References

13.1. Normative References

[1] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,Masinter, L., Leach, P. and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1” RFC 2616, June 1999.

[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 RequirementLevels", 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] Johnsson, J., B. Kaliski, “Public-Key Cryptography Standards (PCS) #1: RSA Cryptography Specifications version 2.1”, RFC 3447, February 2003.

[8] Postel, J., "DoD Standard Transmission Control Protocol", RFC 761, January 1980.

[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

13.2. Informative References

[13] 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.

[14] Mathis, M., Semke, J., Mahdavi, J., Ott, T., "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", Computer Communications Review, 27(3), July 1997.

[15] Floyd, S., “HighSpeed TCP for a Large Congestion Windows”, RFC 3649, December 2003.

[16] Rhee, I., Xu, L., Ha, S., “CUBIC for Fast Long-Distance Networks”, Internet-draft draft-rhee-tcpm-cubic-02, February 2009.

[17] 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.

14. Acknowledgments

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

Sonia Herranz Pablo, Clara Cubillo Pastor, Francisco Duran Pina, Ignacio Moreno Lopez, Michael Scharf, Jesus Soto Viso and Federico Guillen.

15. Authors' Addresses

Jose Javier Garcia Aranda
Alcatel-Lucent
C/Maria Tubau 9
28050 Madrid
Spain
Phone: +34 91 330 4348
Email: Jose_Javier.Garcia_Aranda@alcatel-lucent.com

Jacobo Perez Lajo
Alcatel-Lucent
C/Maria Tubau 9
28050 Madrid
Spain
Phone: +34 91 330 4165
Email: jacobo.perez@alcatel-lucent.com

Luis Miguel Diaz Vizcaino
Alcatel-Lucent
C/Maria Tubau 9
28050 Madrid
Spain
Phone: +34 91 330 4871
Email: Luismi.Diaz@alcatel-lucent.com

Carlos Barcenilla
Universidad Politecnica de Madrid
Avenida Complutense 30
28040 Madrid
Spain
Phone: +34 91 549 5700 - 3032
Email: barcenilla@dit.upm.es

Monica Cortes
Universidad Politecnica de Madrid
Avenida Complutense 30
28040 Madrid
Spain
Phone: +34 91 336 5700 - 3044
Email: cortesm@dit.upm.es

Joaquin Salvachua
Universidad Politecnica de Madrid
Avenida Complutense 30
28040 Madrid
Spain

Phone: +34 91 549 5700 - 3056
Email: jsr@dit.upm.es

Juan Quemada
Universidad Politecnica de Madrid
Avenida Complutense 30
28040 Madrid
Spain
Phone: +34 91 336 7331
Email: jquemada@dit.upm.es