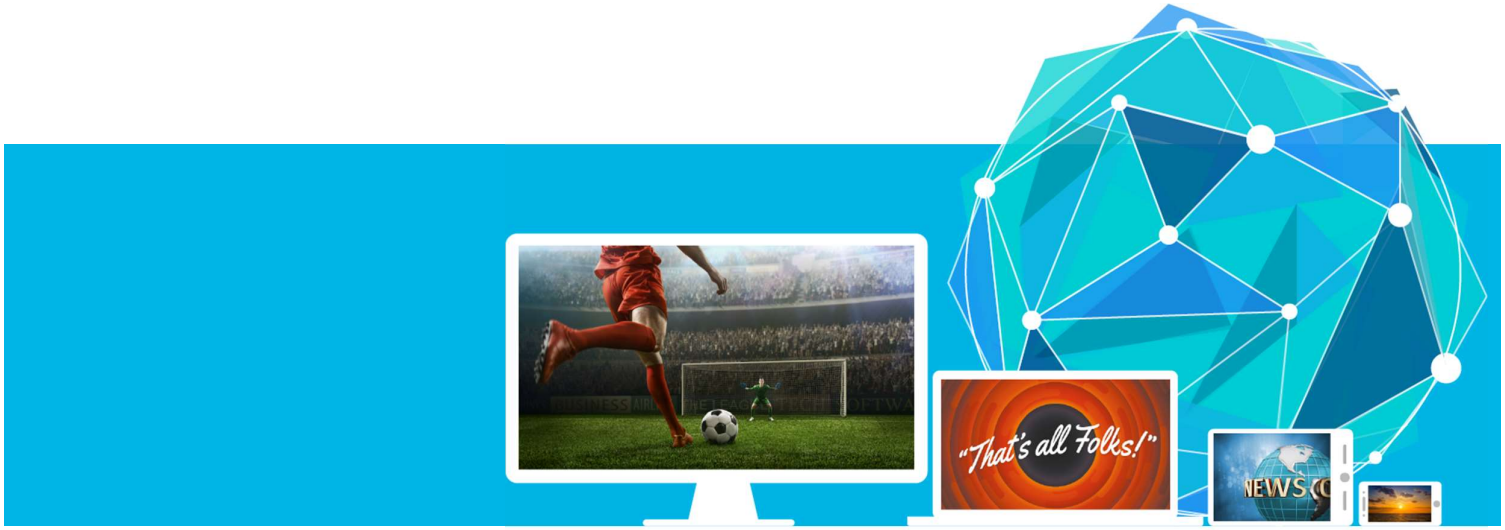
S4S: Modes of operation

Server Side Segment Selection for Streaming (S4S)



www.broadpeak.tv

License

Table of Contents

1 Introduction

In ABR (Adaptive Bit-Rate) streaming, the player controls the quality(bit-rate) selection function of its maximum available bandwidth estimation (most of the time based on the HTTP protocol) and possibly based on its buffer level. The player has one objective that is to maximize the quality of experience perceived by the user while avoiding rebuffering. However it does not and cannot indeed consider other players that may share the medium and therefore the bandwidth. In addition, bandwidth estimation performed by the player is most of the time based on HTTP (application layer) which does not work properly in some situations like with CMAF low latency.

Server Side Segment Selection for Streaming (S4S) is a technology that gives the control of the bandwidth/quality(bit-rate) tradeoff to the ISP. The S4S server selects the quality/bit-rate of the segment to be returned to the player relying on its own available bandwidth estimation (based on the underlaying transport's congestion control) and a bit-rate selection strategy possibly driven by a e.g. [business] rules engine though an API.

S4S provides also means for increasing the cooperation between the player and the cache server for a better user experience.

S4S has been primarily designed for low latency but can also be used in non-low latency deployment when the maximum video bit-rate must be controlled/imposed for a subset or the entire set of active media sessions.

This document specifies two modes of operation: one being [almost] totally transparent for the player and the other one being non transparent, requiring the support of a particular (lightweight) protocol . S4S applies to any ABR streaming protocols. However depending on terminal restrictions (i.e. IOS) it may not always possible to implement/deploy the S4S protocol.

2 Modes of operation

Although S4S is capable of functioning with HTTP 1.1, the handling of multiple concurrent transport connections makes the solution much more complex. Therefore S4S requires HTTP/2 and further versions for optimal operations. The player must therefore supports HTTP/2.

There are two modes of operation: transparent defaut) and non-transparent.
The default transparent mode relies on the support of self-initializing segment. It is fully compatible with MPEG DASH, and should not require any updates in the player. However, in practice, the support of self-initializing segments may not be effective and some player implementation would have to be updated.

> Note that the Dash-IF reference player implementation (dash.js) supports the self-initializing segments.

In non-transparent mode, the player is S4S aware. There is an exchange of information between the player and the server for increasing the quality of experience. The non-transparent mode does not require supporting the self-initializing segments. Two sub-modes are specified. The server driven mode and the server assisted mode. With the latter, the player is on its own but **must** take into account the information communicated by the server. The server assisted mode is used whenever there is no need of tight control of the player. The server decides about what mode the client should comply with. The mode can be selected for the entire session (e.g. on a client/player basis) or can possibly change at any time during a session.

2.1 Default (transparent) mode

With the default mode of S4S. S4S delivers self-initializing segments on [player] request. The player is not aware of the S4S presence, it behaves like in conventional ABR, attempting to control the quality selection.
A self-initializing segment is a media segment with a prepended initialization segment.
The player supporting the self-initialized segments must be capable of parsing the DSI information present at the beginning of the returned [self-initialized] segment.
In addition to the self-initializing segments, the s4S server relies on specific URIs declared on the manifest preserving the stateless nature of the S4S server. Again, this is fully compatible with the MPEG Dash standard and therefore transparent to the terminal.

2.1.1 The case of HLS

With HLS and HLS low latency self-initialized segments are supported by third parties HLS players (i.e. hls.js) but not by the IOS player (AVplayer). However there is the possibility to use a specific tag in the media playlist named EXT-X-GAP.

The presence of an EXT-X-GAP tag attached to a segment means that the player must not attempt to load the segment. Therefore such EXT-X-GAP tag must be added systematically to segments added to the variant stream playlist that corresponds to the variant stream disabled by the S4S server and so until the S4S server enables again the variant stream.

2.2 S4S aware player (non-transparent) mode

The purpose of this mode is in two folds: 1/permit information exchange between the S4S server and the player in order to enhance the control of the user experience and 2/ authorizes server assisted mode wherein the client selects the quality assisted by the server.

The information (said HTTP parameters) sent by the S4S aware player as part of this protocol **must** be according to one the two following format.

- HTTP request header
- HTTP request query parameters according to URL encoding format (RFC 3986)[1]

The information (said HTTP parameters) sent by the S4S server as part of this protocol **must** be formatted as HTTP response headers.

All parameter values encoding gathering multiple fields shall follow the format specified in RFC8941.

2.2.1 Overview

For the player to receive S4S information, it must connect to the server in HTTP/2 **(over SSL)** and must indicate through a HTTP parameter its ability to handle the S4S protocol of a certain version.

After having read the manifest and for all subsequent media requests, the player must indicate the list of supported representations (i.e. quality/bitrates) as a set of extra HTTP parameters (gathering the URL of each supported representation) and information related to its status (buffer length, stall statistics, measured latency, etc).

The S4S server **may** run in a server driven mode or in a server assisted mode. The former **may** rely on self-initializing segments[2] whereas the latter **must** not. These two modes of operation necessitate a change in the player implementation . This is up to the server to decide when working in server-driven mode or in server assisted mode. The mode is indicated in a specific HTTP parameter.

2.2.2 S4S presence

For the player to receive S4S information, it **must** connect to the server in HTTP/2 **(over SSL)** and **must** indicate through a header its capability to handle an S4S server followed by an application-key, server and origin-dependent. This HTTP parameter **must** be repeated in all requests to the server.

X-S4S: version=1, key=AE1987F43DCFEDA[3]

Similarly, the server includes in all HTTP responses a HTTP parameter signaling the S4S support, the protocol version and the mode of operation.

X-S4S: version=1, mode=<mode of operation>

The mode of operation is either “S” for server driven or “C” for Server assisted .

2.2.3 Opaque blob

The opaque blob is a data structure managed by the S4S server used to store a status information associated with the media session and/or the terminal/player the server is associated with. As an example, the S4S server may store the last TCP congestion windows value used with that player at the end of the session in order to accelerate the next session start). It may also store past bandwidth values for smoothing estimates, or anything that can help with the segment selection policy.

The server **may** (if it wants to store a session/terminal/player context) include the opaque blob structure in its HTTP response as an extra parameter. It is encoded according to RFC8941 as a Byte Sequence (base64 encoded):

X-S4S-Context: :cHJldGVuZCB0aGlzIGlzIGJpbmFyeSBjb250ZW50Lg==:

[1] The usage of HTTP request header in CORS situation may trigger preflight request which is potentially damageable for the streaming session.

[2] Depending on the client capability announced with the *X-S4S-Media HTTP parameter.*

[3] In case of relying on query string parameter: X-S4S=*version=1, mode=<mode of operation>*

The player **must**:

- For all requests (manifest and media chunks), save the most recent response S4S context header:

  X-S4S-Context: <Opaque Blob>

- Send the opaque blob back to the S4S server in all requests.

  X-S4S-Context: <Opaque Blob>

This blob value must persist across application restart, and even device restart. It can either be stored locally on the device (e.g., Web Storage API within browsers), within the CMS or within player's backend. As the blob value is encrypted, it does not leak information when stored remotely.

The blob value can be large (up to 4KB), the buffer in the player must be large enough to accommodate this.

Saved entries should be stored at least 30 days. Entries that have not been used/refreshed for more than 30 days can be deleted.

The format of the parameter in the request is identical to the response.

The parameter **can** be omitted only if the player has no knowledge of its value (i.e., first connection, no connection in the last 30 days, or the server did not send any context).

2.2.4 Media session - Client/player side

For media segment requests, the player **must** provide the following HTTP parameters that indicate to the server the list of representations (i.e. quality/bit-rates) supported by the terminal/player: limited to representations that match codec, frame rate or resolution constraints of the terminal/player.

X-S4S-Media: "/<path-media-1>/<sufix1>.mp4"; bitrate=10340000;background=96000;init="/<path-init-1>/<sufix-init-1>.mp4", "/<path-media-2>/<sufix2>.mp4";bitrate= 15340000;background=96000;init="/<path-init-2>/<sufix-init-2>.mp4", "/<path-media-3>/<suffix3>.mp4";bitrate=20000000;background=128000;init="/<path-init-3>/<sufix-init-3>.mp4"

The format is a list of parameterized strings according to RFC8941. The string is the absolute path of the URL . It must have the same encoding as the main URL (GET). Beware of double URL encoding ! The mandatory "bitrate" parameter is the bitrate of the corresponding layer as bits per second and it must be an integer value. The optional "background" parameter must be an integer value and represents the bitrate (in bits per second) consumed by non-video data (including e.g. audio, subtitle). If it is not present, the server will assume zero. The optional "init" parameter indicates the absolute path of the init segment. If the parameter is not present then the server cannot send self-initializing segments; this is up to the player to manage the retrieval of such init segment.

Note that if multiple supported representations at the same bitrate exist, the player can choose the one it prefers.

Note that the server sends back the HTTP response that includes the *X-S4S-Media* HTTP parameter with a parameterized string corresponding to the selected quality/bitrate media segment:

X-S4S-Media: "/<path-media-1>/<sufix1>.mp4";bitrate=10340000; bitrate=10340000;background=96000; init="/<path-init-1>/<sufix-init-1>.mp4"

Note that the "background" and "init" parameters are present in the response only if they were present in the request.

In the Server assisted mode of operation or in server driven mode of operation if the "init" parameter was not joined along with the selected segment path as part of the "X-S4S-Media" HTTP parameter in the request, the client **must** manage the initialization segments conventionally meaning it **must** download it (if not present) and **must** initialize correctly its decoder before processing the corresponding media segment.

In the Server driven mode of operation, if the "init" parameter was joined along with the selected segment path as part of the "X-S4S-Media" HTTP parameter in the request, the returned media segment is self-initializing and therefore, the player has no need to deal with the initialization segments.

Last, but not least, the player must detect when the server leaves the server driven mode of operation which corresponds to whenever the mode = "C" appears the first time after having received at least one *mode="S"* or after the beginning of a session (receiving the manifest). In that case, the player must proceed further with the initialization segments conventionally without any assumption about how the decoder has been initialized having possibly run in Server driven mode before.

2.2.5 Client-side information[4]

In all media requests, the player **may** provide the following information as HTTP parameters that may help the server in selecting the quality/bitrate.

X-S4S-CSI: video_buffer_level=<number of seconds of video currently in the player's buffer>, audio_buffer_level=<number of seconds of audio currently in the player's buffer>, last_stall_timestamp=1944403.310, last_stall_duration=3.200, current_state=p(playing)|a(paused)|s (stalling)

- Last_stall_timestamp and last_stall_duration are the EPOCH time and duration of last stall experienced by the player, respectively. currentState indicates the current state of the player. If the play hasn't experienced any stall since the beginning of the streaming session, these fields are absent.
- All time related values are floating point in seconds.

2.2.6 Media session - Server side

In Server assisted mode of operation, if the requested segment is available, the server accepts the request with the 200 response code and joins the *X-S4S-Media* HTTP parameter with the parameterized string corresponding to the quality/bitrate selected by the player:

X-S4S-Media: "/<path-media-1>/<sufix1>.mp4";bitrate=10340000; background=96000;init="/<path-init-1>/<sufix-init-1>.mp4"

The server returns the segment corresponding to the quality/bitrate selected by the player.

[4] Note that part or all of this information could be delivered through the CMCD CTA-5004 specification when completed (currently under community review).

Note that the “background” and “init” parameters are present in the response only if they were present in the request.

In Server driven mode of operation, whatever the requested quality/bitrate, the server accepts the request with the 200 response code and joins the *X-S4S-Media* HTTP header with the parameterized string corresponding to its selected quality/bitrate.

X-S4S-Media: “/<path-media-1>/<sufix1>.mp4”;bitrate=10340000; background=96000;init=“/<path-init-1>/<sufix-init-1>.mp4”

If the “init” parameter is present (meaning it was present in the request) then the server returns the self-initializing segment corresponding to its selected quality/bitrate. Otherwise, it returns a non-self-initialized segment And this is up to the player to manage correctly the “init” segment.

2.2.7 Server-side information

In addition, in all media responses, the server **should** send the estimated available bandwidth, the maximum allowed bitrate and the round trip time .

X-S4S-SSI: ebw=1384034, mbr=1384034, rtt=12

Note the values of *estimated bandwidth* (ebw) and *maximum bitrate* (mbr) are in bits per second and can be large, it must be parsed/stored as a uint64_t. It correspond to the TCP bitrate (thus excluding overheads of IP, Ethernet and all other encapsulating layers).

The *estimated bandwidth* (ebw) value **must** be understood by the client as the maximum available bandwidth for the client measured by the S4S server during the last bulk (e.g. segment) transfer. This value possibly computed (min, max, average) over a set of measurement samples. The player **may** integrate this value in a smoothing bandwidth formula.

The *max bitrate* (mbr) value **must** be understood by the client as the maximum allowed bit-rate (or available bandwidth) imposed by the network for various reasons (fair bandwidth sharing, decrease the load on the CDN). It is a strict boundary that must not be smoothed.

The value of rtt is in ms.

2.2.7.1 S4S information request

In case of operating in Server assisted mode, the server side information is known by the client at the beginning of the response (i.e. the segment response) and therefore it does not relate to the last sending. For a better accuracy, the client **may** use a specific request (in high priority) for getting at any time the SSI information according to the following:

GET <cache server fqdn>/S4S/info

The client must join as with any other S4S request, the HTTP headers corresponding to the version, key and CSI information but does not need to join the blob.

The server response does not contain data but the conventional headers (version, mode) and the SSI information.

2.2.8 Support of CTA-5004 and CTA-5006 specifications

If required, the Common Media Client Data (CMCD-5004) and Common Media Server Data (CTA CMSD-5006) specifications are supported.

2.2.8.1 CMCD usage (by the S4S aware client)

The client **must** vehiculate the S4S keys and values over CMCD according to one of the following methods as specified in section 2 of the CMCD specification.

Adding an extra HTTP header: CMCD-Request:**tv.broadpeak.s4s-**<**key**>=<**value**> 1*[,**tv.broadpeak.s4s-**<**key**>=<**value**>]

- key: any client side key defined in this document and listed in section 2.2.9.
- value: the value corresponding to the S4S key.
- Operating the query argument CMCD=<URL_encoded_concatenation_of_S4S key_value_pairs>The S4S key value pairs must be ordered in alphabetical order.
- The S4S key value pairs must be formatted according to the following.
    - [?/&] CMCD=**tv.broadpeak.s4s-**< **key**> %3D<**value**>1*[%2C**tv.broadpeak.s4s-**<**key**>%3D<**value**>]

Note that if CMCD is used then the client shall not vehiculate the S4S keys which have their equivalent CMCD key such as bl, bs and ot (see section 2.2.9).

2.2.8.2 CMSD usage (by the S4S server)

The server **may** communicate some CMSD well defined keys according to the following.

CMSD-Dynamic: <**identifier**>;**etp**=1384;**rtt**=12;**mb**=1000

- <Identifier> is a string of characters that should be unique and remanent across the connections/sessions as e.g. *"bks400-POPN2-S4S"*
- "**etp**" is a CMSD reserved key and corresponds to the estimated throughput in Kilobits per second. The value must be equivalent to the S4S "ebw" value defined above (though not expressed with the same unit)
- "**rtt**" is a CMSD reserved key and corresponds to the round trip time estimate in milliseconds. The value must be equivalent to the S4S "**rtt**" value defined above.
- "**mb**" is a CMSD reserved key and corresponds to the *maximum bitrate* in milliseconds. The value must be equivalent to the S4S "**mbr**" value defined above.

The server **must** vehiculate the S4S keys and values over CMSD according to the following method as specified in section 4 of the CMSD specification.

Adding an extra HTTP header: CMSD-Dynamic: <**identifier**> 1*[;**tv.broadpeak.s4s-**<**key**>="<**value**>"]

- <Identifier> is a string of characters that should be unique and remanent across the connections/sessions as e.g. *"bks400-POPN2-S4S"*
- key: any server side key defined in this document and listed in section 2.2.9.
- "value": the value corresponding to the S4S key formatted as a string.

Note that if CMSD is used to vehiculate the three listed keys/values above (etp, rtt, mb) then the server shall not vehiculate the three corresponding S4S keys through the method indicated right above.

Note that for the X-S4S-Media key, if CMSD is used then the value must only contain the bitrate according to the following.

CMSD-Dynamic: <**identifier**> ;**tv.broadpeak.s4s-birate**="<**value**>"

2.2.9 Summary of S4S HTTP parameters

Table 1- Client request: HTTP headers

Header	[Dictionary] Key	Value	Mode of operation	Mandatory/ optional	CMCD Header	CMCD key
X-S4S	version	Integer	All	M (all requests)	CMCD-Session	tv.broadpeak.s4s-version *Note: need to distinguish CMCD version and S4S version in case we add custom keys*
X-S4S	key	String (opaque)	All	M (all requests)	CMCD-Session	tv.broadpeak.s4s-key
X-S4S-Context	N/A	Bytes sequence (base64)	All	M (all requests)	CMCD-Session	tv.broadpeak.s4s-context
X-S4S-Media	N/A	List of Parameterized strings	All	M (media requests)[5]	CMCD-Request	tv.broadpeak.s4s-media *Note: in gzip base64 format*
X-S4S-CSI	video_buffer_level or vbl	Floating point (s)	All	M (all requests)	CMCD-Request	bl *Note: CMCD-Request "ot" value provides the media type for which the buffer level applies*
X-S4S-CSI	audio_buffer_level or abl	Floating point (s)	All	M (all requests)	CMCD-Request	bl *Note: CMCD-Request "ot" value provides the media type for which the buffer level applies*

[5] The parameters "background" and « init » are optional

X-S4S-CSI	last_stall_timestamp or lst	Floating point (s)	All	M (all requests)		
X-S4S-CSI	last_stall_duration or lsd	Floating point (s)	All	M (all requests)		
X-S4S-CSI	current_state or cst	Char	All	M (all requests)	CMCD-Status	bs *Note: "bs" value can be used to determine if current is "playing" or "stalling"*

Table 2- Server response: HTTP headers and Key/value pairs

Header	[Dictionary] Key	Value	Mode of operation	Mandatory/optional	CMSD Header	CMSD Key
X-S4S	version	Integer	All	M (all responses)	CMSD-Static	tv.broadpeak.s4s-version
X-S4S	mode	String	All	M (all responses)	CMSD-Static	tv.broadpeak.s4s-mode
X-S4S-Context	N/A	Bytes sequence (base64)	All	O	CMSD-Static	tv.broadpeak.s4s-context
X-S4S-Media	N/A	Parameterized string	Server driven	M (media responses)[6]	CMSD-Static	tv.broadpeak.s4s-bitrate *Note: provide only selected bitrate*
X-S4S-Media	N/A	Parameterized string	Server assisted	O	CMSD-Static	tv.broadpeak.s4s-bitrate *Note: provide only selected bitrate*
X-S4S-SSI	ebw	Integer (bps)	Server driven	O	CMSD-Dynamic	etp *Note: etp is expressed in kbps*
X-S4S-SSI	mbr	Integer (bps)	Server driven	O	CMSD-Dynamic	mb *Note: mb is expressed in kbps*
X-S4S-SSI	rtt	Integer (ms)	Server driven	O	CMSD-Dynamic	rtt
X-S4S-SSI	ebw	Integer (bps)	Server assisted	M (all responses)	CMSD-Dynamic	etp *Note: etp is expressed in kbps*
X-S4S-SSI	mbr	Integer (bps)	Server assisted	M (all responses)	CMSD-Dynamic	mb *Note: mb is expressed in kbps*

[6] The parameters "background" and « init » are present only they were present in the corresponding request.

X-S4S-SSI	rtt	Integer (ms)	Server assisted	M (all responses)	CMSD-Dynamic	rtt

3 Sequence flows (S4S aware)

The following diagram illustrates an example with a stream having two quality/bitrate sub-streams, one HD ready (1280X720) and one sub-VGA (640X360). Note that in all HTTP transaction flows, the content path is composed basically with a server name (myCDN.com). It is an example and more generally, no assumption should be done on how the content path is composed. It may be absolute as in the example or relative.

Note that the CMCD/CMSD formalism is not used in this example.

3.1 Server driven mode

1. The player downloads the manifest and computes the list of representations it is willing to support (session initialization). It is a live stream and the next segment available has the number 13.

   Note that the player does communicate an opaque blob in its request assuming he could have found an opaque blob matching the server's FQDN

   In the next step, it starts downloading media segments.

2. Based on its bandwidth evaluation and the one delivered by the server, the player is ready to request the segment 13 of the highest quality (1280X720).

   Note that conforming to the specification above, the player includes in all its request the HTTP X-S4S header. In addition, because the S4S server sends an opaque blob (attached with the manifest response), after having stored the blob, the player must include it in all its further requests.

1. Next, the player requests the segment 13 of its selected quality (1280X720). It includes the list of the absolute URLs corresponding to all the representations (i.e. quality/ bitrate) it supports through the "X-S4S-Media" HTTP header. In addition it includes the status metrics as it is mandatory for the server ("X-S4S-CSI" HTTP header).
2. The S4S server drives the quality selection; it forces the player to download the segment of its own selected representation (i.e. quality/bitrate). The server returns the segment indicating its choice to the player (myCDN.com/3096000_1280x720/seg13.m4s) ("X-S4S-Media" HTTP header).
3. Next, the player requests the next segment 14.
4. In the meantime, the server has measured the available bandwidth going down.
5. The S4S server drives the quality selection; it forces the player to download the segment of its own selected representation (i.e. quality/bitrate). Checking the request, there is a mismatch between the player's quality/bitrate requested (3096000_1280x720) and the server's decision/selection (746000_640x360) based on its own available bandwidth estimation. The server returns the segment indicating its choice to the player (myCDN.com/746000_640x360/seg14.m4s).

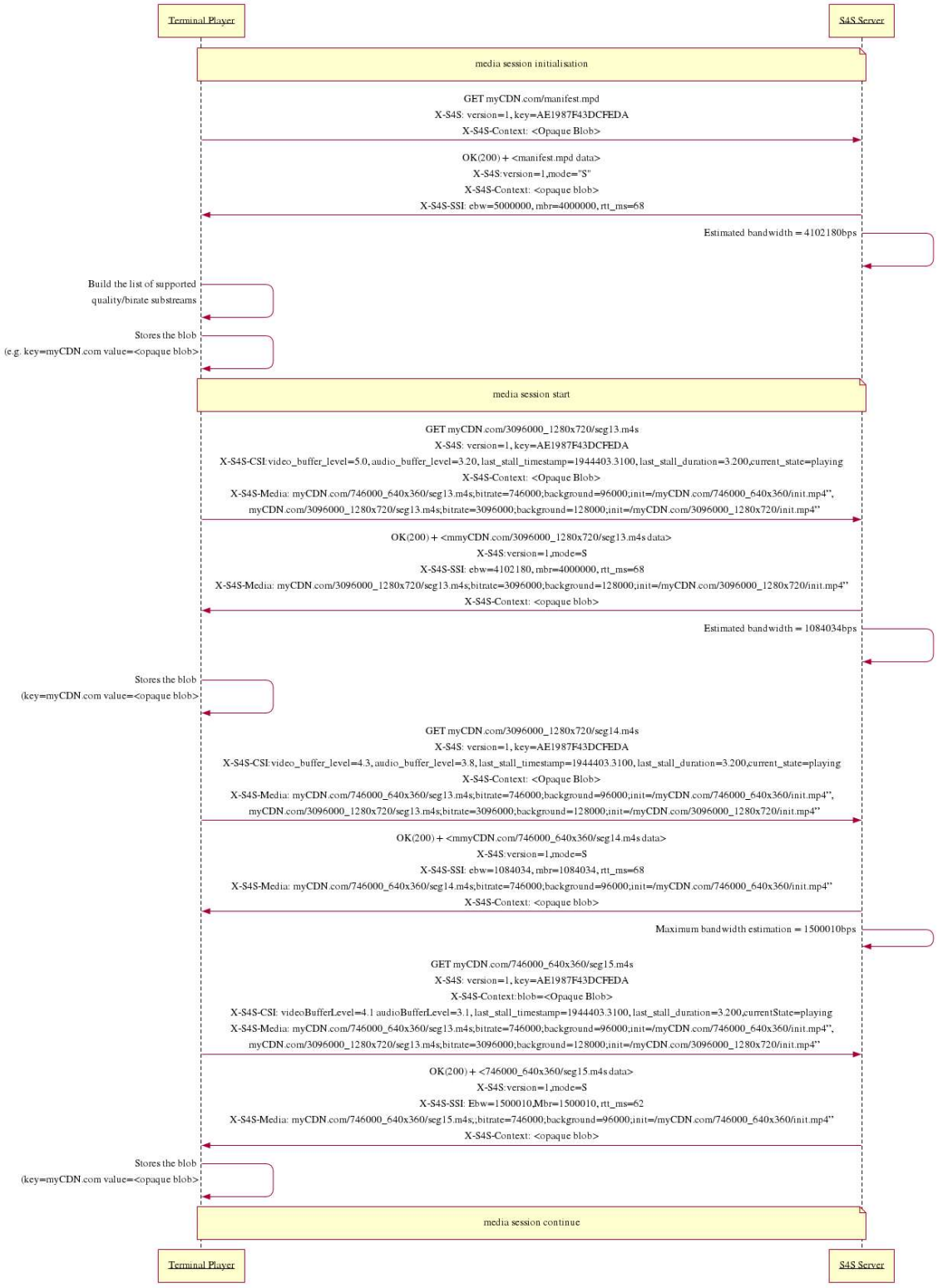


Figure 1- Server driven mode sequence flow

Table 3 – Server driven mode - *WebSequenceDiagrams script*

```
participant Terminal Player as TP
participant S4S Server as S4
note over TP,S4: media session initialisation
TP->S4: GET myCDN.com/manifest.mpd\nX-S4S: version=1, key=AE1987F43DCFEDA\nX-S4S-Context: <Opaque Blob>
S4->TP: OK(200) + <manifest.mpd data> \nX-S4S:version=1,mode="S"\nX-S4S-Context: <opaque blob>\nX-S4S-SSI: ebw=5000000,
mbr=4000000, rtt_ms=68
S4->S4: Estimated bandwidth = 4102180bps\n
TP->TP: Build the list of supported\nquality/birate substreams
TP->TP: Stores the blob\n (e.g. key=myCDN.com value=<opaque blob>
note over TP,S4: media session start
TP->S4: GET myCDN.com/3096000_1280x720/seg13.m4s\nX-S4S: version=1, key=AE1987F43DCFEDA\nX-S4S-
CSI:video_buffer_level=5.0, audio_buffer_level=3.20, last_stall_timestamp=1944403.3100,
last_stall_duration=3.200,current_state=playing\nX-S4S-Context: <Opaque Blob>\nX-S4S-Media:
myCDN.com/746000_640x360/seg13.m4s;bitrate=746000;background=96000;init=/myCDN.com/746000_640x360/init.mp4",\n
myCDN.com/3096000_1280x720/seg13.m4s;bitrate=3096000;background=128000;init=/myCDN.com/3096000_1280x720/init.mp4"
S4->TP: OK(200) + <mmyCDN.com/3096000_1280x720/seg13.m4s data>\nX-S4S:version=1,mode=S\nX-S4S-SSI: ebw=4102180,
mbr=4000000, rtt_ms=68\nX-S4S-Media:
myCDN.com/3096000_1280x720/seg13.m4s;bitrate=3096000;background=128000;init=/myCDN.com/3096000_1280x720/init.mp4"\nX
-S4S-Context: <opaque blob>
S4->S4: Estimated bandwidth = 1084034bps\n
TP->TP: Stores the blob\n (key=myCDN.com value=<opaque blob>
TP->S4: GET myCDN.com/3096000_1280x720/seg14.m4s\nX-S4S: version=1, key=AE1987F43DCFEDA\nX-S4S-
CSI:video_buffer_level=4.3, audio_buffer_level=3.8, last_stall_timestamp=1944403.3100,
last_stall_duration=3.200,current_state=playing\nX-S4S-Context: <Opaque Blob>\nX-S4S-Media:
myCDN.com/746000_640x360/seg13.m4s;bitrate=746000;background=96000;init=/myCDN.com/746000_640x360/init.mp4",\n
myCDN.com/3096000_1280x720/seg13.m4s;bitrate=3096000;background=128000;init=/myCDN.com/3096000_1280x720/init.mp4"
S4->TP: OK(200) + <mmyCDN.com/746000_640x360/seg14.m4s data>\nX-S4S:version=1,mode=S\nX-S4S-SSI: ebw=1084034,
mbr=1084034, rtt_ms=68\nX-S4S-Media:
myCDN.com/746000_640x360/seg14.m4s;bitrate=746000;background=96000;init=/myCDN.com/746000_640x360/init.mp4"\nX-S4S-
Context: <opaque blob>
S4->S4: Maximum bandwidth estimation = 1500010bps
TP->S4: GET myCDN.com/746000_640x360/seg15.m4s\nX-S4S: version=1, key=AE1987F43DCFEDA\nX-S4S-Context:blob=<Opaque
Blob>\nX-S4S-CSI: videoBufferLevel=4.1 audioBufferLevel=3.1, last_stall_timestamp=1944403.3100,
last_stall_duration=3.200,currentState=playing\nX-S4S-Media:
myCDN.com/746000_640x360/seg13.m4s;bitrate=746000;background=96000;init=/myCDN.com/746000_640x360/init.mp4",\n
myCDN.com/3096000_1280x720/seg13.m4s;bitrate=3096000;background=128000;init=/myCDN.com/3096000_1280x720/init.mp4"
S4->TP: OK(200) + <746000_640x360/seg15.m4s data>\nX-S4S:version=1,mode=S\nX-S4S-SSI: Ebw=1500010,Mbr=1500010,
rtt_ms=62\nX-S4S-Media:
myCDN.com/746000_640x360/seg15.m4s;;bitrate=746000;background=96000;init=/myCDN.com/746000_640x360/init.mp4"\nX-S4S-
Context: <opaque blob>
TP->TP: Stores the blob \n (key=myCDN.com value=<opaque blob>
note over TP,S4: media session continue
```

3.2 Server assisted mode

1. The player downloads the manifest and computes the list of representations it is willing to support (session initialization). It is a live stream and the next segment available has the number 13.
   Note that the player does communicate an opaque blob in its request assuming he could have found an opaque blob matching the server's FQDN
   In the next step, it starts downloading media segments.

2. Based on its bandwidth evaluation, the player is ready to request the segment 13 of the highest quality (1280X720). However, because this is the first time the player downloads a segment of this quality/bitrate, it must download first the corresponding initialization segment (3096000_1280x720.init).

3. Note that conforming to the specification above, the player includes in all its request the HTTP X-S4S header. In addition, because the S4S server sends an opaque blob (attached with the manifest response), after having stored the blob, the player must include it in all its further requests.

4. Next, the player requests the segment 13 of its selected quality (1280X720). It includes the list of the absolute URLs corresponding to all the representations (i.e. quality/ bitrate) it supports, the "X-S4S-Media". HTTP header. In addition it includes the status metrics as it is mandatory for the server ("X-S4S-CSI" HTTP header).
5. The S4S server is in server assisted mode; it lets the player to download the segment of its own selected representation (i.e. quality/bitrate) but taking into account information from the S4S server.
6. The Server returns the requested segment and updates its available bandwidth estimation
7. The player would have requested the segment 14 with the highest bitrate that is above the available bandwidth measured by the server which has not yet got the opportunity to communicate its estimation to the client. However it may operate the info request which permits to be aware of the server's bandwidth estimation before sumiting the request dor the segment 14.
8. Similarily, for the next segment request (15), the payer adjusts its bandwidth estimation taking into account the server's bandwidth estimation operating the info request.
9. The player can submit the subsequent request. According to the S4S server measured maximum bandwidth indication (1084034bps), the player requests the lowest representation; the associated bitrate (746000 bps) being below the maximum measured by the server and may be also observed by the player.
10. The S4S server returns the segment of the requested quality.

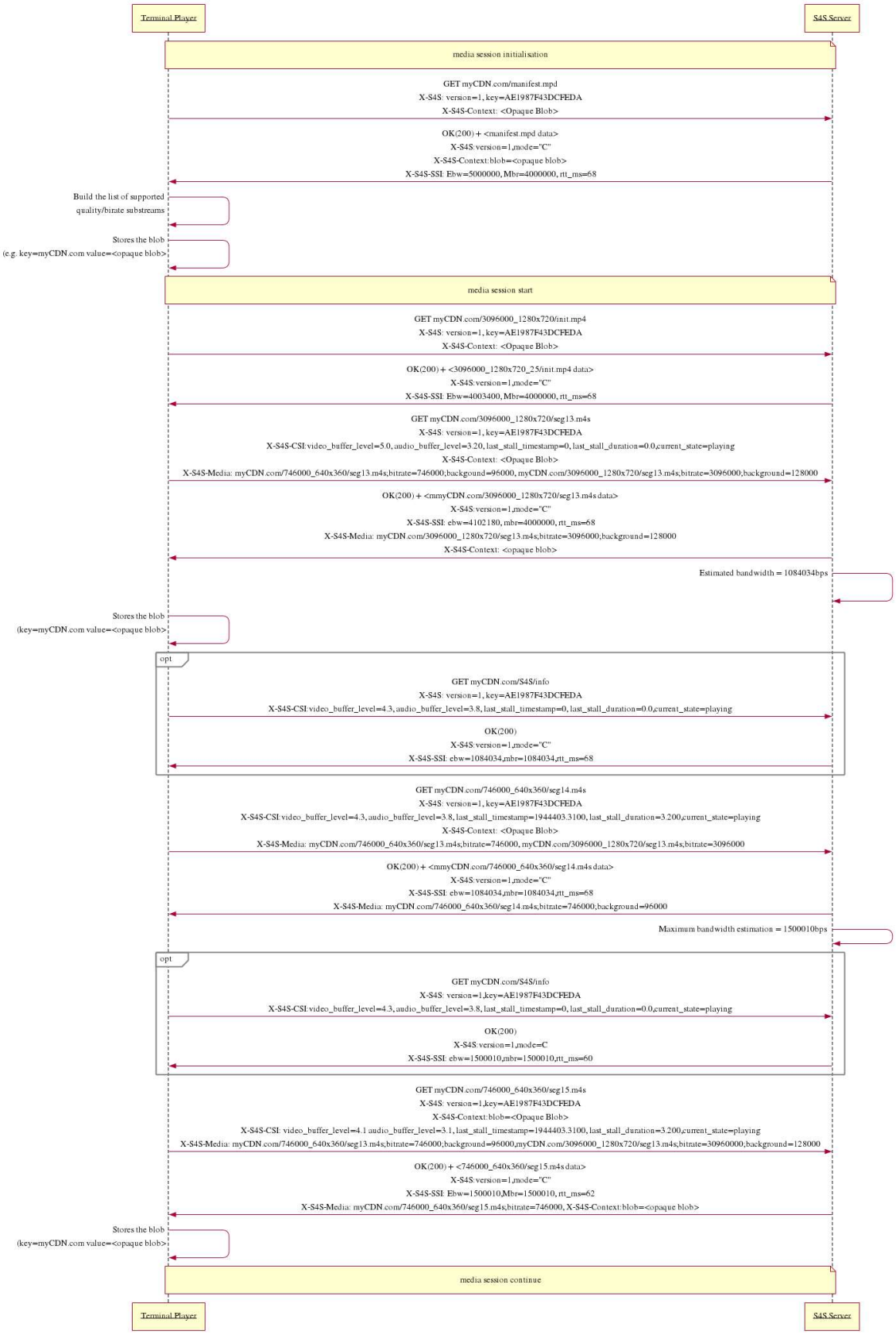


Figure 2- Client driven mode sequence flow

Table 4 - Server assisted mode - *WebSequenceDiagrams script*

```
participant Terminal Player as TP
participant S4S Server as S4
note over TP,S4: media session initialisation
TP->S4: GET myCDN.com/manifest.mpd\nX-S4S: version=1, key=AE1987F43DCFEDA\nX-S4S-Context: <Opaque Blob>
S4->TP: OK(200) + <manifest.mpd data> \nX-S4S:version=1,mode="C"\nX-S4S-Context:blob=<opaque blob>\nX-S4S-SSI: Ebw=5000000,
Mbr=4000000, rtt_ms=68
TP->TP: Build the list of supported\nquality/birate substreams
TP->TP: Stores the blob\n (e.g. key=myCDN.com value=<opaque blob>
note over TP,S4: media session start
TP->S4: GET myCDN.com/3096000_1280x720/init.mp4\nX-S4S: version=1, key=AE1987F43DCFEDA\nX-S4S-Context: <Opaque Blob>
S4->TP: OK(200) + <3096000_1280x720_25/init.mp4 data>\nX-S4S:version=1,mode="C"\nX-S4S-SSI: Ebw=4003400, Mbr=4000000,
rtt_ms=68
TP->S4: GET myCDN.com/3096000_1280x720/seg13.m4s\nX-S4S: version=1, key=AE1987F43DCFEDA\nX-S4S-
CSI:video_buffer_level=5.0, audio_buffer_level=3.20, last_stall_timestamp=0, last_stall_duration=0.0,current_state=playing\nX-S4S-
Context: <Opaque Blob>\nX-S4S-Media: myCDN.com/746000_640x360/seg13.m4s;bitrate=746000;backgound=96000,
myCDN.com/3096000_1280x720/seg13.m4s;bitrate=3096000;background=128000
S4->TP: OK(200) + <mmyCDN.com/3096000_1280x720/seg13.m4s data>\nX-S4S:version=1,mode="C"\nX-S4S-SSI: ebw=4102180,
mbr=4000000, rtt_ms=68\nX-S4S-Media: myCDN.com/3096000_1280x720/seg13.m4s;bitrate=3096000;background=128000\nX-S4S-
Context: <opaque blob>
S4->S4: Estimated bandwidth = 1084034bps\n
TP->TP: Stores the blob\n (key=myCDN.com value=<opaque blob>
Opt
TP->S4: GET myCDN.com/S4S/info\nX-S4S: version=1, key=AE1987F43DCFEDA\nX-S4S-CSI:video_buffer_level=4.3,
audio_buffer_level=3.8, last_stall_timestamp=0, last_stall_duration=0.0,current_state=playing
S4->TP: OK(200)\nX-S4S:version=1,mode="C"\nX-S4S-SSI: ebw=1084034,mbr=1084034,rtt_ms=68
end Opt
TP->S4: GET myCDN.com/746000_640x360/seg14.m4s\nX-S4S: version=1, key=AE1987F43DCFEDA\nX-S4S-CSI:video_buffer_level=4.3,
audio_buffer_level=3.8, last_stall_timestamp=1944403.3100, last_stall_duration=3.200,current_state=playing\nX-S4S-Context:
<Opaque Blob>\nX-S4S-Media: myCDN.com/746000_640x360/seg13.m4s;bitrate=746000,
myCDN.com/3096000_1280x720/seg13.m4s;bitrate=3096000
S4->TP: OK(200) + <mmyCDN.com/746000_640x360/seg14.m4s data>\nX-S4S:version=1,mode="C"\nX-S4S-SSI:
ebw=1084034,mbr=1084034,rtt_ms=68\nX-S4S-Media: myCDN.com/746000_640x360/seg14.m4s;bitrate=746000;background=96000
S4->S4: Maximum bandwidth estimation = 1500010bps
Opt
TP->S4: GET myCDN.com/S4S/info\nX-S4S: version=1,key=AE1987F43DCFEDA\nX-S4S-CSI:video_buffer_level=4.3,
audio_buffer_level=3.8, last_stall_timestamp=0, last_stall_duration=0.0,current_state=playing
S4->TP: OK(200)\nX-S4S:version=1,mode=C\nX-S4S-SSI: ebw=1500010,mbr=1500010,rtt_ms=60
end Opt
TP->S4: GET myCDN.com/746000_640x360/seg15.m4s\nX-S4S: version=1,key=AE1987F43DCFEDA\nX-S4S-Context:blob=<Opaque
Blob>\nX-S4S-CSI: video_buffer_level=4.1 audio_buffer_level=3.1, last_stall_timestamp=1944403.3100,
last_stall_duration=3.200,current_state=playing\nX-S4S-Media:
myCDN.com/746000_640x360/seg13.m4s;bitrate=746000;background=96000,myCDN.com/3096000_1280x720/seg13.m4s;bitrate=309
60000;background=128000
S4->TP: OK(200) + <746000_640x360/seg15.m4s data>\nX-S4S:version=1,mode="C"\nX-S4S-SSI: Ebw=1500010,Mbr=1500010,
rtt_ms=62\nX-S4S-Media: myCDN.com/746000_640x360/seg15.m4s;bitrate=746000, X-S4S-Context:blob=<opaque blob>
TP->TP: Stores the blob \n (key=myCDN.com value=<opaque blob>
note over TP,S4: media session continue
```