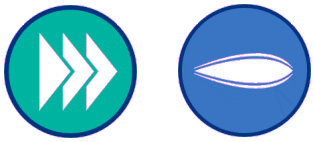

Packet Ship *Streamline* Media Server

Release Note

3.1 “Antigua” Update 2

Package	Version	Revision
ps-streamd	3.1.2	1
ps-index-mpeg2ts	3.1.2	1
ps-analyse-mpeg2ts	3.1.2	1

Release Note

The 3.1 “Antigua” Update 2 release of the Packet Ship Streamline media server contains some bug fixes and improvements to the RTSP controller, support of encrypted content and SSL in HLS, correct handling of BBC HD content and audio-only streams, improved statistics from the indexer and stream analyser:

- Fix RTSP Range headers in responses
- 'start', 'end' and 'variant' parameters supported in RTSP URLs
- Report accurate duration in DESCRIBE
- 'duration' parameter in GET_PARAMETER
- Support for HLS encrypted content
- Support for SSL in HLS and HTTP
- Multiple controllers of the same type
- Withdrawal of `<offset rounding>`
- Handle BBC HD content in ps-index-mpeg2ts
- Handle audio streams in ps-index-mpeg2ts
- Output rate and duration from ps-index-mpeg2ts
- Correct rate output from ps-analyse-mpeg2ts to match ps-index-mpeg2ts

RTSP Range headers

The Range header returned in responses from SETUP, PLAY and PAUSE has been fixed to always return the current position and the current duration (or early end point, if set). This is returned whether or not the command specified a Range header originally.

Note that the position returned is the position at the time the command was received; because changes happen asynchronously within the server and may be subject to I-frame alignment, the actual position at which the change takes place may be slightly different. If it is critical to get the exact position (e.g. in PAUSE) then a GET_PARAMETER(position) a second later will return the exact value.

A bug inherited from 2.3.x where a Range header was not returned from SETUP if 'play immediate' was not set has also been fixed.

RTSP URL parameters

The RTSP controller now supports '`start`' and '`end`' parameters on the request URL, giving a start position to seek to and optional end position to finish at, both expressed in NPT seconds. These values are in addition to any specified by a # fragment on the URL, and/or configured in the asset definition itself.

It also supports a '`variant`' parameter to allow choice of one of a number of variant files (counting from 0) with the same asset ID, in a similar way to HTTP Live Streaming.

The combination of the new parameters provides a mechanism to implement a type of adaptive stream in RTSP.

Duration in DESCRIBE

The duration reported in “a=range” in the DESCRIBE SDP response was previously rounded down to the nearest integer. This is no longer the case and the duration is now floating point and (unlike 2.3.x) represents the actual duration calculated from the index file rather than an estimate based on file length and configured rate.

GET_PARAMETER(duration)

The GET_PARAMETER command can now obtain the duration of the stream as well as its current position. If the string “duration” appears anywhere within the body of the request, a line containing “duration: “ and a floating-point duration will be returned in the result.

If both 'position' and 'duration' are requested, the position will be returned first, whatever order they are requested in.

e.g. Request:

```
GET_PARAMETER rtsp://server/asset RTSP/1.0
Session: 384acc62acf10001
CSeq: 7

duration
position
```

Response:

```
RTSP/1.0 200 OK
Content-length: 38
CSeq: 7
Server: Packet Ship RTSP Server v3.1.2
Date: Fri, Oct 07 2011 16:10:28 GMT
Content-type: text/parameters

position: 1823.39
duration: 7452.88
```

Support for HLS encrypted content

The HLS controller now supports generation of EXT-X-KEY tags in leaf playlists, allowing the client to fetch an AES key for the content. To enable this, create a block like the following in the **`<hls>`** controller in `streamd.cfg.xml`


```
  <!-- Encryption configuration -->
  <encryption type = "AES-128">

    <!-- Template of URL to use for EXT-X-KEY tags -->
    <!-- Template is interpolated with $ variables:
           $ASSET    Asset ID
           $LICENCE  Licence/session ID passed in from top-level playlist URL
    -->
    <url template = "https://keyline.packetship.tv:6780/get-key?
asset=$ASSET&amp;licence=$LICENCE"/>

  </encryption>
```

The encryption type should always be AES-128 (this is the only encryption supported by HLS), or empty (the default) in which case generation of EXT-X-KEY tags is disabled. Note that if encryption is enabled, all content served by the controller will have EXT-X-KEY tags and hence will required to be encrypted. To serve a mixture of encrypted and un-encrypted content, define two **<hls>** controllers on different ports (one with an id attribute, e.g. “hls-enc”) and enable encryption on only one of them. For secure key distribution it is very likely the one serving encrypted content will need to run over SSL (see below).

The URL template gives a template with interpolated variables $ASSET (the asset ID) and $LICENCE (see below). This URL needs to return a 16-byte binary key for the asset; either a static file or generated by a licence server such as Packet Ship Keyline.

Licence parameter in HLS URLs

The controller also supports a 'licence' parameter on the top-level playlist URL which is passed through to the EXT-X-KEY URL in the playlist. This enables secure delivery of a key to a client based on DRM restrictions implemented in the licence server.

Workround for Apple mis-handling of chunked encoding

The HLS controller now generates playlists with chunk length information to allow the file server to use explicit Content-Length headers rather than just relying on chunked encoding. This is to work round an apparent bug in Apple's player when using chunked encoding with encrypted content.

Support for SSL in HLS and HTTP

Apple's deployment guidelines for HLS specify that if the URL used to fetch a key in EXT-X-KEY tags is over HTTPS, then so must the top-level playlist be, using the same “authentication domain” (which we take to mean server certificate Common Name). The HLS and plain HTTP controllers now support this by setting `<ssl enabled="yes"/>` inside the controller definitions and also providing a server-wide SSL context definition.

SSL Context definition

An SSL context defines the server certificate and private key to be used for SSL connections and also any requirements for client-side certificate verification. The Packet Ship Streamline server has a single SSL context for all interfaces.

SSL is enabled with an **<ssl>** element at the top level of `streamd.cfg.xml`, with an **enabled** attribute set to '`yes`':


```
<ssl enabled="yes">

  <!-- Certificate in PEM format -->
  <certificate>
    -----BEGIN CERTIFICATE-----
    === Replace with valid PEM format certificate ===
    -----END CERTIFICATE-----
  </certificate>

  <!-- Private key in PEM format -->
  <private-key encrypted="yes">
    -----BEGIN RSA PRIVATE KEY-----
    === Replace with valid PEM format private key ===
    -----END RSA PRIVATE KEY-----
  </private-key>

  <!-- Certificate verification -->
  <verify enabled="yes" mandatory="yes">

    <!-- Root certificate locations -->
    <!-- If defaults is set, system default locations are also used -->
    <root defaults="no">

      <!-- Location of root certificate file (PEM format, multiple certs) -->
         <file>/etc/packetship/certs.pem</file>

         <!-- Location of root certificates directory (multiple PEM format
              files, named by hash of CN) -->
         <!-- <directory>/etc/packetship/certs</directory> -->

    </root>

  </verify>
</ssl>
```

The server certificate is defined inside a <certificate> element with PEM format text (including the BEGIN and END header and footer). The private key is defined with a **`<private-key>`** element similarly. If the private key is encrypted (requiring a pass-phrase), set an **`encrypted`** attribute on **`<private-key>`** to '`yes`'.

Note: If you use an encrypted private key the server will request the pass-phrase for it at startup. If you enter it wrong the server will still start but SSL will be disabled. Think very carefully about the effect this will have on your boot sequence if `ps-streamd` is automatically started!

Client certificate verification can be enabled with a **`<verify>`** element with **`enabled`** set to '`yes`'. If it is to be mandatory (clients must present a certificate) then also set **`mandatory`**.

The locations for root certificates to verify against is set by a **`<root>`** element inside **`<verify>`**. If this has its **`defaults`** attribute set to '`yes`' then the server will use the standard system root certificates, which will mean the client must present a certificate signed by a globally recognised Certificate Authority.

If you want to use additional root certificates – for example from your own CA or certificates issued by Packet Ship, you can define the certificate locations either as a single multi-certificate PEM file (**`<file>`** element inside **`<root>`**), or a CN-hashed directory of certificates as used by OpenSSL (**`<directory>`** element inside **`<root>`**). If you use your own certificate files with **`<root defaults="no"/>`** then global certificates will *not* be accepted.

Sub-playlist and chunk file URL prefixes

For performance reasons it is not sensible to serve the chunk files themselves over SSL, so the URLs for chunks in the playlist files need to redirected back to the non-SSL HLS server port. Normally the URLs issued in the playlist are server-relative absolute URLs (e.g. “`/asset?...`”). You can prefix

this with an absolute HTTP URL by setting a **<chunk prefix>** in the **<hls>** controller - for example:

```
<hls>
  …
  <chunk prefix="http://my-server-ip:8081"/>
 …
</hls>
```

This would result in chunk URLs in the playlist of the form:

`http://my-server-ip:8081/asset?...`

The same can be done for sub-playlists (variant playlists) by setting a **<playlist prefix>**. Note however that if the sub-playlist contains licence keys (as it will if the licence parameter is used) they should be kept secure; there is not much overhead to serving such small files over SSL.

Multiple controllers of the same type

In the previous version it was possible to create multiple controllers of the same type (e.g multiple RTSP or HLS on different port numbers with different configuration), but only the last one configured would be 'ticked' in order to handle timeouts etc. This has now been fixed.

Withdrawal of <offset rounding>

The `<offset rounding>` feature where seek offsets are forced to a fixed multiple (e.g. 188) has been removed. This was only relevant in older versions where seeking was done without an index file, which is no longer possible. It also breaks HLS streaming of encrypted content.

Handling BBC HD captures in ps-index-mpeg2ts

Additional H.264 syntax elements used in BBC HD broadcasts were not handled by the previous indexer, creating an “SPS containing Scaling Lists not supported ” error. The new version handles these and indexes BBC HD content successfully.

Handling audio streams in ps-index-mpeg2ts

The indexer can now handle producing VBR rate-pacing and seek offsets for audio streams (e.g. MP3 in TS), when using the `'-S'` or `'--seek-pcr-only'` flag.

Rate and duration output in ps-index-mpeg2ts

The indexer now outputs the rate and duration of the stream in the same way as the 2.3 indexer did:


```
$ ps-index-mpeg2ts /home/media/testcard.ts
Packet Ship MPEG2 Transport Stream indexer version 3.1.1
Indexing /home/media/testcard.ts into /home/media/testcard.ts.psi2
Output index format: PSI2
PMT on PID: 66
Detected MPEG-2 video stream on PID: 68
Creating index of 178 blocks
Duration:         99.3522   seconds
Average rate:     4347576   bits/sec
Maximum rate:     8888354   bits/sec
Minimum rate:     1624149   bits/sec
Done
```

The statistics are delimited with tabs so they can easily be parsed out with 'cut' – e.g.

```
$ ps-index-mpeg2ts /home/media/testcard.ts 2>&1 | grep "Duration:" |
cut -f2
99.3522
```

Average rate in ps-analyse-mpeg2ts

The average rate reported by '`ps-analyse-mpeg2ts rate`' has been synchronised with that reported by `ps-index-mpeg2ts`. The previous difference of a few bits/sec was due to different treatment of the bytes that precede the first PCR value in the stream.

