

Adobe® Primetime

Fault Tolerant Deployment for Live Streaming for HDS and JIT HLS

Contents

Fault Tolerant Deployment for Live Streaming for HDS and JIT HLS.....	3
Introduction.....	3
Layers in the live streaming pipeline.....	3
Encoding and packaging.....	3
Origin service.....	3
Content delivery network.....	4
Fault tolerant setup.....	4
Fault tolerance for HDS.....	5
Fault tolerance for live HLS converted just in time (JIT).....	7
Configuration for components.....	12
Setup configuration.....	12
Encoder configuration.....	13
Packager configuration.....	13
Origin configuration.....	14
Top-level manifest.....	14
Reverse proxy configuration.....	19
GTM requirements.....	20
Security considerations.....	20
RTMP authentication.....	20
Authentication at origin.....	20
Firewall protection at origin.....	21
Troubleshooting tips.....	21
Origin's access logs.....	21
4xx response from origin.....	22
High latency between packagers and origins.....	22
Overloaded origins.....	23

Fault Tolerant Deployment for Live Streaming for HDS and JIT HLS

Introduction

Use Adobe Primetime to deploy a reliable and fault-tolerant just in time (JIT) live streaming system for [HTTP Dynamic Streaming \(HDS\)](#) and [HTTP Live Streaming \(HLS\)](#).

The system comprises encoders, packagers, origin servers, reverse proxies, load balancers and content delivery network (CDN) deployed in distinct layers to form an event delivery pipeline.

Each layer contains a specific Primetime component to provide a distinct service. For example, the Packaging layer contains encoders and Packagers to provide encoding and packaging services.

To achieve resilience against faults, you should deploy components redundantly within each layer.

The system employs client-driven failover techniques along with CDN assisted traffic routing to handle long term failures, such as data center unavailability. The server system handles frequently occurring transient faults. These include server machine restarts, machine temporarily going out-of-network, process restarts, and network congestion.

The redundant setup and fault tolerant techniques used in the system help provide highly available and consistent HDS/HLS streams for a smooth video playback.

Layers in the live streaming pipeline

A live streaming pipeline comprises the following layers:

Encoding and packaging

This layer contains one or more encoders that ingest source content from multiple cameras and encode the content into multiple renditions at different bit rates. If the encode is Primetime-enabled and can package the content, you need not deploy packagers in this layer. If you deploy multiple encoders, synchronize them using a master LTC clock to ensure consistent stream output.

If your encoder doesn't have packaging capability, deploy Primetime Packager to package the encoded content. The packager divides the encoded media into fragments and maintains a manifest file, which is a playlist for the fragments.

The format of the fragments and the manifest file conform to HTTP delivery protocols, such as HDS, HLS, or MPEG-DASH. The manifest file may contain other custom metadata, for example, metadata to facilitate dynamic Advertising insertion, SMPTE time-codes for synchronization, and entitlement/DRM-metadata if the packaged content is encrypted and DRM-protected.

The packager pushes the fragments to the origin server using HTTP PUT request.

Origin service

This layer includes origin servers (usually co-located), reverse proxies, and load balancer(s).

The origin server receives the fragments from the packager and delivers them to players through a CDN.

The origin server can store the fragments for a while. In this case, it modifies the entries in the manifest file to indicate the fragments that are available with it and the corresponding metadata. You can also use Primetime Origin server to convert the packaged content from HDS to HLS format just-in-time while delivering it to HLS clients.

The origin service layer also includes a reverse proxy. It is a stateless server that serves as an endpoint for player requests. It can cache the response of HTTP GET request to reduce the load on the origin server.

Reverse proxy implements failover logic to ensure high availability of the packaged content. For example it uses the 503-failover scheme to serve a fragment request if it is available with any origin server.

A load balancer distributes the client load between the reverse proxies. It must have a public IP address that is used to construct rendition stream URLs, which are listed in the top-level manifest.

Content delivery network

The CDN pulls content from the Origin Service layer and caches it. In addition, it delivers the content to players. CDN is also capable of traffic rerouting, which helps in disaster recovery and load-balancing across multiple origin sites.

For example, [Global Traffic Management](#) (GTM) is a service provided by Akamai. This functionality is referred to as GTM.

Fault tolerant setup

You can deploy sample fault tolerant setup for a live streaming pipeline that demonstrates fault resilience features. You can deploy a fault tolerant setup in various ways. For example, you can include Origin and Packaging functionality in a single data center. Alternatively, you can host Origin servers in the cloud, use multiple Packaging sites, or eliminate packagers and use Encoders to package the content.

For the sample setup, you can include two redundantly deployed data centers for Origin service and a single data center for Packaging. Each Origin data center should have two redundantly deployed origin servers, multiple reverse proxies and a load balancer. The packaging center can contain two redundantly deployed packagers and one or more encoders.

The sample deployment scenario has the following features:

- Encoders can be configured to publish the stream in MPEG2TS format or through RTMP to the packagers. If the encoders publish the stream on a multicast port in MPEG2TS format, the encoders and packagers must be in the same subnet. However if the encoder publishes RTMP stream, the encoders and packagers can be in separate subnets.
- Encoder(s) are configured to generate multiple bitrates of each ingested stream.
- For each bitrate, you configure a single stream module on both the packagers.
- For each bitrate, you can configure a single stream module on each origin. This module receives the packaged content from both the packagers.

This setup requires the following public IP addresses:

- GTM IP address

- IP of the load balancer of Origin Data Center 1
- IP of the load balancer of Origin Data Center 2

Top-level manifest requests are routed through GTM IP. All the rendition stream level manifest requests are served directly through the Load Balancer IP addresses.

You can configure GTM to load balance between origin data centers where set-level manifest requests are routed to different sites in a fixed ratio, for example 50:50.

If you designate data-center2 as a backup for the primary data center in case of a failure, configure GTM to route the requests in the ratio 100:0. Based on liveness checks, when GTM determines that data-center1 is not responsive, it can switch-over to the backup site by redirecting all new set-level manifests to it. The existing clients fail-over to the URLs of the data-center2 when the primary data center cannot fulfill stream level manifest requests.

Fault tolerance for HDS

The sample live streaming pipeline provides HDS fault tolerance using Best Effort Fetch and 503 failover, described in [Rock Solid Live Streaming](#). 503 Failover is a proxy configuration technique that is used to serve a fragment request when the fragment is available with any origin server. This functionality is achieved by the reverse proxy logic where the request is routed to a different origin server when the first origin server responds with 503 indicating fragment unavailability. You use this technique with the Best Effort Fetch (BEF) technique to mitigate the effect of transient errors. A BEF-enabled player requests a fragment even if it is not yet listed in the HDS bootstrap (liveness issue) or marked as dropped.

You can enable the BEF feature by adding the `<bestEffortFetchInfo>` element in the set-level HDS manifest (F4M):

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns="http://ns.adobe.com/f4m/2.0">
  <bestEffortFetchInfo segmentDuration="4" fragmentDuration="4"/>
  ....
</manifest>
```

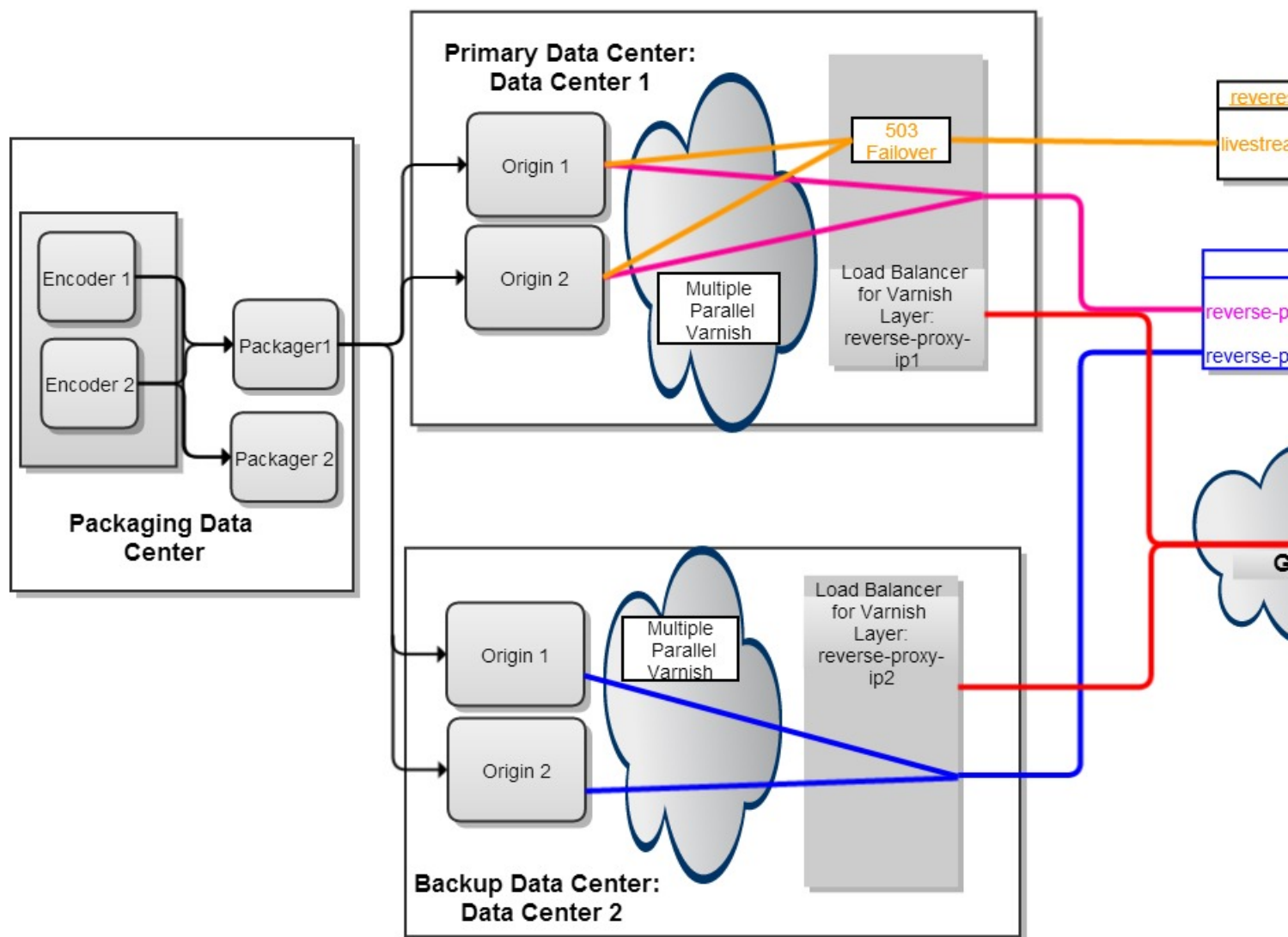


Figure 1: HDS fault tolerant setup

The setup balances the stream load across all origin servers in a data center by the reverse-proxies that route the requests in a round-robin manner. The 503 failover technique also provides fault tolerance as shown in Figure 2.

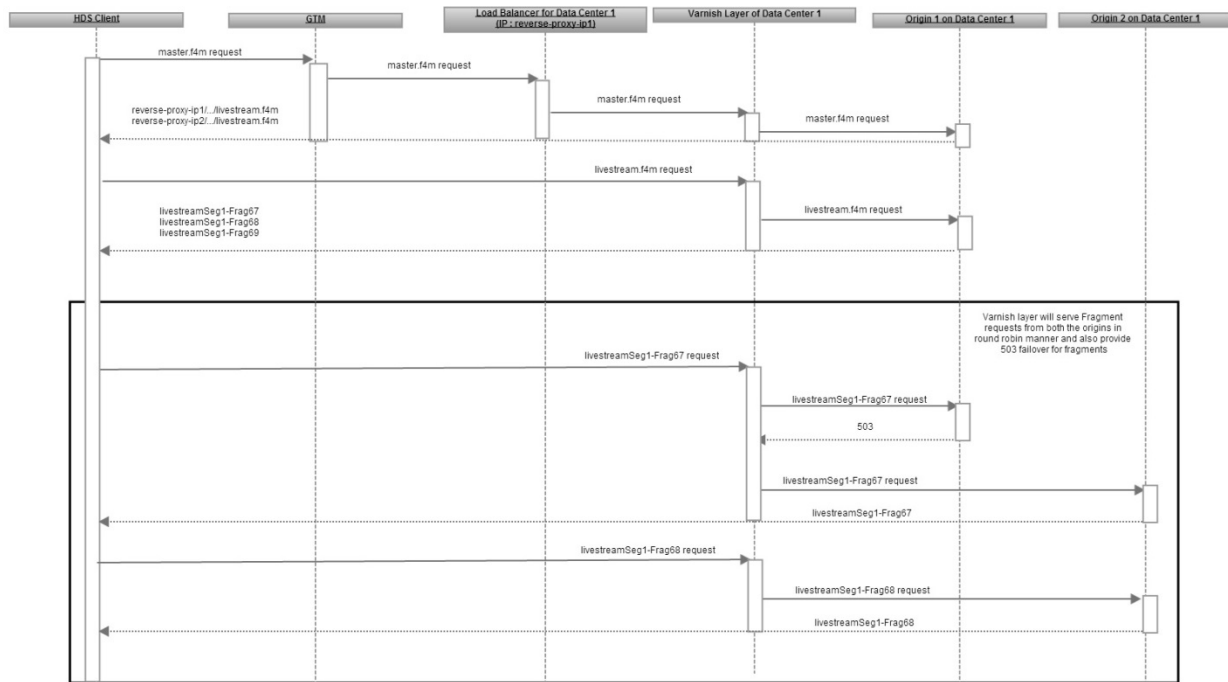


Figure 2: HDS failover sequence diagram

GTM enables load balancing and fault tolerance across data centers by redirecting the set-level manifest requests to either data center in a round robin fashion. For optimal load balancing, you should set the TTL (max-age) for set-level manifest to a just a few seconds. A low value ensures that the set-level manifest is not cached in the downstream HTTP infrastructure for too long and is sent to a significant number of clients. Set the value of TTL of the set level manifest to a few hours if you do not require load balancing and the second data center serves as backup as shown in Figure 1.

The set-level F4M at origin servers in data center 1 includes the URL that contains the IP of the load-balancer in data center 1. The IP is listed before the URL of the stream of the other data center stream. You can achieve this by using the <BaseURL> construct of F4M (version 2.0) or listing each rendition stream in multiple <AdaptiveSet> elements, as specified in [Adobe Media Manifest \(F4M\)](#). The <AdaptiveSet> corresponding to data center1 is listed first in the set-level F4M at data center1. In contrast, the reverse is true for set-level F4M at data center 2.

Fault tolerance for live HLS converted just in time (JIT)

Unlike HDS, the segment URLs are listed in the HLS manifest and cannot be predicted using a pre-defined template. Therefore, it is difficult to support BEF at client side where the URL of a missing fragment must be constructed at the player. Another HLS requirement that makes client-side BEF support impossible is that a gap or a fragment, once advertised in the HLS manifest, cannot be removed in a later refresh operation. Primetime Origin provides fault tolerance for HLS created just-in-time from HDS content pushed from Packagers/Encoders. It utilizes server side redundancy to implement healing from transient failures. Consider a scenario where the advertised manifest reflects the global state of the system by including fragments that may be present on another origin server. In this case, the client can obtain the advertised fragments from any origin server through the reverse proxy using the 503 failover technique.

Each origin server maintains a copy of the M3U8 (HLS manifest). It is updated not on the basis of fragments received. Instead, it is updated by extending it with any subsequent fragment entries in the F4M (HDS manifest), generated

at and pushed from the packager. You perform this operation consistently so that only the subsequent fragment-URLs are added. Updating this way makes the manifest consistent across client-requests and the origin server need not guess the URLs of missing fragments. If an origin server is unavailable for some time, it is able to recover the missing fragment entries from the packager's manifest. The system also handles a transient fault at packager seamlessly because each origin receives manifests and fragments from multiple packagers.

If the source F4M, which is pushed by a packager, contains a gap (missing fragments) near the live edge, the origin will not update its manifest immediately to include the gap. Instead, it will wait for the next manifest push from a packager. This helps when the origin receives a manifest from another packager that has fragment entries instead of the gap. Even if the packagers are out-of-sync by a few fragments, the fragments dropped by the leading packager do not result in gap entries in the M3U8 (created just-in-time at origin) if another packager creates the fragments. Figure 3 illustrates how adaptive wait, while updating M3U8 just-in-time, mitigates occasional fragment drops at packagers.

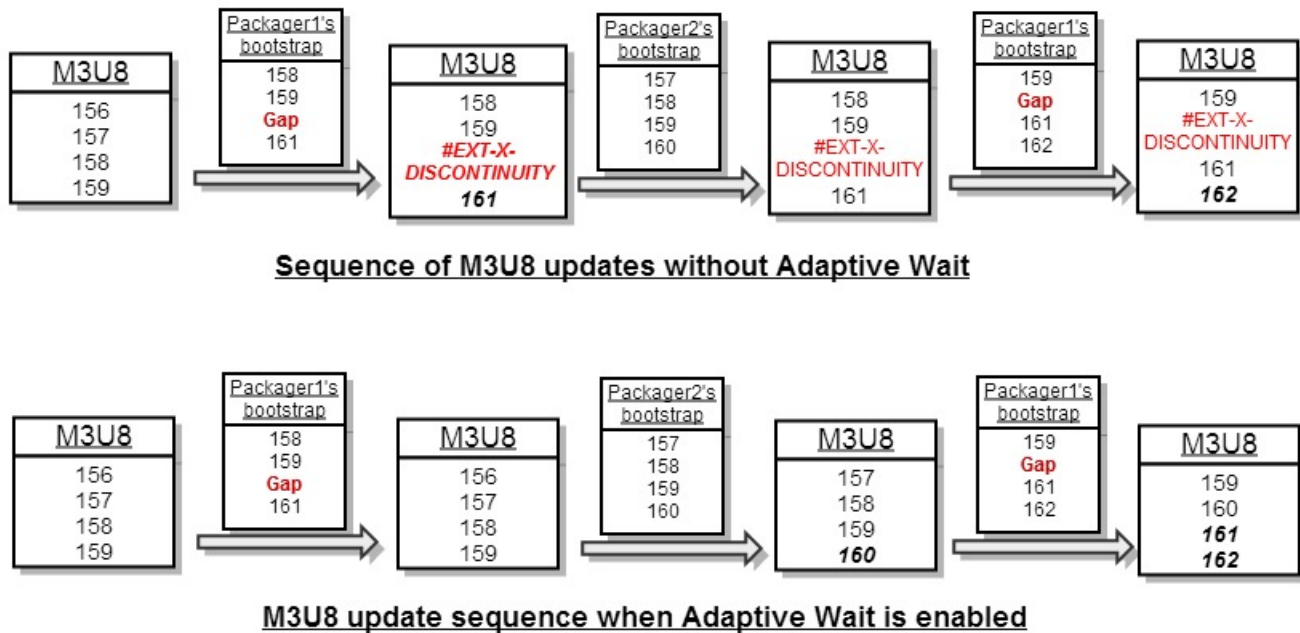


Figure 3: Adaptive wait to fill missing fragment for JIT HLS

To ensure the consistency of HLS manifest, 503 failover is enabled for HLS fragments only and not for stream level M3U8. For the sample setup (Figure 4), the top-level manifest or master m3u8 contains a URL corresponding to each origin server. To avoid client driven failover, you configure the reverse proxy (varnish cache) with a grace period to allow for the serving of stale m3u8 for a few seconds when the serving origin server goes down or is out-of-network.

This scheme doesn't handle all scenarios where multiple transient faults occur simultaneously. For example, the system may reject a request for Fragment3 with a 404 error when Origin1 is down when a fragment request is served and Origin2 is down when the packagers are pushing Fragment3.

Because the probability of the occurrence of a fault is low, we assume that the probability of two or more faults occurring simultaneously will be negligible. Client-side-failover is employed to handle faults that are not recovered at the origin layer.

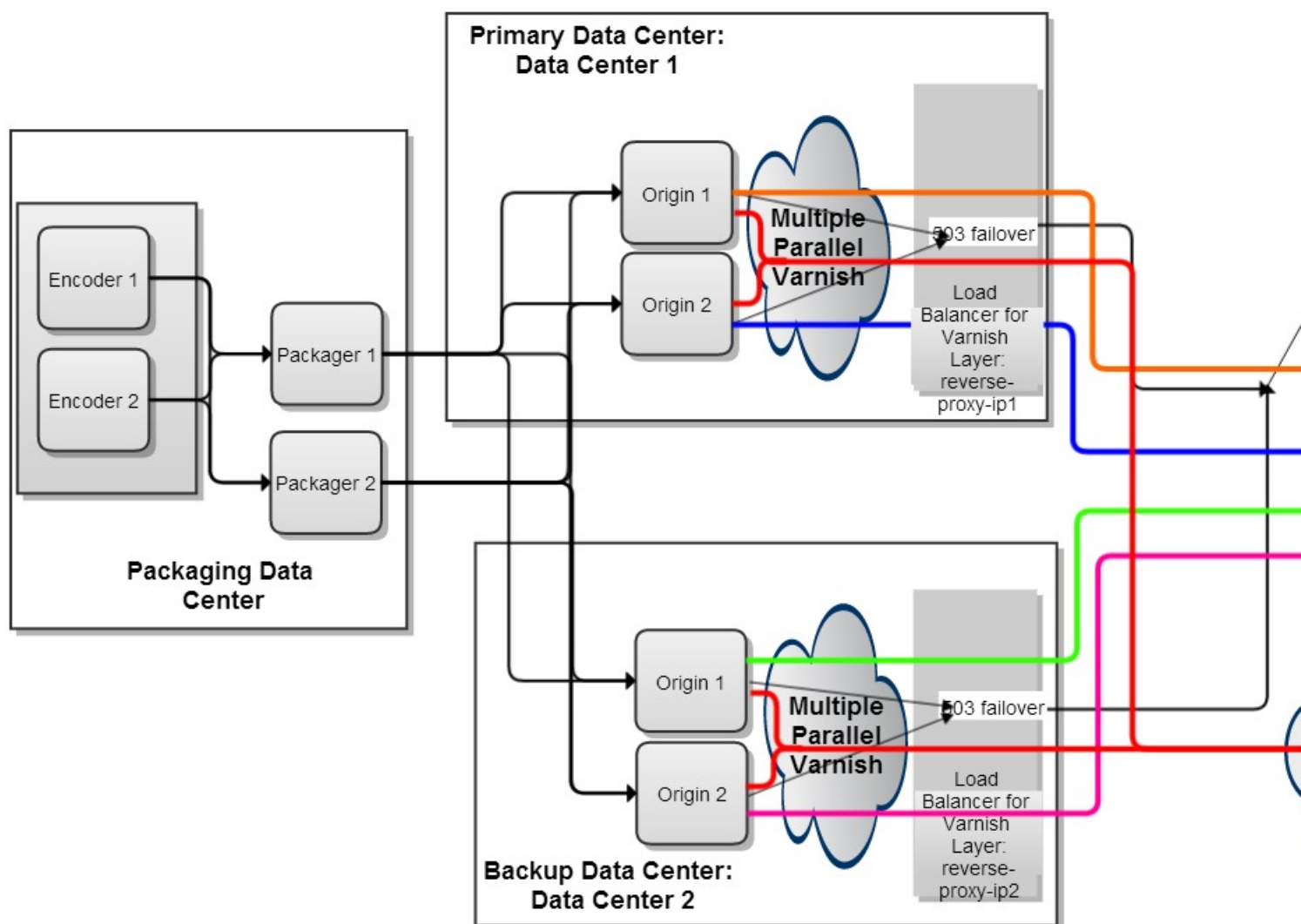


Figure 4: Fault tolerant setup for JIT HLS

Figures 5, 6 and 7 describe the sequence flow for various failure scenarios and how 503 failover, data-center failover via GTM, and client-driven failover work together to achieve uninterrupted and seamless HLS playback despite faults.

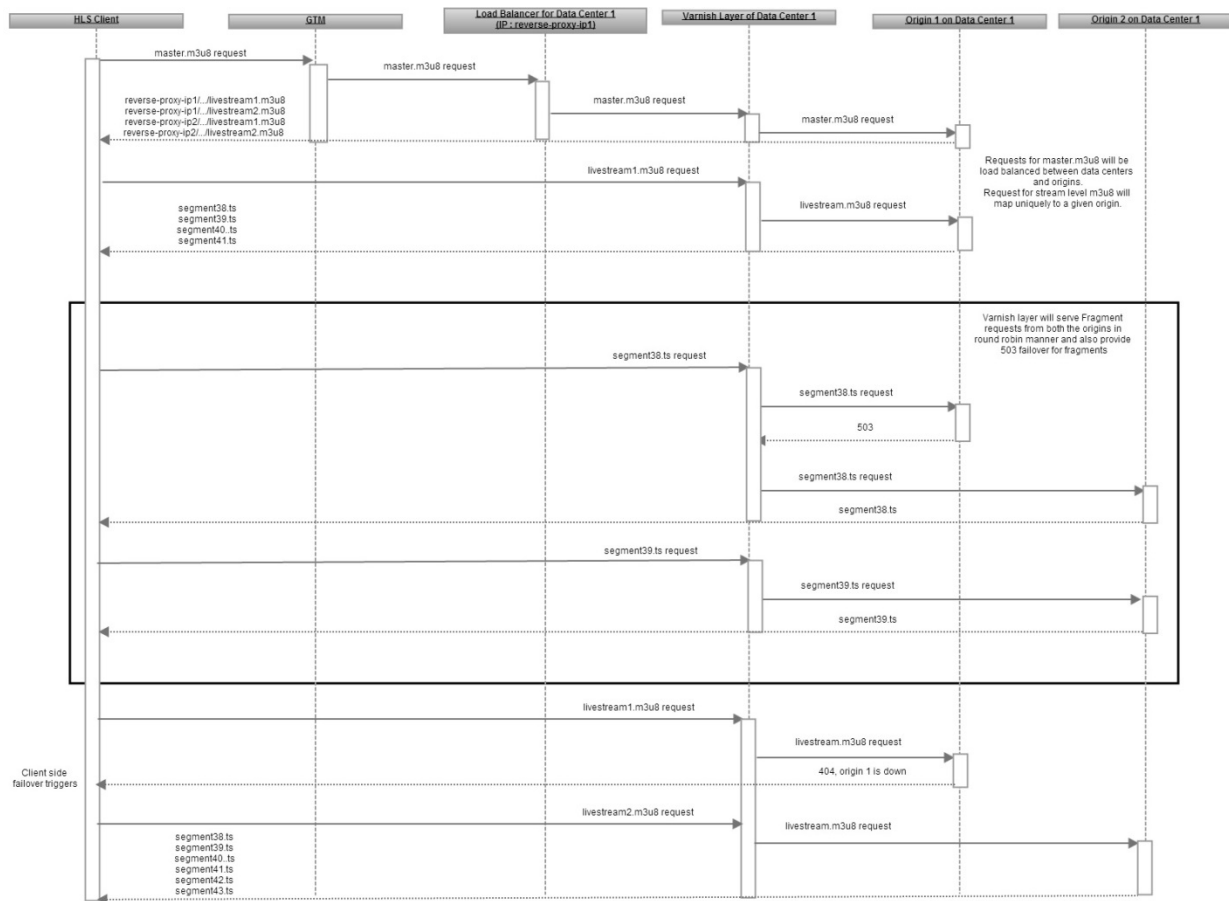


Figure 5: JIT HLS fault tolerance within the same data center

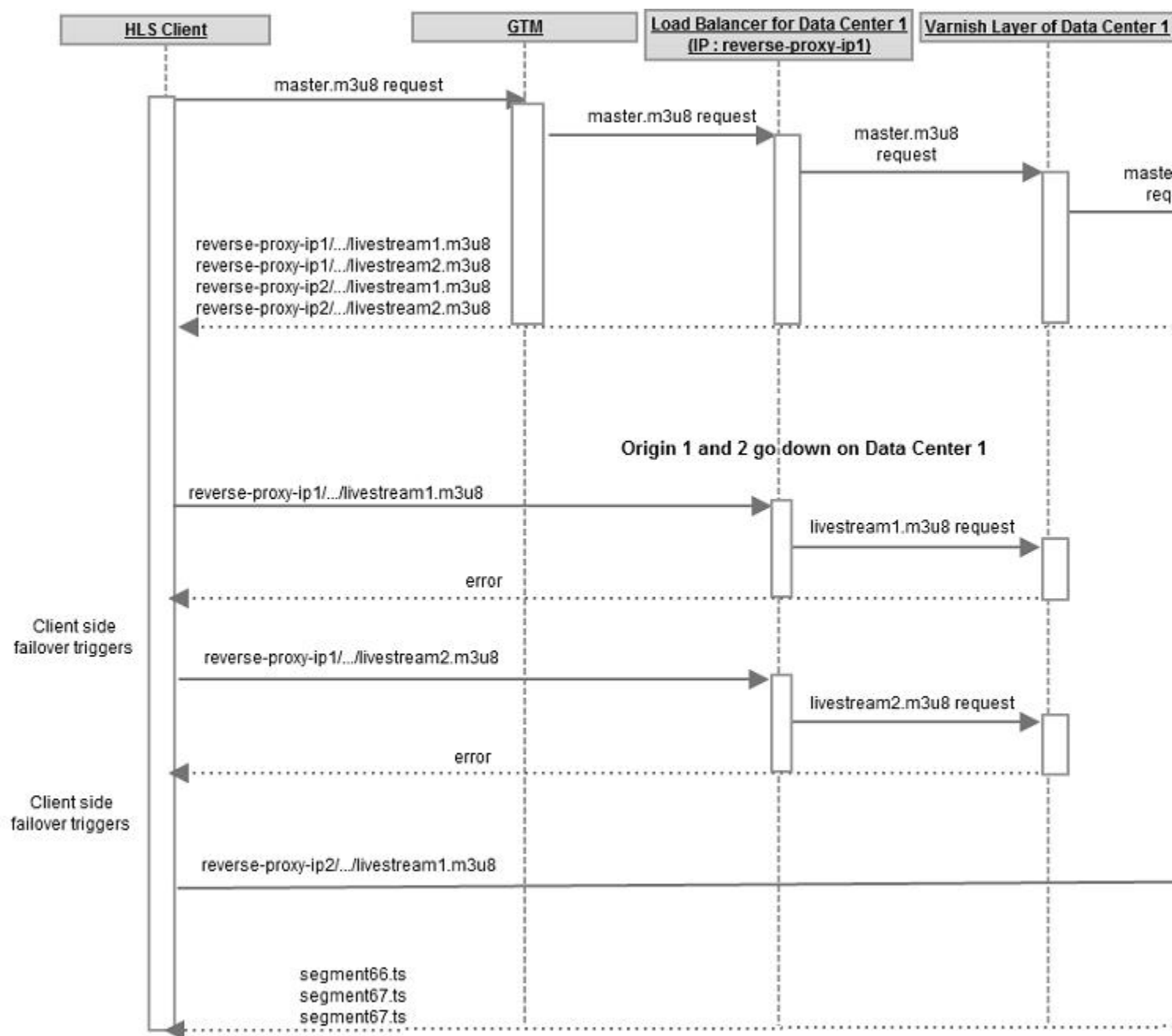


Figure 6: JIT HLS failover across data centers

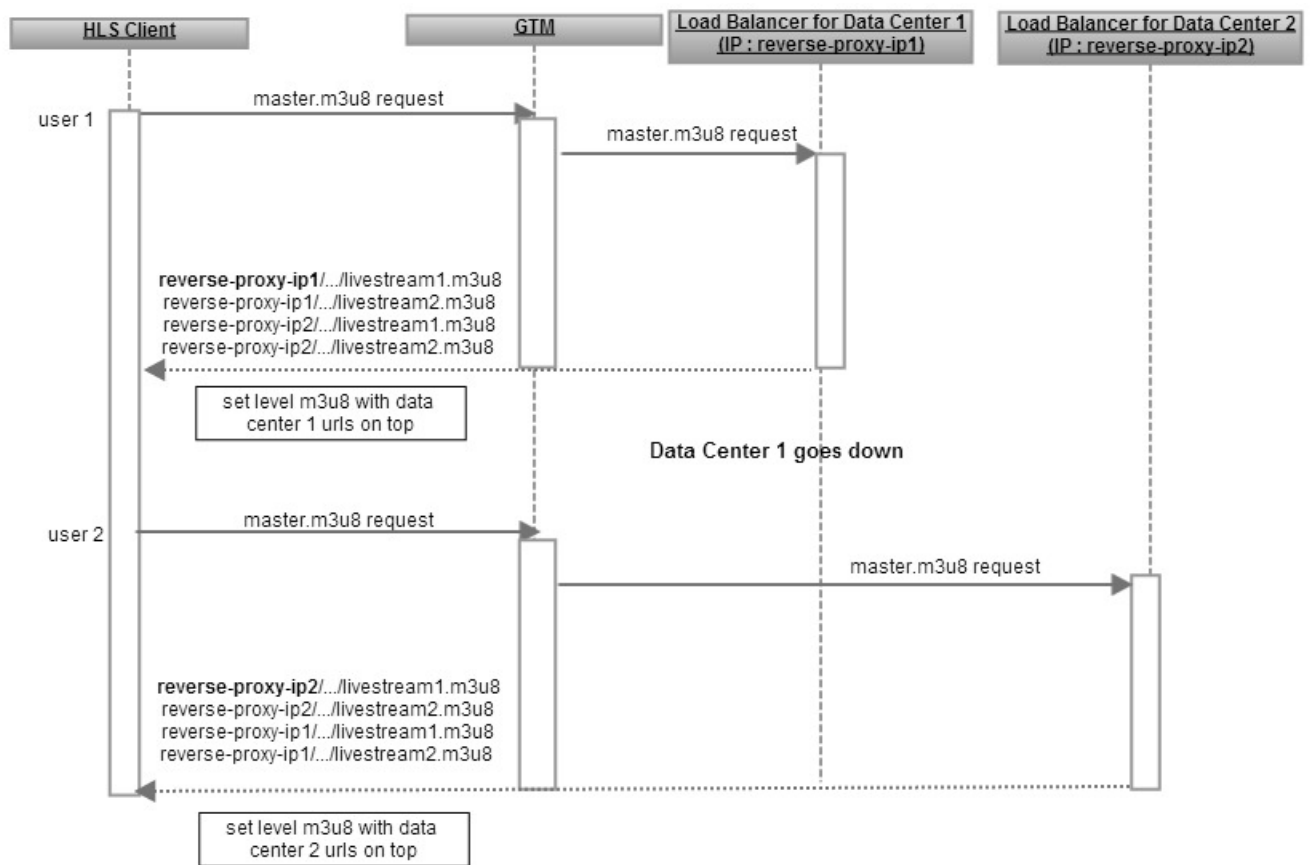


Figure 7: MasterHLS manifest (m3u8) failover by GTM

Configuration for components

To support multi bitrate video streams or alternate language audio-only streams, you should replicate the configuration for the sample setup for each rendition. You should list the rendition streams in each set-level manifest.

Setup configuration

The following are the configuration details of the sample setup:

- Single bitrate stream being multicast by encoder at address `multicast_add:multicast_port`
- Origin 1 on Primary Data Center listening at `primary-origin1:port`
- Origin 2 on Primary Data Center listening at `primary-origin2:port`
- Origin 1 on Backup Data Center listening at `backup-origin1:port`
- Origin 2 on Backup Data Center listening at `backup-origin2:port`
- Reverse Proxy of Primary Data Center listening at `reverse-proxy-ip1`
- Reverse Proxy of Backup Data Center listening at `reverse-proxy-ip2`

Encoder configuration

You can configure the encoder to generate streams through an RTMP connection to the packagers or in MPEG2TS format at a multicast IP address. To be compliant with the HDS protocol, the stream time should not go back in time. Therefore, you should configure the encoder to use Absolute time to avoid stream time getting reset on encoder restart while publishing the stream.

If you use multiple encoders, they must have identical configuration for the stream. Specifically, the encoding parameters including keyframe interval and bitrate must be same. They must generate identical timestamps for the same content. To achieve this, you should configure them to derive timestamps from a same LTC source.

Packager configuration

Both the packagers should have the same stream module replicated. You configure this stream module to push the packaged HDS content to each origin server in both the data centers. Set the value of <SkipBadFragment> to true so that a fragment that contains corrupted or missing data is dropped. In addition, a duplicate fragment is generated at another packager and served and cached downstream. For example, set Fragment-duration to 6 seconds. This should be equal to the key-frame-interval or a small multiple of it when the encoder generates video stream with a fixed key-frame interval. If you do not set the value of <TargetDuration>, the system assumes the default value to be about twice the Fragment-duration while creating M3U8 just-in-time at origin.

stream.xml

```
<Stream>
  <FragmentDuration>6000</FragmentDuration>
  <TargetDuration>10000</TargetDuration>
  <Input>
    <TSMulticast>
      <MulticastAddress>multicast_add</MulticastAddress>
      <MulticastPort>multicast_port</MulticastPort>
    </TSMulticast>
  </Input>

  <OutputPipeline>
    <OutputType>HDS</OutputType>
    <DiskManagementDuration>3.0</DiskManagementDuration>
    <SkipBadFragments>true</SkipBadFragments>

    <Output>
      <HttpPush>
        <TargetURL>http://primary-origin1:port/eventX/bitrate_1</TargetURL>
      </HttpPush>

      <HttpPush>
        <TargetURL>http://primary-origin2:port/eventX/bitrate_1</TargetURL>
      </HttpPush>

      <HttpPush>
        <TargetURL>http://backup-origin1:port/eventX/bitrate_1</TargetURL>
      </HttpPush>

      <HttpPush>
        <TargetURL>http://backup-origin2:port/eventX/bitrate_1</TargetURL>
      </HttpPush>
    </Output>
  </OutputPipeline>
</Stream>
```

Origin configuration

You should replicate the same configuration at each of the four origins.

Stream module configuration

```
/conf
  /containers
    /eventX
      /container.xml
      /streams
        /bitrate_1
          /stream.xml
```

stream.xml

```
<Config>
  <DiskManagementDuration>3.0</DiskManagementDuration>
  <HttpConfig>
    <Headers>
      <TTL>
        <M3U8>5</M3U8>
      </TTL>
    </Headers>
  </HttpConfig>
  <FaultTolerance>
    <MultiServerManifestSyncEnabled>true</MultiServerManifestSyncEnabled>
    <MaxLagFromLiveEdge>20</MaxLagFromLiveEdge>
    <ConstantLagFromLiveEdge>0</ConstantLagFromLiveEdge>
  </FaultTolerance>
</Config>
```

Top-level manifest

Set level f4m

The Set level f4m should contain two URLs for each rendition. The first URL should point to the Load Balancer of the Primary Data Center (reverse-proxy-ip1). The second URL should point to the Load Balancer of the Backup Data Center (reverse-proxy-ip2). The Load Balancer at each data center sends stream-level f4m requests to the origin servers of the corresponding Data Center in a round-robin manner. Unlike stream-level m3u8 requests that must have a sticky session, you can safely route stream-level f4m requests to various origin servers in a data center in a round robin manner.

TTL setting for f4m

You can set the value of TTL for set-level f4m to a high value (a few hours). However, ensure that the TTL for the stream-level f4m is low, preferably half the duration of the fragment. For a stream with fragment duration four seconds, stream level f4m should have a TTL of two seconds. You should set the TTL for set-level f4m to a low value (for example, half the fragment duration) when the client-load must be divided equally or in a pre-defined ratio between the data centers. You configure this ratio at GTM.

If the set-level f4m is served from the static folder at the origin server, you can set TTL in the origin.xml. When set level f4m resides in a container level, you can set TTL in container.xml.

Any TTL that you set for f4m in the origin.xml file applies to all the f4m requests served from any of the modules of the origin server. To achieve a lower TTL for stream-level f4m, configure the TTL for the stream-level f4m in the stream module's configuration. Alternatively, modify the varnish script to configure set-level f4m TTL.

Master m3u8

Each origin server should host a different set-level manifest. The individual stream URLs inside the set level manifest remain the same on all origin servers.

The following is the prescribed order of m3u8 URLs for each bitrate:

1. URL pointing to the same origin server
2. URL pointing to the other origin server of the same data center
3. URLs of origin servers in a different data center in any order

If the set-level m3u8 contains four redundant URLs for the same bitrate, the iOS client picks and fails-over for these streams in the order they are listed. If all set-level manifests list the same first URL, the client load is directed to the same origin server. To distribute the load, ensure that each origin server lists the redundant URLs in a different order in set level m3u8.

The stream name in the m3u8 URL is in the format livestreamN. The stream name hosted by each origin server is "livestream". To differentiate between m3u8 requests for origin servers (in the same data center), modify the stream name in the set-level m3u8. The varnish script can create the following mapping with the modified stream name:

livestream1 => origin 1

livestream2 => origin 2

The system uses the integral suffix attached to the stream name to redirect stream level manifest requests to the appropriate origin. Therefore, ensure that the set-level m3u8 name does not use any suffix that is used by the stream-level m3u8. For example, in above case, the set level m3u8 name must not end with 1.3u8 or 2.m3u8.

In the sample setup, the following should be the top-level or master m3u8 for each origin:

Data Center 1	Origin 1	<pre>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_1/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_1/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_1/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_1/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_2/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_2/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_2/livestream1.m3u8</pre>
---------------	----------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_2/livestream2.m3u8</p> <p>..</p> <p>..</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_N/livestream1.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_N/livestream2.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_N/livestream1.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_N/livestream2.m3u8</p>
Origin 2	<p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_1/livestream2.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_1/livestream1.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_1/livestream1.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_1/livestream2.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_2/livestream2.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_2/livestream1.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_2/livestream1.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_2/livestream2.m3u8</p> <p>..</p> <p>..</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_N/livestream2.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_N/livestream1.m3u8</p> <p>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_N/livestream1.m3u8</p>

		#EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_N/livestream2.m3u8
Data Center 2	Origin 1	#EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_1/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_1/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_1/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_1/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_2/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_2/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_2/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_2/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_N/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_N/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_N/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_N/livestream2.m3u8
	Origin 2	#EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_1/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_1/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_1/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_1,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_1/livestream2.m3u8

	<pre>#EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_2/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_2/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_2/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_2,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_2/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_N/livestream2.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip2/eventX/bitrate_N/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_N/livestream1.m3u8 #EXT-X-STREAM-INF:BANDWIDTH=bitrate_N,PROGRAM-ID=1 reverse-proxy-ip1/eventX/bitrate_N/livestream2.m3u8</pre>
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

TTL setting for m3u8

The TTL for the master m3u8 should be low to enable clients to receive master m3u8 from different Origins. If the static folder in the Origin serves the top-level manifest, TTL can be set in the origin.xml file. Any TTL that you set for m3u8 in the origin.xml file applies to all the m3u8 requests served from any of the modules of the Origin.

If you require a different TTL for set-level and stream-level m3u8, configure the TTL for stream-level m3u8 in the stream module. Alternatively, modify the set-level m3u8 TTL in the varnish script. If the static folder of the origin serves the set-level m3u8, use the following configuration to set the TTL for the master m3u8 to 10 seconds and the TTL for stream-level m3u8 to five seconds:

origin.xml

```
<Config>

  <!-- Interface and Port for the HTTP server to listen on. -->
  <InterfaceAddress></InterfaceAddress>
  <Port>8090</Port>
  ....
  ....
  ....

  <!-- Specifies TTL values in integral seconds for various file types. When
  TTL value is not set for a file type, then following headers are set
  in Http Response of corresponding file type: 1) max-age header - set to
  TTL value
  2) Date header - set to response date 3) Expires header - set to response
  date + TTL The file types supported are F4F,F4M,Bootstrap,CrossDomain,
  M3U8, HLSSegment, DRMMetadata, Timeline, DashSegment, DashManifest
  and DashManifestPart.
  -->
```

```

    <Headers>

    <TTL>
    <M3U8>10</M3U8>
    </TTL>

    </Headers>
</Config>

```

The following are sample contents of the stream.xml file for each of the configured stream modules:

```

<Config>
    ....
    ....
    <HttpConfig>

    <Headers>
    <TTL>
    <M3U8>5</M3U8>
    </TTL>
    </Headers>

    </HttpConfig>
    ....
</Config>

```

Reverse proxy configuration

In the sample setup, [Varnish cache](#) is used as the reverse proxy.

VCL requirements

A sample VCL Script, to support Fault Tolerance workflows is available with the origin server at `/Origin_install/sample_scripts/varnish_sample/default.vcl`

This vcl script performs the following functions:

1. Routes the stream-level m3u8 request to the correct origin using the origin number suffix attached to the stream name. In other words, it maps livestream1 to origin 1 and livestream2 to origin 2. To implement this, selecting the appropriate backend in `vcl_recv` subroutine and modify the outgoing URL to remove the extra qualifier in `vcl_miss` subroutine:

```

vcl_recv:
    if (req.url ~ "1\.m3u8")
    {
        set req.backend = b1;
    }
    elseif (req.url ~ "2\.m3u8")
    {
        set req.backend = b2;
    }
vcl_miss:
    if (req.url ~ "1\.m3u8") {
        set bereq.url = regsub(req.url, "1\.m3u8", "\.m3u8");
    }
    elseif (req.url ~ "2\.m3u8") {
        set bereq.url = regsub(req.url, "2\.m3u8", "\.m3u8");
    }
1.}

```

2. Routes the set level m3u8 requests to the origin servers in the data center in a round robin manner.
3. Routes the set-level and stream-level f4m requests to the origins in the data center in a round robin manner.

4. Provides 503 failover of requests for HDS/HLS fragments and HDS manifest between the origin servers within a data center.

GTM requirements

The following are the functional requirements of GTM or an equivalent feature within CDN:

- GTM provides one public IP address. The system arbitrarily routes a request with the IP address as destination to either of the two (or more) public IP addresses.
- You can configure the ratio of traffic between the IP addresses at GTM. For example, you can achieve a fair load balancing across different sites with a ratio of 50:50.
- GTM actively monitors the IP addresses for responsiveness through a liveness object/URL. If it discovers that a particular site is unavailable, it redirects the whole traffic to the active site, irrespective of the configured ratio.
- GTM IP address is always available. It is never unresponsive.

Security considerations

RTMP authentication

Primetime Packager 1.2 (and earlier versions) do not support authentication for RTMP stream ingest. Ensure that the packager's RTMP port is not visible externally outside the firewall. Otherwise, the system may publish a rogue RTMP stream.

Authentication at origin

You should configure origin servers, packagers, and encoders to use a shared secret for the security token. If the workflow involves the encoder calling HTTP DELETE on origin for older fragments, you must set the <DeleteSecurityToken> element on the origin servers and the encoder. You must provide the same AES-128 key under Auth/SecurityToken/Key at the origins and packagers.

The following is a sample configuration snippet of stream.xml for a packager module:

```
....
    <HttpPush>
        <UseSecurityToken>true</UseSecurityToken>
        <SecurityTokenKey>4ff4756ed68239d34d482dbc88819abc</SecurityTokenKey>
        <TargetURL>http://primary-origin1:port/eventX/bitrate_1</TargetURL>
    </HttpPush>
....
```

Here is the corresponding security token configuration in origin's stream.xml file:

```
<Config>

  <!-- (Optional) Auth security applied to processing inbound PUT/POST/DELETE requests. -->
  <Auth>
    <!-- If the 'SecurityToken' element is present, then an auth test is applied
         to all inbound PUT/POST requests that will check for a valid X-Adobe-HDS
         -Token or X-Adobe-HTTP-Token request header with a valid token value.
         Requests not containing this header or passing an invalid or
         timed-out token value receive an error response of 401 Not Authorized.
    -->
```

```

        If the <SecurityToken> element is not present, this
        check is not performed. -->

<SecurityToken>
  <!-- If <SecurityToken> element is enabled, it requires a valid AES-128
        key to use for Token value encryption/decryption, and only Packager(s)
        configures with matching key can write to this Origin. -->
  <Key>4ff4756ed68239d34d482dbc88819abc</Key>
  <!-- Optional 'Timeout' for Token values in seconds. Default value is
        600 seconds (10 minutes). If enabled, this value needs to be applied
        consistently here as well as at the Packager. -->

<Timeout>600</Timeout>

</SecurityToken>

  <!-- If the 'DeleteSecurityToken' element is present, then an auth test is
        applied to all inbound DELETE requests that will check for a valid X
        -Adobe-HDS-Token or X-Adobe-HTTP-Token request header with a valid token
        value. Requests not containing this header or passing an invalid or
        timed-out token value receive an error response of 401 Not Authorized.
        If the <DeleteSecurityToken> element is not present, this check is not
        performed. -->
  <!--
  <DeleteSecurityToken>

    If <DeleteSecurityToken> element is enabled, it requires a valid AES-128
    key to use for Token value encryption/decryption, and only clients
    configured with matching key can delete from this Origin.
    <Key>4ff4756ed68239d34d482dbc88819abc</Key> -->

    <!-- Optional 'Timeout' for Token values in seconds. Default value is
        600 seconds (10 minutes). If enabled, this value needs to be applied
        consistently here as well as at the Packager. -->
    <!-- <Timeout>600</Timeout> -->

  <!-- </DeleteSecurityToken> -->
</Auth>
</Config>

```

Firewall protection at origin

When the packagers and origin servers are located in different data centers, you should secure origin servers from external attacks. If the IP address and the origin server port are publicly accessible, protect the origin server with at least the following firewall settings:

- Allow traffic only for the origin server's port as the destination.
- Permit only white-listed or known packager IP addresses to send requests.

Troubleshooting tips

Origin's access logs

Origin server logs each HTTP request in the access.log file under <Origin install>/logs/http folder. Each entry in the file comprises the following fields:

- Local Date and Time on the Server

- HTTP verb : GET/PUT/POST/DELETE
- IP address and port of the remote client
- Requested URL
- Response code
- Request Content Length
- Response Content Length
- Time taken to serve out the request in milliseconds

Access logs usually provide good insight into most streaming issues, for example fragments not reaching origin or rejected with 4xx code.

4xx response from origin

401 Unauthorized

The origin responds with a 401 error to a request when Auth/SecurityToken is configured at the origin server and the request contains an invalid or expired token due to the following reasons:

- Packager is not configured with same SecurityToken as on origin.
- The machines running the packager and the origin are not time-synchronized. The time difference exceeds the value of SecurityToken/Timeout set at origin.

403 Forbidden

Origin server responds with a 403 error to a request when:

- The request can overwrite a file that is not accessible or another folder exists with the same name.
- There is no origin module configured to handle the request.

409 Conflict

This type of error occurs when the origin server is unable to accept the pushed HDS fragment due to its current state. For example, such errors occur when the request attempts to push content with a reverse timestamp to the origin server.

To eliminate chances of the occurrence of this type of error, do the following:

- Verify that the encoder maintains the stream time on restart if the content is generated at the encoder.
- Sometimes when you reset the Packager state in preparation for another event, but the corresponding origin module may still contain the fragments and manifest from the previous session. In this case, the reset origin state too.

High latency between packagers and origins

A high round trip time between the packager and origin networks causes high push latency. Associated symptoms could be:

- **Fragments dropping at the packager:**

By default, the packager is configured to push fragments to one origin server in a serial manner. If the average rate at which the fragments are pushed goes below the rate at which they are generated, the queue size continues to increase at the packager. This occasionally causes fragments to be dropped when it reaches the threshold maximum (default 5).

- **Out-of-order delivery**

Pushing smaller fragments usually take less time. Therefore, the origin server finds the fragments arriving out of order for a single stream from the same packager.

The stream with the highest bitrate displays these symptoms in a severe form. Streams with lower bitrates rarely exhibit these symptoms.

To resolve this problem, you should make one or more of the following configuration changes in the packager and origin and tune the origin machine for larger TCP window size.

Parallel push at packager

Set the configuration element `//Stream/OutputPipeline/Output/HttpPush/MaxConcurrentPushCount` in the `stream.xml` file to a value higher than 1. For example, set it to 3 or 5, depending on the severity of the problem observed. The default value is 1, which indicates that all fragment pushes are serialized.

TCP tuning of the origin server

To increase it to 4MB, run the following command in a live system (with root privileges)

```
echo 4194304 > /proc/sys/net/core/rmem_max
```

Alternatively, add/update the following line in `/etc/sysctl.conf` and reboot:

```
net.core.rmem_max = 4194304
```

The Linux kernel has a limit of 128KB on the TCP receive buffer size. This is sufficient to utilize the network bandwidth for most scenarios. However, it is not sufficient when the round trip time is unusually high. For a large fragment, the packager can push data at most one TCP window size at a time. It must then wait for an ACK message before sending more data. Because end-to-end latency is high, pushing the whole fragment takes lot of time. You can optimize this by increasing the maximum limit on the TCP receive buffer size at the origin server.

Enable constant lag for JIT HLS

If you observe that the HDS fragments arrive out-of-order at the origin server, set `//Config/FaultTolerance/ConstantLagFromLiveEdge` in the `stream.xml` file to the maximum amount of time (in seconds) for which a fragment can fall out-of-order. This causes the live edge of the just-in-time HLS stream to lag behind the live edge of the source HDS by the configured time. This should resolve any problem in just-in-time HLS due to out-of-order fragment pushes.

Overloaded origins

Capacity planning may be required if you observe one or more of the following symptoms:

- The health status of the backend origin servers goes down intermittently, as observed from the varnish logs.
- Origin server machines record high CPU usage.

- Processing times of many requests, as recorded in the origin access logs, are unusually high.

If machines running Primetime components are appropriately configured and are equipped with the requisite hardware, these symptoms indicate an over load condition. The following observations may help you determine whether you should deploy additional origin servers or reverse proxies:

- When most of the client traffic stops, fragment pushes from the packagers account for the entire load on the origin servers. If you observe overload symptoms even without client traffic, you should deploy additional origin servers to decrease the number of streams that is served from each origin.
- If GET requests account for most of the load on the origin servers, beef up the varnish / reverse-proxy layer. This could mean restructuring the varnish layer into two tiers, each tier comprising a set of servers that perform a specific function. The first tier interacts directly with the origins. The second tier, which comprises a set of reverse proxies, receives data from the first tier. This arrangement can effectively decrease the fan-out between the first reverse-proxy layer and the origins.
- • You can utilize auto scaling if you deploy the reverse proxies in the cloud to handle large variations in the client traffic.