

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: April 28, 2011

Q. Wu  
R. Huang  
Huawei  
October 25, 2010

Problem Statement for HTTP Streaming  
draft-wu-http-streaming-optimization-ps-03

## Abstract

HTTP Streaming allows breaking the live contents or stored contents into several chunks/fragments and supplying them in order to the client. However streaming long duration and high quality media over the internet to satisfy the real time streaming requirements has several Challenges when we require the client to access the same media content with the common Quality experience at any device, anytime, anywhere. This document explores problems inherent in HTTP streaming. Several issues regarding network support for HTTP Streaming have been raised, which include QoE improvement offering to streaming video over Internet, efficient delivery, Playback control and real time streaming media synchronization support.

## Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on April 28, 2011.

## Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

Internet-Draft

HTTP Streaming Problem Statement

October 2010

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Draft

HTTP Streaming Problem Statement

October 2010

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction . . . . .</a>	<a href="#">4</a>
<a href="#">2.</a>	<a href="#">Terminology . . . . .</a>	<a href="#">6</a>
<a href="#">3.</a>	<a href="#">Architecture of HTTP Streaming System . . . . .</a>	<a href="#">6</a>
<a href="#">3.1.</a>	<a href="#">Functions of Streaming Components . . . . .</a>	<a href="#">7</a>
<a href="#">3.2.</a>	<a href="#">Functions of Distribution Components . . . . .</a>	<a href="#">8</a>
<a href="#">3.3.</a>	<a href="#">Functions of Client Components . . . . .</a>	<a href="#">8</a>
<a href="#">4.</a>	<a href="#">Existing Work and Model . . . . .</a>	<a href="#">8</a>
<a href="#">4.1.</a>	<a href="#">Media Fragments URI . . . . .</a>	<a href="#">8</a>
<a href="#">4.2.</a>	<a href="#">Media Presentation Description . . . . .</a>	<a href="#">9</a>
<a href="#">4.3.</a>	<a href="#">Playback Control on media fragments . . . . .</a>	<a href="#">9</a>
<a href="#">4.4.</a>	<a href="#">Server Push . . . . .</a>	<a href="#">9</a>
<a href="#">4.5.</a>	<a href="#">Existing HTTP Streaming Model(Client Pull Model) . . . . .</a>	<a href="#">10</a>
<a href="#">5.</a>	<a href="#">Analysis of different use cases . . . . .</a>	<a href="#">10</a>
<a href="#">5.1.</a>	<a href="#">Live Streaming Media broadcast . . . . .</a>	<a href="#">10</a>
<a href="#">5.2.</a>	<a href="#">"Multi-Screen" Service Delivery . . . . .</a>	<a href="#">11</a>
<a href="#">5.3.</a>	<a href="#">Content Publishing between Servers . . . . .</a>	<a href="#">12</a>
<a href="#">6.</a>	<a href="#">Aspects of Problem . . . . .</a>	<a href="#">13</a>
<a href="#">6.1.</a>	<a href="#">Inefficient Streaming Content Delivery . . . . .</a>	<a href="#">13</a>
<a href="#">6.2.</a>	<a href="#">No QoE Guaranteed Support . . . . .</a>	<a href="#">14</a>
<a href="#">6.3.</a>	<a href="#">No QoS Control and Feedback Support . . . . .</a>	<a href="#">15</a>
6.4.	<a href="#">No Streaming Content Distribution and Discovery Support .</a>	<a href="#">16</a>
<a href="#">6.5.</a>	<a href="#">Lacking Streaming media Synchronization support . . . . .</a>	<a href="#">16</a>
<a href="#">6.6.</a>	<a href="#">No Multicast Support . . . . .</a>	<a href="#">17</a>
<a href="#">6.7.</a>	<a href="#">Inadequate Streaming Playback Control . . . . .</a>	<a href="#">17</a>
<a href="#">7.</a>	<a href="#">Scope of the problem . . . . .</a>	<a href="#">18</a>
<a href="#">7.1.</a>	<a href="#">Enhanced HTTP Streaming Pull model . . . . .</a>	<a href="#">19</a>
<a href="#">7.2.</a>	<a href="#">HTTP Streaming Push model . . . . .</a>	<a href="#">19</a>
<a href="#">8.</a>	<a href="#">Security Considerations . . . . .</a>	<a href="#">20</a>
<a href="#">9.</a>	<a href="#">Acknowledgement . . . . .</a>	<a href="#">21</a>
<a href="#">10.</a>	<a href="#">References . . . . .</a>	<a href="#">21</a>
<a href="#">10.1.</a>	<a href="#">Normative References . . . . .</a>	<a href="#">21</a>
<a href="#">10.2.</a>	<a href="#">Informative References . . . . .</a>	<a href="#">21</a>
	<a href="#">Authors' Addresses . . . . .</a>	<a href="#">22</a>

## 1. Introduction

Streaming service is described as transmission of data over network as a steady continuous stream, allowing playback to proceed while subsequent data is being received, which may utilize multiple transport protocols for data delivery. HTTP streaming refers to the streaming service wherein the HTTP protocol is used for basic transport of media data. One example of HTTP streaming is progressive download streaming which allows the user to access content using existing infrastructure before the data transfer is complete.

In recent years, HTTP streaming system has been widely used on the Internet for the delivery of multimedia content. There are several reasons for this industry trend, for example:

- o Existing Media Streaming protocols often have difficulty getting around firewalls because they are commonly based on UDP with unusual port numbers. HTTP streaming has no such problems because firewalls know to pass HTTP packets through well-known port 80.
- o Due to the success of Web, both HTTP server and HTTP client are quite common in industry, which means that building a HTTP based media delivery system may have less cost compared to those using dedicated media server/client and intermediaries.

In order to better support the streaming characteristics, such as trick modes, adaptation to resolutions, some approaches have been

commonly adopted in current HTTP streaming systems. For example, media segmentation method separates the whole media content to a series of small chunks and supplies them to the client through a sequence of HTTP responses. Segmentation allows the client to seek to different piece of media content, change the bit-rate of the next-to-fetch chunk, as well as enables reducing the overall transmission delay in case that one TCP connection trying to resend the lost packet before sending anything further.

With media segmentation support, existing HTTP streaming technology (e.g., progressive download HTTP streaming) is characterized as:

- o Client based pull schemes that is, the client firstly acquires a manifest file containing the reference (e.g. URI) to each media chunks from the streaming server, then requests the media chunks by forming a sequence of HTTP request messages to the server. This client based pull model more relies on client to handle buffer and playback during download.

- o Relying on existing web infrastructure, i.e., no special server and intermediaries are required other than a standard HTTP Server and HTTP caches/proxies.

However streaming long duration and high quality media over the internet to offer TV experience at any device and satisfy the real time streaming requirements faces several unique Challenges when there are no network capabilities available for HTTP Streaming:

- o In client pull model, there will be additional round trips between the client and the server for manifest file update before the client can request each new chunk, which could risk the real-time feature of live streaming.
- o Lack of QoE guarantee on the packet switching based Internet and hard to offer better experience than TV viewing. Compared to the dedicated IPTV system, the HTTP streaming based on the best-effort Internet may suffer more from network transition. For example, when a user switches live channel or start VoD, there is no guarantee on startup time.

- o Lack of Feedback on Quality of data delivery. e.g., there is no streaming quality control mechanisms like RTCP to report QoE metrics that are important to the HTTP streaming system for control or diagnostic purpose.
- o Lack of QoS Control. For example, there is no QoS difference between high speed Internet traffic and Streaming over Internet traffic and hard for QoS surcharges on consumer broadband subscription.
- o Lacking multicast support.

With these above challenges, the typical user experience with the existing HTTP streaming schemes may be limited by unacceptable latency and waiting time, better effort quality, buffering delays or interruption, etc, and inadequate playback control, especially for live broadcast. Therefore these existing streaming schemes is more applicable to recorded contents viewing that offers a better experience over slower connections for recorded contents viewer. Especially, in the case of "Multi-Screen", the service provider intends to provide a common user experience when the user enjoys the media content across PCs, TVs, and smart-phones. Therefore, HTTP streaming over the Internet without some optimization on network transport for QoE improvement may lead difficulty for the service provider to comply the service level agreements (SLAs) between

service provider and users.

This document explores problems inherent in HTTP streaming. Several issues regarding network support for HTTP Streaming have been raised, which include QoS improvement offering to streaming video over Internet, efficient delivery, playback control and real time streaming media synchronization support etc. The following sections described architecture of HTTP streaming system, introduce related works and model on HTTP streaming, analyze some use cases in HTTP streaming and list the potential problems when providing better streaming quality over the Internet.

## [2.](#) Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)].

Push Model: The model that allows the server keep pushing data packets to the client.

Pull Model: The model that allows the client keep pulling data packets from the server.

Live Streaming: Live events can be streamed over the Internet with the help of broadcast software which encodes the live source and delivers the resulting stream to the server. The server then transfers the stream. So the user experiences the event as it happens.

On-Demand Streaming: To provide "anytime" access to media content, client is allowed to select and playback on demand.

Progressive Download: A mode that allow client playback the media file while the file is downloading, after only a few seconds wait for buffering, the process of collecting the first part of a media file before playing.

Adaptive Streaming: Adaptive streaming is a process that adjusts the quality of a video delivered to a client based on the changing network conditions to ensure the best possible viewer experience.

### [3.](#) Architecture of HTTP Streaming System

Figure 1 shows reference Architecture for HTTP Streaming in general

case. The Architecture should comprise the following components:

+--Streaming--+	+--Distribution+	+-- Client --+
Component	Component	Component
+-----+	+-----+	+-----+
Media	HTTP	HTTP
Encoder	Proxy/	Client

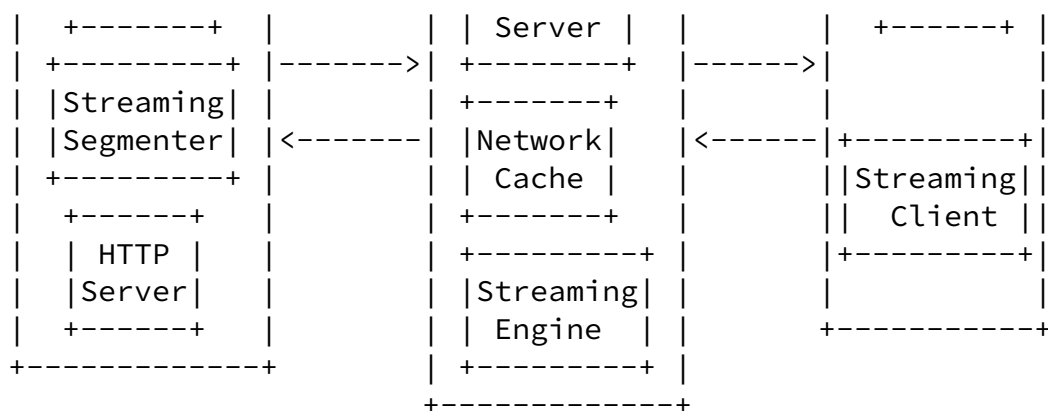


Figure 1: Reference Architecture for HTTP Streaming

- o Streaming Component
- o Distribution Component
- o Client Component

### 3.1. Functions of Streaming Components

Function	Role
Media Encoder	Encode the media and encapsulate with specific streaming formats for delivery
-	-
Streaming Segementer	divide the input media into a series of small chunks;Create index file containing reference to each chunks
-	-
HTTP Server	handles HTTP request from client and respond to HTTP connections

Figure 2: Functions of Streaming Component

### 3.2. Functions of Distribution Components



Function	Role
Network Cache	Cache the data
-	-
Streaming Engine	Distinguish between regular HTTP traffic and HTTP Streaming Traffic; Enable HTTP Streaming localized
-	-
HTTP Proxy/Server	Handles HTTP request from client;Forward data to client or respond to HTTP connections

Figure 3: Functions of Distribution Component

### 3.3. Functions of Client Components

Function	Role
Streaming Client	Handle Streaming Playout and Buffer
-	-
HTTP Client	Initialize HTTP Connection

Figure 4: Functions of Client Components

## 4. Existing Work and Model

Based on the architecture described in [Section 3](#), a growing number of works have been done to build HTTP streaming system with the intend to provide more streaming characteristics such as URI for media fragment, media presentation description, playback control etc. Also how existing HTTP Streaming model(i.e., client pull model) has been discussed.

### 4.1. Media Fragments URI

W3C Media Fragments Working Group extends URI defined in [[RFC3986](#)]and specifies some new semantics of URI fragments and URI queries [[MediaFragments](#)] which is used to identify media fragments. The

client can use such Media Fragments URI component to retrieve one fragment following the previous fragment from the server. However such component is not extensible to convey more important streaming information about bandwidth utilization, quality control and buffer management. Therefore it is a big challenge to use the existing web infrastructure with such component to deliver streaming contents with QoE guaranteed.

#### [4.2.](#) Media Presentation Description

[\[I.D-pantos-http-live-streaming\]](#) formerly defines media presentation format by extending M3U Playlist files and defining additional flags. 3GPP TS 26.234 also centers around media presentation format and specifies Semantics of Media presentation description for HTTP Adaptive Streaming [\[TS26.234\]](#), which contains metadata required by the client(i.e., Smartphone) to construct appropriate URIs [\[RFC3986\]](#) to access segments and to provide the streaming service to the user. We refer to this media presentation description as playlist component. With such component support, client can poll the new data in chunks one by one. However without client request using HTTP, the server will not push the new data to the client, therefore it may not meet the real-time requirements when streaming live media to clients only relying on client poll model to deliver high quality streaming contents across the best-effort Internet, especially when experiencing network transition. More survey on MPD and client pull model can be found in [\[GapAnalysis\]](#)

#### [4.3.](#) Playback Control on media fragments

W3C HTML5 Working Group has incorporated video playback features into HTML5 Specification which we refer to as local playback control. Such local playback capability has been previously dependent on third-party browser plug-ins. Now HTML5 specification lifts video playback out of the generic <object> element and put it into specialized <video> handlers. With such playback control support, implementers can choose to create their own controls with plain old HTML, CSS, and JavaScript. However this playback control can not be used to control streaming contents which are not downloaded to the browser client. Another example of playback control is trick mode support specified in 3GPP HTTP Adaptive Streaming [\[TS26.234\]](#), where the client can implement seeking by locating the corresponding chunk by using the information acquired in MPD. More survey on playback control can also be found in [\[GapAnalysis\]](#)

#### [4.4.](#) Server Push

there is no server-push protocol to be defined in IETF, which can be used to work with Server Sent Push API developed by W3C. IETF Hybi working group specifies websocket protocol, as one complementary work, W3C specifies websocket API. This websocket technology provides two-way communication with servers that does not rely on opening multiple HTTP connections. However it still lacks capability to push real time streaming data from the server-side to the client.

#### 4.5. Existing HTTP Streaming Model(Client Pull Model)

In the HTTP Streaming Pull model, the Distribution component does not take a stab into HTTP Streaming traffic. The Streaming Content flows directly from the server to the Client. Adaptive HTTP Streaming specified in 3GPP is one typical example of pull model. In the adaptive HTTP Streaming, the video streaming application is decoupled from existing web infrastructure. However, how the streaming contents is distributed from dedicated streaming server to HTTP Server is unspecified. This put Streaming live video feeds problematic. Internet streams created from live video feeds are usually fraught with glitches and interruptions. Once again, image quality, size and features are sacrificed because of limited encoding CPU resources and the absolute need to "keep up" with the live feed.

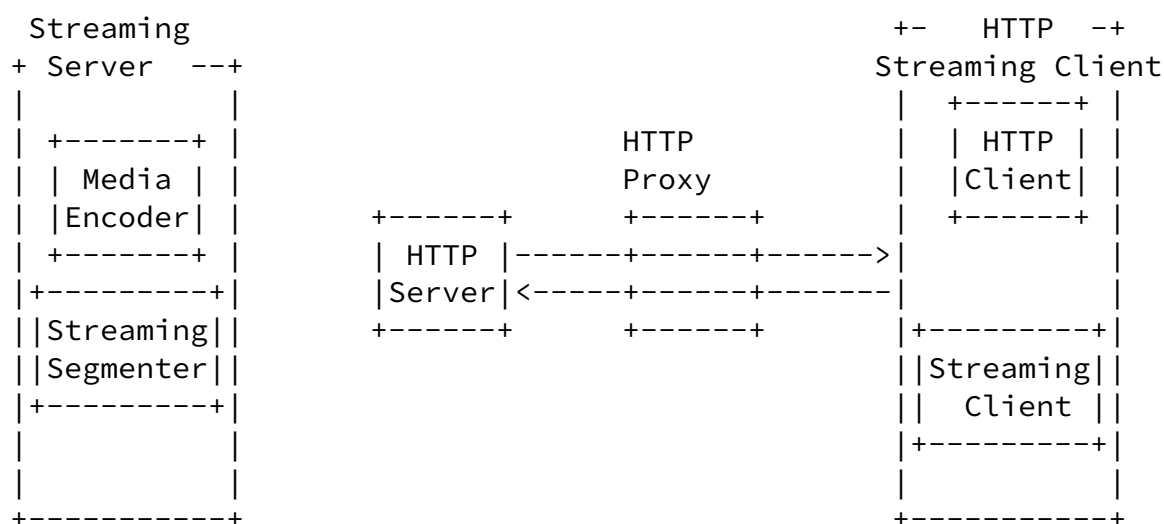


Figure 5: Pull model for HTTP Streaming

## [5.](#) Analysis of different use cases

### [5.1.](#) Live Streaming Media broadcast

Today, live video streaming technologies are widely used in broadcasting news, connecting friends and relatives in online chat rooms, conducting businesses online face to face, selling products

and services, teaching online courses, monitoring properties, showing movies online, and so on. Current HTTP streaming (both live streaming and VoD) is based on the pull model in which the client pulls a sequence of chunks one after another from the server, based on the manifest file produced by the server describing currently available chunks. The pull model gives a better control of media delivery, better handling of chunk scheduling, as well as buffer management on the client's side. In the case of live streaming, the server will need to update the manifest file frequently once a new chunk of live media becomes available. This may cause a major concern in time-sensitive scenario. There will be two additional round trips between the client and the server for manifest file update before the client can request each new chunk, which could risk the real-time feature of live streaming.

HTTP server push model, on the other hand, enables the server to actively and continuously push chunks to the client once a new chunk is available on the server, without the round trips between the client and the server for manifest file update. In this sense, push model could be more efficient and a better candidate for time-sensitive scenario.

### [5.2.](#) "Multi-Screen" Service Delivery

With the existing deployment today, the services like Network DVR and TV/Video anywhere are generally limited as to the types of device that they support, or the level of integration and interactivity between screens. "Multi-Screen" Service provides a common user experience across PCs, TVs, Smartphones, Tablets that enable consumers to access the same media content and quality of experience (QoE) on any device, anytime and anywhere. Such multi-Screen Experience is lacking for end user in the services like Network DVR

and TV/Video anywhere. Since all the clients have browser support, it is obviously one best choice to choose HTTP Streaming to deliver Multi-Screen Service. However utilizing HTTP Streaming to deliver Multi-Screen Service and meet the real time streaming requirements face several great challenges, which include:

Startup delay:

- \* Compared to the dedicated IPTV system, the HTTP streaming based on the best-effort Internet may suffer more from network transition. Although the client buffer can mitigate the overall effect of network transition, there is lack of guarantee on startup delay, which is an important QoE metric. For example, when a user switches live channel, the current group of pictures (GoP) and initialization information for decoders (a.k.a. Reference Information (RI)) of the media

content need to be acquired by the client ASAP to start playback. Rapid Acquisition of Multicast Session (RAMS) [[RAMS](#)] defined in IETF AVT WG aims to address the similar problem in the case of MPEG2-TS over RTP transmission by storing the important RTP packets for quick startup in intermediary node and feeding the packets to the client with high priority. Unfortunately, there is no mechanism so far to improve the transmission of the important HTTP packets, hence may introduce a long delay to start the playback in the scenario of HTTP streaming.

- \* With the intermediary nodes, it is possible that the important HTTP packets, e.g. those including current GoP and RI of the media content can be stored before the client switches live channel or start VoD. Then it could be possible to reduce the startup delay on the client by transmitting these packets to the client from the intermediaries, which are closer to the client than the normal HTTP server. Furthermore, the intermediaries could transmit these packets with higher priority than other HTTP packets, or faster than playback bit-rate, to achieve further QoE enhancement on the client side.

QoE feedback:

In IPTV system, RTCP is responsible for sending the feedback

about the media transmission quality to the media server. In the case of HTTP streaming, due to reliable data transmission provided by TCP, QoE metrics related to data transport such as packet loss are not relevant. However, some QoE metrics at session level, such as startup delay are still important to the HTTP streaming system for monitoring or diagnostic purpose. Unfortunately, there is no such quality monitoring mechanisms (e.g. like RTCP report) in current HTTP streaming system. To provide a high-quality service for the user, monitoring and analyzing the system's overall performance is extremely important, since offering the performance monitoring capability can help diagnose the potential network impairment. Furthermore QoE feedback mechanisms will facilitate in root cause analysis and verify compliance of service level agreements (SLAs) between service providers and users.

### [5.3.](#) Content Publishing between Servers

HTTP Streaming can be used in the CDN to optimize content delivery. Content Publisher may utilize HTTP Streaming to publish the popular contents on the Streaming sever to the Web Server or Proxy, which, in turn, reduce bandwidth requirements and server load, improve the

client response times for content stored in the cache. Also when the web cache fails to provide the contents that have greatest demand to the requester (e.g., Client), the web cache can use HTTP Streaming protocol to retrieve the contents from the web server and cache them on the web proxy waiting for the next request from the requester.

## [6.](#) Aspects of Problem

The Real time streaming service (e.g., RTSP) is superior in handling thousands of concurrent streams simultaneously, e.g., flexible responses to network congestion, efficient bandwidth utilization, and high quality performance. However streaming long duration and high quality media over the internet to offer TV experience at any device and satisfy the real time streaming requirements faces several unique Challenges.

### [6.1.](#) Inefficient Streaming Content Delivery

HTTP is not streaming protocol but can be used to distribute small chunked contents in order, e.g., transmit any media contents relying on time-based operation. However Client polling for each new data in chunks may not be efficient way to deliver high-quality streaming video content across the Internet for the following reasons:

- o Clients today send a significant amount of redundant data in the form of HTTP headers. Because a single web page may require 50 or 100 sub-requests, this data is significant. It is desirable to compress the headers which may save a significant amount of latency and bandwidth compared to HTTP.
- o Clients can not request certain resources to be delivered first. This may cause the problem of congesting the network channel with non-critical resources when a high-priority request is pending.
- o HTTP relies solely on multiple connections for concurrency. This causes several problems, including additional round trips for connection setup, slow-start delays, and a constant rationing by the client where it tries to avoid opening too many connections to a single server.
- o Since HTTP is operated over TCP, it is much more likely to cause major packet drop-outs and greater delay due to TCP with the characteristic which keeps TCP trying to resend the lost packet before sending anything further. Thus HTTP streaming protocols suffer from the inefficient communication established by TCP's design and they are not well suited for delivering nearly the same amount of streams as UDP transmission or RTSP transmission. When

network congestion happens, the transport may be degraded due to poor communication between client and server or slow response of the server for the transmission rate changes.

- o Client polling may keep waiting for the arrival of data in response to the previous request before sending the next new request.
- o In the case of live streaming, the server will need to update the manifest file frequently once a new chunk of live media becomes available. This may cause a major concern in time-sensitive

scenario. There will be additional round trips between the client and the server for manifest file update before the client can request each new chunk, which could risk the real-time feature of live streaming.

Therefore it is desirable to offer better transport for streaming contents delivery.

## 6.2. No QoE Guaranteed Support

Due to the lack of QoE guarantee on the packet switching based Internet, the internet streaming can only provide best effort quality to consumers.

This may have the following effects that are not desirable:

- o The quality of Internet media streaming may significantly degrade due to rising usage and concurrent streaming delivery. The Internet traffic generated by HTTP streaming may exhibit burstiness extremely or other dynamics changes.
- o The filling of the client play-out buffer, which is used to smooth jitter caused by network bandwidth fluctuation, may further increase user's waiting time.
- o Streaming service tends to over-utilize the CPU and bandwidth resource to provide better services to end users, which may be not desirable and effective way to improve the quality of streaming media delivery, in worse case, the server may not have enough bandwidth to support all of the client connections. When CPU resources are exhausted or insufficient, the server must sacrifice/downgrade quality to enable the process to keep pace with live contents rendering for viewing. Therefore the content owner is forced to limit quality or viewing experience in order to support live streams.

- o when MBR(i.e., Multiple Bit Rate) encoding is supported, the encoder usually generates multiple streams with different bit rates for the same media content, and encapsulates all these streams together, which needs additional processing capability and



a possibly large storage and in worse case, may cause streaming session to suffer various quality downgrading, e.g., switching from high bit rate stream to low bit rate stream, rebuffering when the functionality of MBR is poorly utilized.

A study conducted by StreamGuard.net examined more than 2,400 different Internet video streams. They concluded that:

- o 51% of stream viewers were frustrated with the experience.
- o 17% of streams have an immediate fatal error.
- o 25% take between 5 and 10 seconds to start playing, and stop to rebuffer before the video is finished.

With current technologies, dial-up users do not typically attempt video streaming and broadband users often give up after prolonged waits or unresponsiveness due to regular rebuffering.

### 6.3. No QoS Control and Feedback Support

The usage of streaming media is rapidly increasing on the web. To provide a high-quality service for the user, QoS control such as monitoring and analyzing the system's overall performance is extremely important, since offering the performance monitoring capability can help diagnose the potential network impairment, facilitate in root cause analysis and verify compliance of service level agreements (SLAs) between Internet Service Providers (ISPs) and content provider.

In the current HTTP streaming technology, it fails to give the server feedback about the experience the user actually had while watching a particular video. This is because the server controls all processes and it is impossible to track everything from the server side.

Consequently, the server may be paying to stream content that is rarely or never watched. Alternatively, the server may have a video that continually fails to start or content that rebuffers continually. But the Content owner or encoder receives none of this information because there is no way to track it.

Therefore it is desirable to allow the server view detailed

statistics using the system's extensive network, quality, and usage monitoring capabilities. This detailed statistics can be in the form of real-time quality of service metrics data.

#### [6.4.](#) No Streaming Content Distribution and Discovery Support

Unlike standard web pages and graphics and P2P Streaming data, streaming video files may not be cacheable by web cache servers on the network or by the browser on your local hard drive. Therefore how to distribute the streaming video files to the distributed component in the Content Delivery Network and how the distribution server serves the Client with these streaming video files are still problematic issues.

#### [6.5.](#) Lacking Streaming media Synchronization support

In the push model, the client just passively accepts what the server pushes out and always knows how the live stream is progressing. However if the client's clock is running slower than the encoder's clock, buffer overflow will happen, i.e., the client is not consuming samples as fast as the encoder is producing them. As samples get pushed to the client, more and more get buffered, and the buffer size keeps growing over time. This can cause the client to slow down packet processing and eventually run out of memory. On the other hand, if a client's clock is running faster than the encoder's clock, the client has to either keep re-buffering or tune down its clock. To detect this case, the client needs to distinguish this condition from others that could also cause buffer underflow, e.g. network congestion. This determination is often difficult to implement in a valid and authoritative manner. The client would need to run statistics over an extended period of time to detect a pattern that's most likely caused by clock drift rather than something else. Even with that, false detection can still happen.

In the pull model, the client is the one who initiates all the fragment requests and it needs to know the right timing information for each fragment in order to do the right scheduling [Smooth Streaming]. Given that the server is stateless in the pull model and the client could communicate with any server for the same streaming session, it has become more challenging. The solution is to always rely on the encoder's clock for computing timing information for each fragment and design a timing mechanism that's stateless and cacheable.

With the pull model for HTTP Streaming, The client is driving all the requests and it will only request the fragments that it needs and can handle. In other words, the client's buffer is always synchronized

to the client's clock and never gets out of control. Currently most

of existing streaming schemes are based on pull model. However the side effect of this type of clock drift would be that the client could slowly fall behind, especially when transitioning from a "live" client to a DVR client (playing something stored in the past).

#### [6.6.](#) No Multicast Support

HTTP is sent over TCP and only supports unicast which may increase processing overhead by 30% in contrast with using multicast transmission.

#### [6.7.](#) Inadequate Streaming Playback Control

Playback control allows user interact with streaming contents to control presentation operation (e.g., fast forward, rewind, scrub, time-shift, or play in slow motion). RTSP streaming provides such capability to control and navigate the streaming session when the client receives the streaming contents. Unlike RTSP streaming, current HTTP streaming technologies do not provide such capability for playback control that users are accustomed to with DVD or television viewing, which significantly impacts the viewing experience.

This also has the following effects that are not desirable:

- o When the user requests media fragments that correspond to the content's new time index and the media fragments from that point forward, the client can not have the possibility to change the time position for playback and select another stream for rendering with acceptable quality.
- o The user can not seek through media content whilst viewing the content with acceptable quality.
- o When the user requests to watch the relevant fragments rather than having to watch the full videos and manually scroll for the relevant fragments, the client can not have the possibility of jumping to another point within the media clip or between the media fragments with acceptable quality (i.e., random access).

- o When the media content the user requests to watch is live stream and needs to be interrupted in the middle, e.g., when the user takes a phone call, the client can not have the possibility to pause or resume the streaming session with acceptable quality after it has been invoked.

- o When the user begins to see the content at the new time point, if the media fragments retrieved when changing position require the same quality as the media fragments currently being played, it will result in poor user experience with longer startups latency.
- o When there are different formats corresponding to the terminal capabilities and user preferences available for contents, the client has no capability to select one format for which the content will be streamed.
- o When the user doesn't have time to watch all the streaming contents and want to skip trivial part and jump to the key part, the client does not provide the capability for selective preview or navigation control.
- o When the server wants to replace the currently transmitted video stream with a lower bit-rate version of the same video stream, the server has no capability to notify this to the client.

## 7. Scope of the problem

Goal: Develop interoperable solutions that offer more efficient transport for real time streaming media and allows interoperability with other existing streaming techniques.

Possible Directions forward:

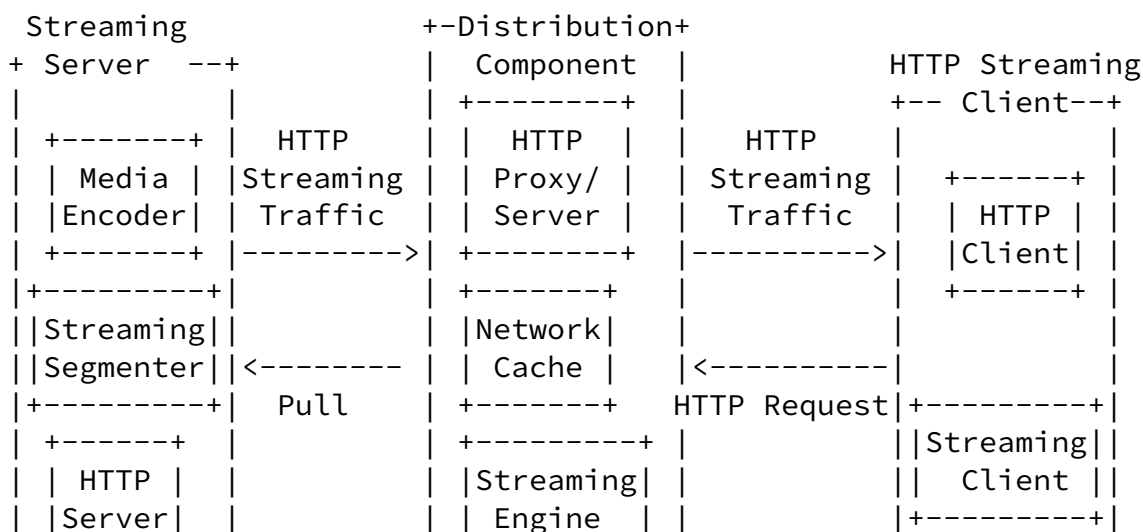
- o Specify how an application running over HTTP should operate in order to be able to support HTTP streaming and avoid certain challenges/issues

- o Extend websocket protocol to support HTTP Streaming and avoid certain challenges/issues
- o Build a different protocol to HTTP like RTMP for streaming data
- o Specify a different transport layer for HTTP to avoid the problems
- o Other suggestions from working group via mailing list or ad-hoc meeting

Possible Models to be built:

### [7.1.](#) Enhanced HTTP Streaming Pull model

Unlike the basic pull model defined in [section 5.1](#), the interface between the Streaming Server and the Distribution Server is defined. HTTP Streaming schemes is used for data transport between two servers. Also the Distribution Component is introduced with Streaming support to provide a "hint" to the server/cache as to which chunk the client is likely to request next then the distribution component could elect to retrieve that chunk ahead of it actually being requested to keep the response latency (or some other factor) more consistent and avoid additional bit rate switches.



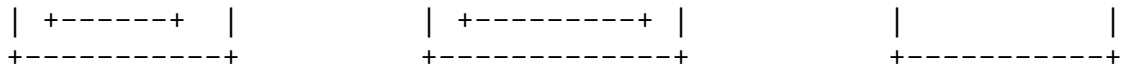


Figure 6: Enhanced Pull model with involvement of Distribution Component

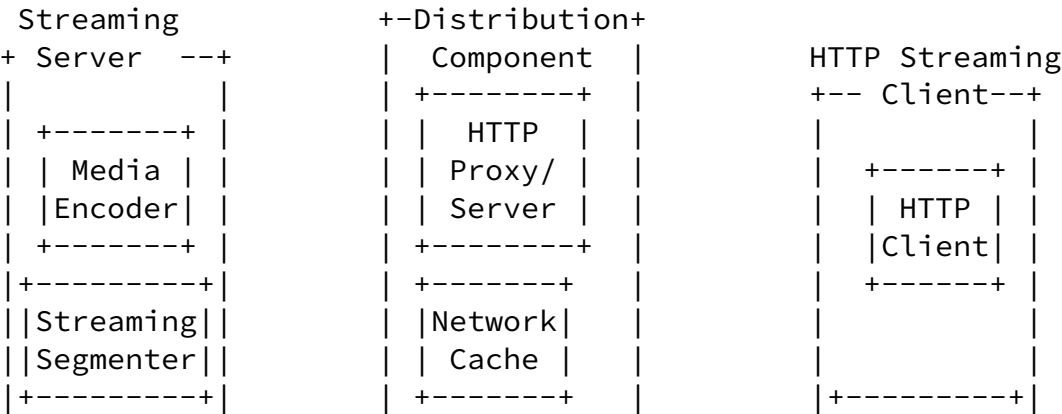
### 7.2. HTTP Streaming Push model

In the push model, there may have one or more than one Distribution Servers placed between HTTP Streaming Server and HTTP Streaming Client.

When the Distribution Server does not get involved in HTTP Streaming, Streaming Content flows directly from the server to the Client. The server keeps pushing the latest data packets to the client and the client just passively receives everything. The distribution server is just responsible for forwarding stream as all the other HTTP proxies do.

When the Distribution Server gets involved in HTTP Streaming, The HTTP Streaming Server keeps pushing the latest data packets to the client, in the meanwhile, the HTTP Streaming server may also push the

data packets to the distribution server for caching when there is enough interest from clients on the same streams. When the new client requests the same data packets as the previous client and the data packets requested is cached on the distribution server, the distribution server can terminate this request on behalf of the HTTP Streaming server and push the requested data cached on itself to this new client.



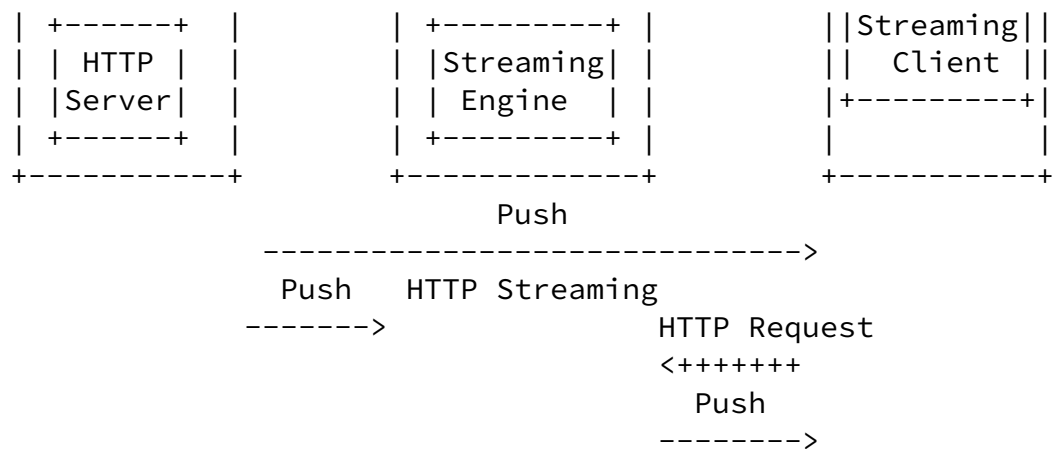


Figure 7: Push model for HTTP Streaming

## 8. Security Considerations

In order to protect the content against theft or unauthorized use, the possible desirable features include:

- o Authorize users to view a stream once or an unlimited number of times.
- o Permit unlimited viewings but restrict viewing to a particular machine, a region of the world, or within a limit period of time.

- o Permit viewing but not copying or allow only one copy with a timestamp that prevents viewing after a certain time.
- o Charge per view or per unit of time, per episode, or view.

## 9. Acknowledgement

The authors would like to thank David A. Bryan, Ning Zong, Bill Ver Steeg, Ali Begen, Colin Perkins, Roni Even, Daniel Park, Henry Sinnreich, Wenbo Zhu, Lars Eggert, Spencer Dawkins, Ben Niven-

Jenkins, Marshall Eubanks, Kathy McEwen for their suggestions and inputs on this document. Also Thanks Thomas Stockhammer, Luby, Michael, Mark Watson for their precious comments.

## 10. References

### 10.1. Normative References

- [RFC2326] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", [RFC 2326](#), April 1998.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC1945] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", [RFC 1945](#), May 1996.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, [RFC 3986](#), January 2005.
- [TS26.234]  
3GPP Standard, "Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs (Release 9)", 3GPP TS 26.234 , March 2010.

### 10.2. Informative References

- [RAMS] VerSteeg , B., Begen , A., VanCaenegem, T., and Z. Vax, "Unicast-Based Rapid Acquisition of Multicast RTP Sessions", [draft-ietf-avt-rapid-acquisition-for-rtp-16](#) (work in progress), October 2010.
- [I.D-pantos-http-live-streaming]  
Pantos, R. and W. May, "HTTP Live Streaming", June 2010.

#### [GapAnalysis]

Zong, N., "Survey and Gap Analysis for HTTP Streaming Systems", October 2010.

- [J.1080] ITU-T, "Quality of experience requirements for IPTV



services", Recommendation ITU T G.1080 .

[I-D.ietf-pmol-metrics-framework-02]

Clark, A., "Framework for Performance Metric Development".

[HTML5] W3C, "HTML5",

<http://www.w3.org/TR/html5/video.html#media-elements> .

[ServerSentEvent]

W3C, "Server Sent Event",

<http://www.w3.org/TR/eventsource/> .

[MediaFragments]

W3C, "Media Fragments", <http://www.w3.org/2008/WebVideo/Fragments/WD-media-fragments-spec/> .

[SmoothStreaming]

Microsoft, "Smooth Streaming", <http://blogs.iis.net/samzhang/archive/2009/03/27/>

live-smooth-streaming-design-thoughts.aspx .

#### Authors' Addresses

Qin Wu  
Huawei  
101 Software Avenue, Yuhua District  
Nanjing, Jiangsu 210012  
China

Email: [sunseawq@huawei.com](mailto:sunseawq@huawei.com)

Rachel Huang  
Huawei  
101 Software Avenue, Yuhua District  
Nanjing, Jiangsu 210012  
China

Email: [Rachel@huawei.com](mailto:Rachel@huawei.com)