ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (404-409), Month: April-June 2014, Available at: www.researchpublish.com

Optimized Video Transmission on Network

Munish

Kurukshetra University, Kurukshetra, India

Abstract: **As video becomes one of the most demanding services for network traffic and Internet video becomes an important part of modern life, the quality of experience needs to meet the user's expectations, regardless of the device or network being used. More and more video is being consumed on Smartphone's and tablets. The smaller screens allow for lower bit-rates, but the video playback need to start quickly and remain smooth throughout. Video Transmission on Network is challenging because of factors such as high bitrates and sensitivity to delay or packet loss. The past, video streaming was typically associated to RTSP, with RTP used for transmission. This protocol uses "VCR-like" commands such as PLAY and PAUSE. In this scheme, the server has to keep track of the client's state. Today, HTTP based adaptive streaming services provide a more efficient means of delivery over the Internet. Similarly to RTSP, the video stream can start playing out before the entire file has been downloaded. The main advantage of adaptive streaming however that is it allows the end point to adjust to changes in available bandwidth and delay. Our early work focused on demonstrating the viability of server-side pacing schemes to produce an HTTP-based streaming server. We investigated the ability of client-side pacing schemes to work with both commodity HTTP servers & our HTTP streaming server.**

I. INTRODUCTION

Over the last decade, the role of on-demand video delivery has grown significantly. In the past days of video transmission and listening to music online wasn't always fun. It was a little like driving in stop-and-go traffic during a heavy rain. If you had a slow computer or a dial-up Internet connection, you could spend more time staring at the word "buffering" on a status bar than watching videos or listening to songs. On top of that, everything was choppy, pixilated and hard to see. Initial forays into Internet-based video delivery were limited to low resolution clips, much of which was user generated content (UGC), through services like You Tube. This eventually evolved into standard definition professional content, distributed through content aggregators like Hulu. Today, premium content providers are going direct to consumers with full-length high-definition content, delivered in an over-the-top (OTT) manner. The dramatic shift in consumer viewing habits, from a rigid in-home linear television model to a more flexible mobile on-demand model, has caught the attention of traditional television and Web-based video distribution services. Two of the most significant contributing factors have been the popularity of the Apple iPhone and iPad and the popularity of the Netflix streaming service. Though Apple was not the first to manufacture a smart phone or tablet, nor were they the first to define a segment-based HTTP delivery protocol, i.e., the HTTP Live Streaming (HLS) protocol, they were the first to integrate a segment-based HTTP delivery protocol into a mobile device that became instantly popular with tens of millions of consumers. In the same way, Netflix was not the first to deliver content to televisions using an OTT distribution model, but they were the first to do so with great commercial success. The common availability of video to mobile users provided by smart phones and tablets, combined with the on-demand convenience of OTT streaming services has created a higher level of expectation among viewers seeking both quality and flexibility in their content delivery. To fully comprehend this shift in video delivery paradigms, we must first investigate the video streaming protocols which came before, but with the appreciation that the assumptions which motivated both their design and optimization may require re-evaluation. In the following sections we follow the migration of streaming technologies as they have kept pace with advances in the server, networking, and storage technologies on which they rely. We can then see the incremental addition of features that eventually led to a reassessment and simplification of the delivery paradigm. From there, we can begin to address new enhancements to OTT delivery methods, as well as consider the modification and reapplication of previous streaming protocol optimizations.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (404-409), Month: April-June 2014, Available at: www.researchpublish.com

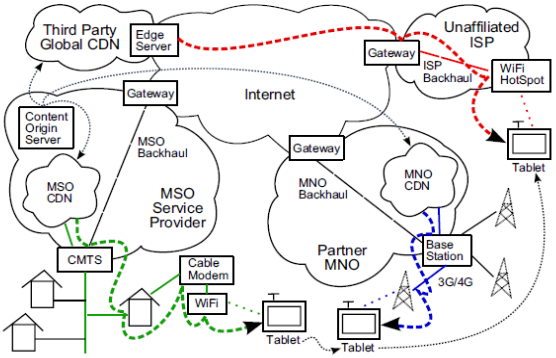
II. VIDEO DELIVERY PARADIGMS

Beyond the many orders of magnitude increases in bandwidth available to consumer devices, what may be considered the primary factor in the migration to OTT delivery is the acceptance that, in many cases, near-real-time delivery (or time-shifted delivery) of content is as good as, if not better than, real-time delivery of the same content.

Clients typically issue partial content requests using either HTTP Range Gets, or by retrieving pre-processed file segments. to improve the quality and predictability of RTP delivered video. Standard techniques like caching and packet prioritization have been adapted to support higher quality delivered video in challenged (loss) networks.

Transmission Rate Adaptation

Hyper text Transfer Protocol: adaptive streaming protocols use segment techniques, not only for paced continues download, but also for initiating bitrates adaptation. The Master manifest information file contains a list of individual bitrates manifest files locations, providing encoding information for each representation. The individual bitrates manifest files contain lists of segment file locations. The segments for each bitrates are synchronized so that segment boundaries are aligned, making switching between iterates seamless when performed on segment boundaries and delivery to the destination.



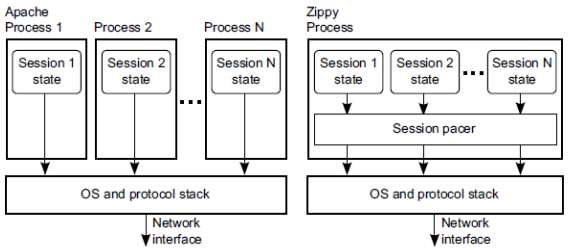
OTT Video Delivery

Two adaptive streaming protocols HLS & DASH were initially developed in HTTP to enable mobile clients to adjust to the dynamic nature of Mobile networks, however, OTT video delivery has started to emerge in more traditional video transmission & distribution outlets. HTTP adaptive streaming schemes have mainly focused on maximizing QoE for individual clients, rather than considering overall network throughput, but as the ability to deliver content to multiple screens becomes increasingly important, the data content providers and network providers are becoming more concerned with the ability to provide fair access to content and enforcing minimum QoE levels for all clients.

Outline

The research evaluated in this topic addresses the need for optimized Video Transmission on Network through the use bandwidth management for HTTP streaming. As MSOs and MNOs evaluate HTTP adaptive streaming technologies performs his duty to address the on-demand expectations and content delivery of their increasingly mobile customers, it is unusually obvious that the current state of technology lacks the level of control to which they have become familiarized. This research seeks to answer of the question of how we can provide monitoring and management of High Video Quality on a limited bandwidth network medium using the capabilities of HTTP Protocols. Our initial work, focused on server-side pacing in single bitrates progressive download applications, and to improve server and data-centre uplink scalability. Our main concerned was on client-side architectures for implementing data proxies, to enable data rate adaptation in clients. As HTTP adaptive bitrates client support became more common, our research refocused onto methods for providing general network operators the ability to perform traffic management at the HTTP segment request level.

III. HTTP PROGRESSIVE DOWNLOAD SERVER

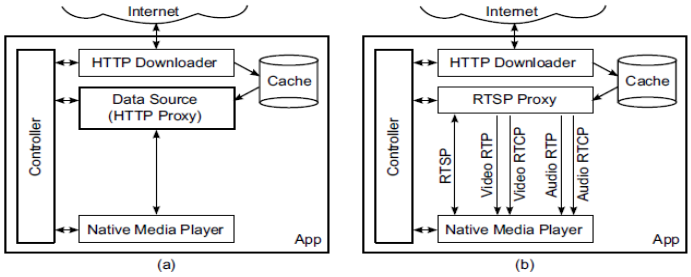


We have developed an HTTP streaming network server architecture, named Zippy, which uses paced delivery to distribute bandwidth usage over time, and to increase the scalability of the server, compared to a standard HTTP Web server. The Zippy architecture relies on a single pacer thread for managing individual session data transmissions, rather than using an individual process per connection, as

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (404-409), Month: April-June 2014, Available at: www.researchpublish.com

with a default Apache perform mode HTTP server shows the difference between the Apache multi-process architecture and the Zippy single thread architecture. With Zippy, connection fairness between sessions is explicitly enforced by the session pacer, rather than the OS scheduler. Sessions are never pre-empted by the session pacer; sessions perform a limited amount of processing and then yield to the session.

HTTP Adaptive Bitrates Client



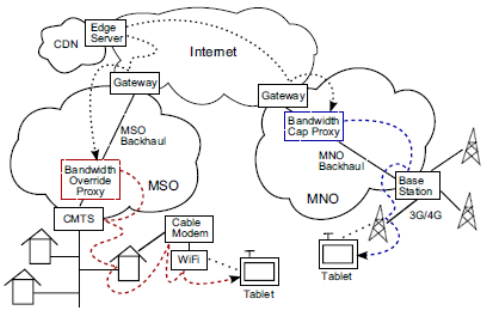
Though many modern devices natively support HTTP adaptive streaming protocols (e.g., HLS or Smooth Streaming), this is a relatively recent development. There are a large number of legacy devices and platforms which do not fully support HTTP adaptive streaming natively (including all mobile devices and Mobile OS like Android, Blackberry etc). Many of those devices do support other network-based video streaming options. For those devices, we developed client-side proxy architecture for implementing an HTTP adaptive streaming like approach. Protocol support for the non-HTTP adaptive streaming platforms fall into four general categories:

- HTTP progressive download,
- Local file data source,
- Frame-based data source,
- RTSP streaming.

Architectures rely on an HTTP downloader module which retrieves segments and stores them in a local cache. They also both have a controller which coordinates the initiation of HTTP downloads and the starting of the native media player. The controller is also responsible for dynamically selecting the bitrates and instructing the HTTP downloader to retrieve data for the selected bitrates.

IV. HTTP ADAPTIVE STREAMING BITRATES SELECTION OVERRIDE PROXY

In working with the HTTP adaptive streaming clients, it became evident that the greedy nature of the individual clients was not entirely conducive to fair bandwidth allocation between multiple clients. Though greedy clients may eventually reach a steady state bandwidth distribution, competition for bandwidth can cause bitrates thrashing and throughput squelching which affect QoE in individual clients. We began to look at methods for implementing session tracking and fair bandwidth distribution between sessions. In a degenerate case, this could be implemented as part of a server, similar to the methods described in Chapter 2, but most large scale deployments have many servers in a server farm, which require coordination, and individual clients may experience different network conditions downstream from the servers. For this reason, we adopted a network proxy-based approach to managing HTTP adaptive streaming session. In this chapter, we discuss two paradigms for performing bandwidth management using rate selection overrides on a per-segment basis.

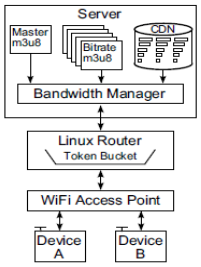


In the full override mode, the network proxy takes full control over rate adaptation. Available bitrates information is hidden from the client by the network proxy. The client has no expectation of rate adaptation, and the network proxy performs rate selection and replacement in a transparent fashion.

To better illustrate the basic functionality of our network controlled segment selection scheme, we use a scaled down configuration with two clients and limit the available bandwidth such that insufficient bandwidth exists for both clients to stream at the highest bitrates simultaneously. Though

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (404-409), Month: April-June 2014, Available at: www.researchpublish.com

the scheme scales well beyond two devices, limiting the number of clients allows us to more easily isolate the actions of each individual client, and allows us to use actual iOS devices, where large scale tests with actual iOS devices would be cost prohibitive.



In the bandwidth cap mode, the network proxy measures total band width usage and only overrides client bitrate requests when they would cause the overall bandwidth usage to exceed the cap. Available bitrate information is processed by the proxy so that it is aware of the available bitrates, but the information is passed through to the client unmodified. The client is unaware of the network proxy's existence and performs rate adaptation as it normally would. The network proxy performs bitrates capping and segment replacement in a transparent fashion.

In the bandwidth capping scenario, as with the full override case, we use a scaled down configuration to simplify the discussion. In this case we use four clients and assume that the downstream bandwidth is more than sufficient to support all clients streaming the highest bitrates. A bandwidth cap is assumed to apply to the upstream link.

V. HTTP ADAPTIVE STREAMING RATE ADAPTATION ALGORITHM

We discuss a rate adaptation algorithm, which, in its base form, provides a typical greedy approach to rate selection in HTTP adaptive streaming clients. We also detail configurable parameters which enable the rate adaptation algorithm to support CoS differentiation. The algorithm is suitable for use with any client-side architecture, including the one described. It also works with the transparent bandwidth capping capabilities described in Chapter 4. Though we focus our discussion on the CoS differentiation properties of the rate adaptation algorithm, our baseline comparison is with the degenerate single CoS case of our rate adaptation implementation.

Rate Adaptation Call-back Procedures

Procedure 5.1 *SegmentDownloadStart()*

inputs
$c \in \{0 \ldots C-1\}$
do
$abortTimer \leftarrow \alpha_c$
if $\gamma_c = 0$ **then**
$\gamma_c \leftarrow RAND\left(\frac{L}{W}\right)$
end if

Procedure 5.2 *SegmentDownloadComplete()*

inputs
$c \in \{0 \ldots C-1\}$
do
$q_c \leftarrow q_c + 1$
$abortTimer \leftarrow \emptyset$
if $q_c = Q$ **then**
if $\frac{b_{i+1}^c}{b_i^c} \cdot \delta_c < \alpha_c$ **then**
$b_i^c \leftarrow b_{i+1}$
end if
else
$backoffTimer \leftarrow \beta_c^+$
end if

Procedure 5.3 *AbortTimeoutExpired()*

inputs
$c \in \{0 \ldots C-1\}$
do
$AbortCurrentDownload(c)$
$\gamma_c \leftarrow 0$
if $b_{i-1}^c > 0$ **then**
$b_i^c \leftarrow b_{i-1}^c$
end if
$backoffTimer \leftarrow \beta_c$

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 2, Issue 2, pp: (404-409), Month: April-June 2014, Available at: www.researchpublish.com

Procedure 5.4 $SegmentPlayoutComplete()$

```
inputs
  c ∈ {0 . . . C − 1}
do
  q_c ← q_c − 1
  if q_c = 0 then
    if abortTimer ≠ ∅ then
      AbortTimeoutExpired(c)
    end if
  else if q_c = Q − 1 then
    StartNextDownload(c)
  end if
```

Procedure 5.5 $BackoffTimerExpired()$

```
inputs
  c ∈ {0 . . . C − 1}
do
  backoffTimer ← ∅
  StartNextDownload(c)
```

Procedure 5.6 Network Simulation Model

```
do
  unused ← 0
  goodput ← 0
  badput ← 0
  t ← 0
  while t < T do
    for all c ∈ 0 . . . C − 1 do
      if q_c ≠ 0 then
        d_c ← d_c + 1
        if d_c mod L = 0 then
          SegmentPlayoutComplete(c)
        end if
      end if
      if backoffTimer ≠ ∅ then
        backoffTimer ← backoffTimer − 1
        if backoffTimer = 0 then
          BackoffTimerExpired(c)
        end if
      end if
    end for
    n ← 0
    while n < N do
      if ∀c ∈ 0 . . . C − 1 : q_c = Q ∨ backoffTimer ≠ ∅ then
        unused ← unused + (N − n)
        n ← N
      else
        c ← RAND(C′)
        f_c ← f_c + P
        if f_c ≥ b_i^c · L then
          goodput ← goodput + f_c
          f_c ← 0
          SegmentDownloadComplete(c)
        else if δ_c > α_c then
          badput ← badput + f_c
          f_c ← 0
          AbortTimeoutExpired(c)
        end if
        n ← n + P
      end if
    end while
    t ← t + 1
  end while
```

VI. CONCLUSION

From last few years, researchers have been investigating methods for enhancing the capabilities of RTP streaming protocol to improve the video delivery reliability with the ultimate goal of improving the quality of the rendered content. The RTP protocol was defined with the specific intent of delivery real-time content. To minimize latency, content generated in real-time must be delivered just-in-time. Any content that is lost in transmission must be ignored as the stream must go on. With these constraints in mind, RTP was designed to use frame-based packetization and unreliable UDP transport to enable "graceful degradation", i.e., the ability to ignore lost frames. While graceful degradation is necessary for real-time content and offers advantages in extremely challenging networking environments, under normal circumstances RTP-based delivery simply creates the possibility for unnecessary degradation. Researchers have long regarded RTP as a one size fits all video delivery protocol, applying it equally to live and interactive real-time video, as well as VoD and time-shifted broadcast video. Though RTP is a capable video delivery protocol, it is not optimal in all cases. MSOs and MNOs are accustomed to having complete control over an entire network of managed devices. Each linear television channel is provisioned for bandwidth and multicast distribution over a QAM, while individual STBs and modems are registered and provisioned or on-demand bandwidth requirements. Moving to an OTT model with unmanaged clients poses a significant issue for operators who wish to deliver their own "on-deck" (i.e., operator managed) services. For "off-deck" services (i.e. services not administered by the network operator), the operator simply sees a generic data connection. Traditional traffic management techniques can be used to throttle individual clients. because the service is off-deck, there is no concern about any impact to quality which may occur from rate limiting. With on-deck services, however, operators want to ensure high QoE so that customers will continue to pay for the service. In these on-deck cases, the ability to manage OTT delivery in a way that ensures high QoE is highly desirable. Our research has followed the evolution of HTTP-based content delivery, from our early work with HTTP streaming servers, through the development of our client rate adaptation architecture, to our network proxy-based traffic management schemes, and culminating with our distributed CoS enforcement scheme. Though we have surveyed a large portion of the prior work in the RTP space, we have approached HTTP-based content delivery with a clean slate. Taking the primary advantage of streaming delivery, i.e., pacing, and removing the primary disadvantage of streaming delivery, i.e., silent data loss, we embarked on a journey to understand the barriers to an efficient (i.e., paced), high quality (i.e., lossless), and manageable HTTP-based video delivery protocol.

REFERENCES

[1] B. Wang, W. Wei, Z. Guo, and D. Towsley, "Multipath live streaming via TCP: Scheme, performance and benefits," ACM Transactions on Multimedia Computing Communications and Applications, vol. 5, no. 3, August 2009.

[2] L. Shen, W. Tu, and E. Steinbach, "A flexible starting point based partial caching algorithm for video on demand," in Proceedings of the 2007 IEEE Internatio

[3] R. Kumar, "A protocol with transcoding to support QoS over Internet for multimedia traffic," in Proc. of the 2003 IEEE International Conference on Multimedia & Expo (ICME 2003), July 2003, pp. 465–468.

[4] D. Mauro, D. Schonfeld, and C. Casetti, "A peer-to-peer overlay network for realtime video communication using multiple paths," in Proceedings of the 2006 IEEE International Conference on Multimedia & Expo (ICME 2006), July 2006, pp. 921–924.

[5] J. G. Apostolopoulos, W. Tan, and S. J.Wee, "On multiple description streaming with content delivery networks," in Proceedings of the 2002 IEEE International Conference on Computer Communications (InfoCom 2002), June 2002, pp. 1736–1745.

[6] S. Chattopadhyay, L. Ramaswamy, and S. M. Bhandarkar, "A framework for encoding and caching of video for quality adaptive progressive download," in Proceedings of the 2007 ACM International Conference on Multimedia, September 2007, pp. 775–778.