Abstract - Progressive download (PD) is a video streaming method over HTTP. Although PD is the most common streaming method over the internet it is highly inefficient from the internet service provider (ISP) point of view. ISPs need to compete with increasing competition, declining profitability and increasing client demand for network bandwidth (BW). ISPs, therefore, depend on the ability to optimize their network traffic, where video streaming has become the number one task.

As ISPs depend on deep packet inspection (DPI) systems in order to optimized and control their network, client/server shaping solution cannot be leaned on. Furthermore, such solutions are traditionally created by different buffer-based systems such as Leaky Bucket, but it is problematic to implement them on buffer-limited systems. Therefore, a highly efficient video traffic solution is needed.

This paper presents a buffer-free video streaming traffic shaping solution, based on TCP window size and scale modification which depends on the CBR video encoding rate and network conditions. Our solution can save up to 60% percent of bandwidth per connection under certain viewing habits conditions. Our simulation, which consisted of 3600 users over the time span of one hour, managed to achieve better network utilization by up to 25%.

Keywords: Traffic Shaping, Video Mobile Network optimization, Flash Video, Progressive Download over HTTP, Deep Packet Inspection

1 Introduction

Video streaming is constantly gaining popularity, with hundreds of millions of Internet users viewing video online. Video streaming is now responsible for the majority of Internet traffic, and is expected to keep growing over the next years, growth that is expected to be even more substantial in mobile networks. Service providers now face many new challenges when managing their networks bandwidth (BW).

The MEDIEVAL [18] project is addressing the huge demand for video traffic by specifying an enhanced architecture to advanced mobile networks to handle video services and optimizing the Quality of Experience (QoE) rather than the Quality of Service (QoS). As part of the optimization proposed in MEDIEVAL a packet dropping approach for UDP traffic based on content priority was proposed in [20] for Scalable Video Coding (SVC) [19] however the mechanism for optimization of TCP based traffic has to adopt a different approach as proposed in here.

While traffic consumption was once dominated by peer-to-peer (P2P) networks, P2P traffic is now declining rapidly due to the increasing popularity of streaming services like YouTube, Netflix and Hulu. Streaming services has gained a market share of over 35% from the total network BW [1], while YouTube, for example, has over 2 billion video views per week and is responsible for more than 17% of mobile network traffic.

The most popular technology to stream video over the internet is called PD. PD uses HTTP over TCP in order to stream video content to the client's side. In this method, the video is packed in a container and encapsulated over the HTTP connection. The PD streaming server typically sends packets in the highest data rate possible with a constant payload size using the advantages of TCP's rate control mechanism. This way, video is encoded in constant bit rate (CBR), a very simple and reliable method. Video delivered using this technique can be played as soon as the player receives a small amount of data. However, while this technique is suitable for the client side, it isn't optimal from the ISP point of view. The PD technique requires a short time peak BW at the beginning of the transmission, in order to fill out the buffer at the client's side and avoid unsmooth video decoding (avoiding jitter phenomena). The average peak time in low resolution streams is ten seconds, but HD streams peak can last throughout the transmission. We have observed that the peak rate ratio compared to the CBR encoding rate can be up to 5 times more.

The initial peak video transmission creates two main problems. The first is the high BW demand from the server side, which leads to inefficient BW usage. The second problem is related to the fact that most clients tends to stop the streaming before it ends, meaning that data accumulated in the client's buffer will not be used.

Since ISP suffers from heavy congestion and limited BW resources, the PD method is not efficient in the sense of BW utilization. The purpose of this work is to improve the PD algorithm by reducing the initial peak rate and increasing the network's BW utilization, while enabling the client to obtain smooth video playback.

Another approach to video streaming that is gaining popularity at present is the Adaptive Streaming protocols (ASP) [2]. ASP implements various methods that adjust the video quality according to changing network conditions while transmitting, thus ensuring the best possible viewing
experience. ASP over HTTP is similar to PD: the stream is split into small chunks (each about a 2-10 second protocol, depending on the specific protocol implementation), and each chunk can be encoded into different bit rate and quantization variable bit rate (VBR). Based on the client's evaluation of network conditions, the suitable chunk is requested from the server. ASP, in general, consists of two different stages of operation: in the first approach, the "buffering mode", the player accumulates video content until an excessive amount of content has been buffered. Then the client automatically switches to the second stage, called "steady state", in which he will ask for a new chunk only when the previous one has been successfully received according to the new available network conditions. Using this buffer control technique, the protocol adapts the BW requirements while preserving the optimal QoE.

The ASP method, however, has one drawback. It is not efficient in the sense of memory requirement on the server side, due to several different encoding rates for the same video stream. Our suggested algorithm is a solution that combines the two approaches - DP and ASP. The proposed combination uses the advantages of both methods while using a single encoding of the original stream without any video encoding modification on the content provider side. This algorithm uses the simplicity of the first algorithm (PD) and the adaptive BW consumption of the second.

Traffic shaping is based on the ability to adjust the streaming data rate to the video data rate. The proposed algorithm can be regarded as an online video rate traffic shaping algorithm based on adaptive control over TCP window size. Traffic control/smoothing is usually attributed to VBR traffic. However, CBR traffic over TCP (PD), as previously mentioned, has a transferring rate of up to five times higher than the encoded CBR video rate traffic. Therefore, from the ISP's point of view, there is a need for traffic optimization from one hand, and from the other maintain high QoE.

There is a great deal of previous research regarding video rate smoothing algorithm [3-8]. However, most did not consider the streaming protocol, and is either more focused on the application level or based on preliminary video data rate knowledge, without consideration of real time environment.

Online smoothing for the live video streaming algorithm must have low time complexity, and usually deals with small chunks of real time video streaming. Smoothed video streams can be sent in a piecewise-CBR fashion [3]. Another suggested online method is VBR real time protocol (RTP), video rate smoothing at the proxy side [4]. On the one hand, this method has a huge advantage over others, because it does not dependant on the server side, which leads to better QoE. On the other hand, expensive memory resources are used to store the stream at the proxy server. Our proposed solution would work at the proxy side as well and optimized the traffic from the proxy to the client.

Offline smoothing is ideal for video on demand, many research have been done on this subject including, work-ahead smoothing [5], Enhanced Piecewise Constant Rate Transmission and Transport (e-PCRTT) [6]. Additional approaches to server side smoothing are "frame smoothed" streaming on the server side [7], and using a geometrical algorithm to determine optimal Leaky Bucket (LB) parameter for CBR video streaming [8]. However, this proposed algorithm needs extra information about the encoding.

It should be mentioned, that all of the aforementioned solutions are not suitable for ISPs because they depend on the will of the content video provider (server side) to shape their traffic.

A common solution for handling heavy congestion networks is based on the ability to identify the different network protocols and enforce QoS policies. Identification is based on DPI and enforced with different buffer algorithm such as LB. LB is a metaphor for a bucket with a hole: the bucket is full of water that drains in constant rate, which allows control over transmission speed. However, this requires a sufficient amount of memory, and while LB is considered to be a very reliable solution, it is not suitable to limited memory systems. Implementing traffic shaping in platforms with limited resources (buffers) can lead to major performance degradation.

Therefore, TCP window size control can be a suitable solution for real time systems with a limited buffer. A related work for QoS improvement using TCP window control and channel occupancy in wireless media [9] reduced the packet loss ratio of CBR traffic up to 45 percent. Our proposed solution controls and modifies the TCP window based on the stream encoded data rate. Window modification enables replacing the LB mechanism with TCP streams without dropping any packets, while adjusting to the client network conditions.

2 Proposed video rate shaping

2.1 Overview of the proposed shaping mechanism

Our solution is based on reading the PD video header, extracting the video data rate, and modifying the TCP window throughout the connection depending on the network condition, client buffer redundancy and video encoding rate.

Fig. 1 explains the general scheme of the proposed solution that consists of connection matching, DPI and TCP Window modification scheduler. The “Out gate” represents the opposite network interface card (NIC), through which we send the incoming packet to the other side of the network.

Connection matching purposes aim to control and manage the incoming traffic using 5-tuples: TCP/UDP, source/destination IP's and source/destination ports. Using the information from the first packet of connection, we redirected the packets to the appropriate next stage. If it is a new TCP connection or an ongoing connection that the DPI module didn't finish handling, then the next stage is DPI. If it isn't TCP or if the DPI module knows it is not PD then we forward the connection without any modifications. If it is a PD stream, we will redirect it to the TCP window modifier module.

The DPI module investigates the first packets of the connection, identifies each PD connection, updates
Connection Matching and calls the video container parser function. Parser is needed in order to extract the CBR data rate from the video header container. Here, we focused on the Adobe FLV [10] container, while using an FFmpeg [11] open source library as the base code for our container parser. A study in [12] reveals that online video is viewed in full in less than 50% of the cases. Considering a movie streamed using PD can be fully downloaded after a mere few seconds, under this assumption traffic is not optimized. Therefore, our algorithm optimizes the traffic where QoE is ensured and optimized unwatched traffic that can be wasted. Therefore, the TCP window modifier module is divided into two stages: buffering state and steady state. The buffering state ensures that the client will achieve a sufficient amount of redundant buffer, which we define as Threshold [sec]. When the client redundant buffer size is greater than Threshold, we switch to steady state mode. Steady state job is to restrict the client buffer and prevent unnecessary streaming traffic from being sent from the server to the client. The combination of both states enables the algorithm to adapt to changing network conditions and remain unaffected by short term network problems.

\[ W = \left\lfloor RTT * DataRate \right\rfloor \left\lceil \frac{Bytes}{Sec} \right\rceil \quad (2) \]

We gave discovered that (2) is insufficient because the server fills the window only if a full packet has enough space to enter. This is because PD stream servers use a fixed packet length size. Therefore, the optimized window size for streaming is (3.1). If (2) is used, then the stream is downloaded at a slower rate, which results in an unsmooth playing characteristic due to jitter effects. For example, given that \( W = 2500 \) and the Packet-Length = 1400 - because the window is too small for two packets, the streaming server will send a non-optimal window size with an actual size of 1400, which will cause an underflow of \( W_{uf} = 1100 \) Bytes. Another example: if \( W = 2900 \), then the \( W_{uf} = 100 \). This means we are \( W_{uf} \) short. In order to send a window that is suitable for our stream needs, it is required that compensate our window in order to send full window.

\[ W_{mod} = W \mod PacketLength \quad (3.0) \]

\[ If \left( W_{mod} == 0 \right) \{ W' = W + W_{mod} \} \]

\[ else \{ W' = W + \left\lfloor \frac{PacketLength}{W_{mod}} \right\rfloor \left\lfloor \frac{Bytes}{Sec} \right\rfloor \} \quad (3.1) \]

Therefore (4) underflow (\( W_{uf} \)) is the amount of data that was needed in order to send an optimize window. Overflow (\( W_{of} \)) is defined by the amount of redundant data that was sent in the current window (5) compared to (2).

\[ UnderFlow = W_{mod} \quad (4) \]

\[ OverFlow = W' - W \quad (5) \]

From analyzing the previous examples, we can see that in the second one our optimized formula for streaming purposes created an increased \( W_{of} \).

2.2 TCP window size and scale modification

After parsing the container header using the DPI module, the TCP scale [13], (1) and TCP window [14] is to be updated. The proposed solution changes the SYN packet scale value and sets it to zero. Traditionally, the TCP window is calculated by (2). This is the proposed minimal window size (2) that should satisfy the minimum needs of our traffic window shaper.

\[ Send.W = Segment.W \ll Send.W.Scale \quad (1) \]

Figure 1: General Scheme of the proposed algorithm

2.3 Modifier algorithm

Upon initialization of the algorithm, buffering mode is activated and the window size is set to \( W \). The client buffer is calculated with the consideration of the viewing progress over time. When the client buffer reaches a Threshold redundancy size, the steady state mode is activated. In order to control the client buffer we use two stages: soft shaping and aggressive shaping. In the first stage, TCP window size is modified with two kinds of windows: \( W \) and \( W' \). The use of the small window - \( W \), helps reduce and control the client buffer. However, in HD streaming the data rate is much higher, and aggressive temporary reduction of the window is needed (\( tempW' \) parameter). When control is achieved, adaptive increase of the client TCP window is necessary in order to preventing traffic peaks.
The following pseudo code illustrates the algorithm:

```
Buffering mode:
{  
WindowSize = W'
If (Client Buffer Size > Threshold)
  {Go to Steady state mode}
Else {Send modified packet and continue to Buffering mode}
}
Steady state mode:
{
  Gap = \left\lceil \frac{UnderFlow}{OverFlow} \right\rceil
  \text{Timer} = T_{size}
While \left( \left( \text{Timer} > 0 \right) \&\& \left( \text{Client Buffer Size} > \text{Threshold} \right) \right)
  {
    \text{// soft shaping stage}
    \text{Gap} = \text{Gap} - 1 \quad \text{// Gap minimum value is 1}
    \text{Send Gap \ modified packets with WindowSize = W' and one modify packet with WindowSize = W}
  }
If (Client Buffer Size < \text{Threshold})
  {return to Buffering mode}
While (\text{Client Buffer Size} > \text{Threshold})
  {
    \text{// aggressive shaping}
    \text{tempW'} = \frac{W'}{2}
    \text{Send modified packet with window size of tempW'}
  }
If (\text{Client Buffer Size} < \text{Threshold})
  {
    \text{// adaptive window size increases}
    \text{While (tempW' \leq W')}
      {
        \text{NumOfPackets} = 0
        While (\text{Client Buffer Size} < \text{Threshold}) \&\& (\text{NumOfPackets} < \text{P\_Gap})
          {
            \text{NumOfPackets}++
            \text{Send modified packet with window size = tempW'}
          }
        If (\text{Client Buffer Size} > \text{Threshold})
          {return to Steady state mode}
      }
    \text{tempW'} = \text{tempW'} + \text{incFactor}
  }
  \text{// gradual window size increase. Every P\_Gap packets}
Return to Buffering mode
}
```

Algorithm 1: Window size modification traffic data rate shaping.

We now turn to explaining. Gap is the compensation between the overflow and underflow. If the gap is relatively small, less smooth traffic will be observed (resulting in a larger amount of small peaks). However, faster control over the client buffer is achieved. If the gap is larger, then the traffic will be smoother, but controlling the client buffer will be harder. Moreover, the system reaction time will be longer. If after \( T_{size} \) seconds we cannot control the client buffer, we will switch to aggressive mode and adjust the window size to \( \text{tempW'} \). Large \( T_{size} \) slower system response time. In order to prevent unnecessary peaks in traffic, we adaptively increase the window size every \( \text{P\_Gap} \) packets. We used a \( \text{Threshold} = \) 25 seconds of client buffer size. Influenced by adaptive streaming, the switch time period from buffering state to steady state is after the client accumulates 20-30 seconds of video redundancy (depending on the algorithm that is being used). We have tested the algorithm with different threshold sizes, and found that it is quicker to smooth the traffic when the client buffer size is small. The drawback of small threshold is that the traffic is less smooth. After hundreds of network testing with different ISPs, we found out that the optimized parameters best fitted from our standard deviation (STD) tests are: \( T_{size} = 3, \text{P\_Gap} = 20 \) and \( \text{incFactor} = 10 \). However, the location of the implemented system can have different optimized values due to changing network conditions. Therefore, the system needs an initial optimization in order to a chive the desired results.

2.4 Implementation

The experiments were conducted on Ubuntu version 10.10, Intel I5-2300 2.8 GHz 8 GB RAM with 2 network interface cards, one for an internal network and the other for an external (WAN). The software was written in C/C++. In order to forward packets from one side to the other we used libpcap library [15]. The ADSL line capacity was 5Mbps link with download rate of 5.33 Mbps and upload rate of 0.7 Mbps. The following are the assumptions used to test our software:

- **Assumption 1** - The user will not skip to a later segment in the video. Skipping will close the current connection and open a new connection from the new segment point (assuming that the segment wasn’t downloaded). This is the typical way PD works.
- **Assumption 2** - The user will not change the video resolution while watching (same explanation as assumption 1).

Due to space limitations, in this paper we will present only result with video resolution of 630 x 430.

3 Results

3.1 Results without the algorithm

In this case, where the stream was downloaded without shaping stream-1[16] was used. Fig. 2 shows that the ratio between the maximum download rate and the desired data rate is 7.

3.2 Shaping without buffer control

In this mode the algorithm only shape the traffic with the desired window (3) without steady state mode. In the
given example (Fig. 2), the stream is downloaded very fast because the download isn't optimal and the ISP suffers from overloading and high throughput on his line. It can be seen in Fig. 3 that we managed to minimize the maximum peak due to our TCP scaling and window modification. It is possible to decrease the maximum peak even more but in exchange to increasing initial play out delay. It can also be seen that we are above the desired data rate due to the overflow done by (3). Experiments have shown that using (2) will cause the stream to get stuck while downloading.

It can also be seen that we are above the desired data rate due to the overflow done by (3). Experiments have shown that using (2) will cause the stream to get stuck while downloading.

3.3 Shaping with buffer control

Fig. 5 presents the full algorithm, where at 32 seconds the algorithm switches to steady state mode. From Fig. 4 we can see that the maximum difference in the client buffer without the algorithm compared to shaping with buffer control is doubled. Furthermore, we can see that without shaping mode the client reached a 25 seconds buffer size in only 7.81 seconds. Table .1 sums up the statistical difference between the proposed methods.

Full video demonstration of the proposed solution with Threshold size of 5 [sec] can be seen here [17].
4 Conclusions

In this work, a novel method and system for providing video data rate shaping for PD videos was introduced. The proposed solution is especially useful for mobile and 4G networks with existing or new DPI systems, as well for proxy servers. Since DPI systems have limited computational and memory resources, our proposed solution eliminates the buffer requirements and reduce the computational complexity.

5 Acknowledgment

We are grateful to Chen Hadad and Eyal Ron for their testing, evaluating and important inputs throughout this project. We would like to thank Tomer Margolin, Tal Vazana and Ofer Hermoni for their support and important feedback.

This work was supported by the EU MEDIEVAL research project. MEDIEVAL (MultiMEDia transport for mobile Video AppLications) [18] is a medium-scale focused research project (STREP) of the 7th Framework Program of the European Commission, addressing the core of the strategic objective “The Network of the Future”.

6 References


[16] Avatar Trailer video can be watched at: http://www.youtube.com/watch?v=d1_JBMrrYw8.

[17] This Paper video example can be seen at: http://www.youtube.com/watch?v=B85L8KZODkQ&feature=mh_lolz&list=FLSHKdmosPXDR0gOk8vUeZzg.
