Empirical Evaluation of HTTP Adaptive Streaming under Vehicular Mobility

Jun Yao, Salil S. Kanhere, Imran Hossain, and Mahbub Hassan

School of Computer Science and Engineering
University of New South Wales, Sydney, NSW 2052, Australia
{jyao,salilk,imranh,mahbub}@cse.unsw.edu.au

Abstract. Adaptive streaming is a promising technique for delivering a high-quality video streaming experience. In this technique, the streaming bit-rate is constantly adjusted in accordance to the variations in the underlying network bandwidth conditions. A popular instantiation of this approach is to extend traditional HTTP-based streaming. While several such implementations are widely available, it is unclear how they perform under a typical high-speed vehicular environment, wherein the wireless bandwidth varies significantly and rapidly. In this paper, we seek to provide some insights on this issue through empirical experiments driven by real-world wireless bandwidth traces collected from moving vehicles. Our results suggest that, with appropriate parameter configurations, HTTP adaptive streaming is an effective solution for delivering a high-quality smooth streaming experience even under high-speed vehicular mobility.

Keywords: Adaptive streaming, HTTP streaming, Empirical experiments, Vehicular mobility, Mobile computing

1 Introduction

With the success of websites, such as YouTube and Vimeo, video streaming over the Internet has become increasingly popular. A recent report [19] estimates that the streaming media business will grow by 27% per year, generating over US$78 billion in revenue in the U.S. alone over the next six years. Given the widespread coverage of high-data rate Wireless Wide Area Networks (WWAN) and the emergence of personal mobile devices with high resolution displays and fast processing speeds (e.g., smartphones and tablets), multimedia streaming is now one of the fastest growing mobile applications.

Till recently, video streaming technologies have mostly adopted the non-adaptive approach, wherein the bit-rate and quality of the video are selected prior to the start of the streaming, and fixed during a streaming session. This is usually sufficient when the viewer is connected via a wired connection, which has high and fairly stable bandwidth capacity. However, in the context of vehicular mobility, where a viewer on a moving vehicle watches the video stream on a mobile device (phone, tablet or laptop) connected to the Internet via a WWAN connection, the non-adaptive solution may not be optimal. This is because, the

WWAN bandwidth conditions in such an environment fluctuate significantly, even across smaller time scales. This is due to the heterogenous radio characteristics and WWAN network load conditions at different locations as a vehicle makes its way along the route [7]. The ensuing bandwidth fluctuations in such a dynamic environment can seriously compromise the QoS experienced by traditional streaming applications. For example, during a live streaming session, if the WWAN bandwidth suddenly drops significantly below the selected streaming rate, the video viewer can suffer from noticeable "glitches" caused by frame loss or frequent pauses in playback caused by the exhaustion of the playout buffer.

To mitigate the effect of the bandwidth dynamics and achieve the smooth streaming experience, adaptive streaming is emerging as the promising solution [2–4, 9–12, 16, 18, 20]. In adaptive streaming, the streaming servers and/or clients constantly monitor real-time end-to-end network conditions. Based on this information, the video bit-rate and quality are dynamically adjusted to adapt to the changes in underlying network conditions. As a result, adaptive streaming delivers a better quality video stream as compared to conventional non-adaptive approaches. The principle of adaptive streaming has been incorporated with traditional streaming protocols, such as RTSP [11], and evaluated in various prior works [16, 18, 20]. However, due to the popularity of using HTTP in video streaming, the common trend in the industry is to use adaptive HTTP progressive download [2–4, 9–12]. Several such products have even been released by Move Networks, Microsoft, Adobe and Apple. Despite the popularity, a comprehensive empirical evaluation of HTTP adaptive streaming is lacking. In particular, the performance of the HTTP adaptive streaming under high-speed vehicular mobility has not been reported.

Our goal in this paper is to address the above issue and empirically evaluate the performance of HTTP adaptive streaming under vehicular mobility. However, conducting "live" experiments in which the streaming client operates on a mobile device connected to a WWAN network and travelling in a vehicle has several problems. To comprehensively evaluate the effect of varying certain parameters on the streaming performance, we would need to make separate trips, each with a different value of the parameter. However, the WWAN network bandwidth is known to vary from trip to trip [21]. As such, it would be impossible to study the effect of changing parameters in isolation. Further, conducting sufficient tests with statistical reliable results would require making a large number of trips, which is prohibitively expensive in man hours and monetary costs (fuel and WWAN bandwidth subscription). In this paper, we have hence opted to conduct the experiments in a controlled lab environment, but using "real" WWAN bandwidth traces that are empirically collected from a moving vehicle. In our experiments, we report the performance of HTTP adaptive streaming under the effect of various parameters, such as buffer threshold and video chunk size. Our results show that adaptive HTTP streaming is effective in reacting to the widespread fluctuations that are inherent in vehicular mobility and ultimately achieving an improved viewer experience by over *four-fold*. Based on our evaluations, we make recommendations on suitable values of the key parame-

ter settings for achieving optimal performance, which can be of use to industry practitioners.

The rest of this paper is organized as follows. Section 2 discusses the background and related works. In Section 3 we present the vehicular measurement campaign for collecting the bandwidth traces used in this paper. We present the setup of our trace-driven experiments in Section 4. The findings of our experiments are presented in Section 5. Section 6 concludes this paper.

2 Background and Related Works

In this section, we review the state-of-the-art in multimedia delivery in the Internet. We start with the discussion on the traditional non-adaptive streaming approaches. Then we discuss the emerging adaptive techniques with a particular emphasis on HTTP adaptive streaming.

2.1 Non-adaptive Streaming

Most traditional streaming services rely on specialized protocols that are specifically designed for streaming. RTSP (Real Time Streaming Protocol) [17] is a typical example of such protocols. After a streaming session has been established, the media file is sent as a stream of small fixed-sized packets. The server usually sends enough data to fill the client's buffer (typically less than 10 seconds). This traditional streaming approach can be used to stream both on-demand and live video. However, it requires the deployment of dedicated media servers and uses special ports. This may have issues with scalability, caching and penetrating client firewalls [11]. Another popular method for video steaming is *HTTP progressive download*. The idea is similar to a normal file download from an HTTP server. The only difference is that the video can be simultaneously played back while it is being downloaded. This also means that even if the user pauses the media, the server will continue sending data until the whole file is downloaded. Note that, this differs from the first approach explained earlier, where the server stops transmission of packets if the client buffer is filled. The apparent drawback of the HTTP method is that if the user decides to terminate the session, then all stored (but un-viewed) video in the buffer is discarded, thus wasting the bandwidth that was used up in transferring this data. In addition, a significant disadvantage of HTTP progressive download is that it cannot support live streaming, as the size of the entire video is predetermined and fixed during the streaming session [11]. However, the HTTP approach is a cost-effective solution for on-demand video delivery, as it reuses the existing HTTP caches/proxies without the need for specialized servers. Also, the use of HTTP protocol eliminates the issues with firewalls [4, 11]. Nowadays, most popular video sharing websites, e.g., Youtube and Vimeo, exclusively use this approach.

Both of the aforementioned techniques are *non-adaptive*, in that the streaming quality and bit-rate remain unchanged during the entire duration of the streaming. The choice of these parameters is either determined automatically

or based on the user's preference at the start of the session. This may serve well in the stationary and wired network environment, where the bandwidth is relatively constant. However, in the context of vehicular mobility (which is the focus of this paper), the wireless bandwidth is known to fluctuate significantly as the vehicle changes its location. Hence the non-adaptive approach can be significantly affected by the bandwidth fluctuations and leads to jerky video playback and ultimately impacts the user viewing experience (see results in Section 5.1).

2.2 Adaptive Streaming

In contrast with the traditional methods, adaptive streaming allows dynamic adjustments in the streaming bit-rate (and quality) to adapt to the varying end-to-end bandwidth conditions, during a live session. Figure 1 illustrates a typical scenario. An adaptive streaming server hosts the videos that have been encoded at various bit-rates. These files are created by encoding the same source video multiple times by varying the quantization and frame rate settings, or using advanced encoding techniques, such as Scalable Video Coding (SVC) [10]. Each of the encoded video streams are then partitioned into a sequence of small "chunks", e.g., Group of Pictures (GoP), which can be decoded independently [10]. Since the video chunks partitioned under different qualities are synchronized, there are various bit-rates available for each video chunk. During streaming, the server can seamlessly switch the streaming quality by using different bit-rates for each video chunk. For example, the server can switch to sending chunks with lower bit-rate when it detects that the available bandwidth reduces, thus gracefully degrading the viewing quality. Note that, the adaptive principle can be readily applied to both streaming-specific protocols and HTTP streaming.

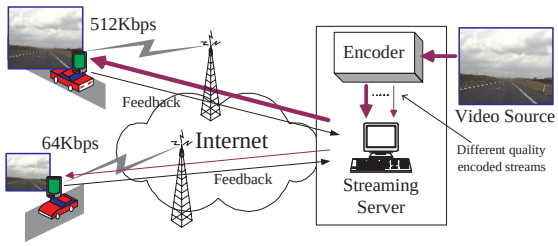


Fig. 1. Adaptive video streaming in a vehicular mobility scenario

Prior works [16,18,20] have proposed mechanisms for implementing adaptive streaming by extending traditional streaming protocols. The basic concept involves the server and/or client relying on some rate control algorithms/protocols to infer the actual bandwidth conditions. Based on this estimate the server selects the next chunk with the appropriate quality from the choices available. Discussion on several such rate adaptation algorithms have been proposed [8, 10]. In general, the network conditions are inferred in response to the changes in observable metrics, such as the occupancy of video playout buffers or certain end-to-end network parameters (i.e. delay, packet loss and throughput).

Due to the advantages mentioned in Section 2.1 of using HTTP in video delivery, HTTP-based adaptive streaming has drawn significant interest in the industry. In HTTP adaptive streaming, the entire video streaming is split into many small progressive downloads, wherein each HTTP transaction delivers one video chunk to the client. After a session is established, the server returns a manifest file to the client. The manifest file lists the available bit-rates (typically 4-5) for each video chunk [4, 11]. On receiving each chunk, the client measures the throughput to estimate the end-to-end bandwidth conditions. Based on the estimated bandwidth and playout buffer conditions, the client determines the suitable bit-rate and requests the appropriate video chunk through a HTTP GET request. Note that, since each video chunk is individually encoded and transmitted, the streamed video sizes can be unbounded. Thus, unlike its predecessor (HTTP progressive download), HTTP adaptive streaming also fully supports for streaming live events. Recently, Adaptive HTTP Streaming (AHS) has been integrated into the 3GPP Transparent end-to-end Packet-switched Streaming Service (PSS) [2]. 3GPP AHS also has been adopted by Open IPTV Forum (OIPF) as the core component of their adaptive streaming solution [14]. MPEG is also drafting a new standard called Dynamic Adaptive Streaming over HTTP (DASH) [9]. Driven by the popularity, commercial HTTP adaptive streaming products, such as Move Networks [12], Microsoft Smooth Streaming [11], Apple HTTP live streaming [5] and Adobe Dynamic HTTP Streaming [3], have been made available. This technique has been successfully used by NBC in broadcasting the Beijing Olympic Games in 2008 [11].

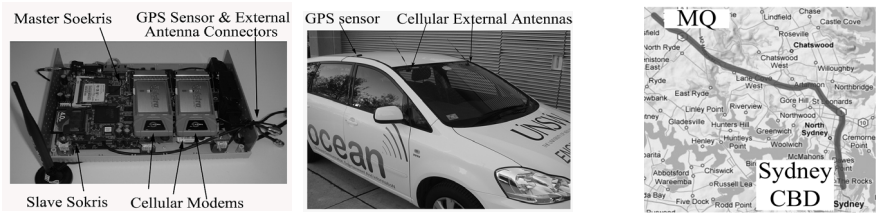
The performance of the non-HTTP adaptive streaming techniques have been evaluated and studied in previous works [16, 18, 20]. However, evaluations were conducted using simulations with synthetic bandwidth data. Despite the popularity, a detailed empirical evaluation of HTTP adaptive streaming is lacking, to the best of our knowledge. In this paper, we intend to fill this gap and investigate the performance of HTTP adaptive streaming under vehicular mobility.

3 Bandwidth Trace Collection

The goal of this paper is to evaluate adaptive HTTP streaming in a real-world vehicular mobility scenario. The obvious approach is to implement a real client and server and conduct live tests while driving. However, there are several problems in conducting such experiments. To comprehensively evaluate the effect of changing certain parameters (such as playout buffer threshold, video chunk size) on the streaming performance, we would need to make separate trips, each with a different value of the parameter. However, it is known that the WWAN network bandwidth can vary from trip to trip even when the successive trips are made one after the other (see [21] for a detailed investigation on bandwidth variability). As such, it would be impossible to study the effect of changing parameters in isolation. Further, even if the bandwidth remained fairly stable across different trips, conducting all the tests that we have conducted in Section 5 to achieve sufficient statistical significance would require us to make hundreds of

trips. Clearly, the amount of man hours and costs (fuel, WWAN subscription) involved are prohibitively high to conduct such tests. Hence, we have opted to conduct trace-driven evaluations in a controlled lab setting. For this, we use the empirical bandwidth traces collected from our previous wardriving campaign [21] in Sydney. In the following, we briefly review the details of the campaign.

For the measurements, a simple client-server measurement system (further details can be found in [21]) was developed using off-the-shelf hardware. The server was housed in our lab at the University of New South Wales (UNSW). As shown in Fig. 2(a), the client comprised of two Soekris Net4521 boards interconnected via 10 Mbps Ethernet. The boards were enclosed in a protective casing and housed in the boot of a car. Three PCMCIA cellular modems were housed in the system. To enhance the wireless signal reception, the cellular modems were connected to external antennas mounted on the car windshield. We measured two HSDPA [1] networks and one iBurst [6] network. A Garmin GPS sensor was connected to the system for recording the vehicle location.



(a) measurement setup (b) route trajectory

Fig. 2. Bandwidth measurements under vehicular mobility.

We developed a lightweight packet-train utility to measure the WWAN bandwidth. We refer readers to [21] for further details and validations. We collected one bandwidth sample for approximately every 200m section of the route. The samples are tagged with location coordinates and time, and stored in a repository. On occasions, some bandwidth samples were missing due to burst packet loss. To mask the effect of missing samples, we use the average bandwidth of all raw samples collected within each 500m segment to represent the bandwidth for the segment. Hence the granularity of the bandwidth traces is 500m.

We have collected bandwidth samples by driving the car along two representative daily commuting routes in the Sydney metropolitan area. During the eight-month measurement period, we have collected 75 traces of WWAN bandwidth along both routes. Fig. 2(b) presents the trajectory of the route that is used in this evaluation study. The chosen route (16.5 Km) runs from Sydney CBD to Macquarie University (MQ). In our experiments, we use the bandwidth traces from all 75 trips for one of the HSDPA networks.

4 Experiment Setup

The goal of this study is to evaluate the performance of HTTP adaptive streaming under real-world bandwidth conditions in high-speed vehicular mobility. We

consider the scenario that a passenger in a vehicle watches a streaming video on his mobile device (phone, tablet or laptop) while driving from location A to B. We assume that the viewer watches the video for the entire trip.

We implemented an adaptive HTTP streaming client-server prototype in JAVA and conducted experiments using our empirical bandwidth traces in a controlled lab environment (see discussions in Section 3). Figure 3 presents the experiment setup. The experiment involves 3 Linux (Ubuntu 10.04) machines. The HTTP server and video files are hosted at the server machine. The bandwidth shaper emulates the bandwidth changes of the HSDPA link according to the bandwidth trace files. The client initiates the streaming session and requests video chunks using HTTP from the server as discussed in Section 2.2.

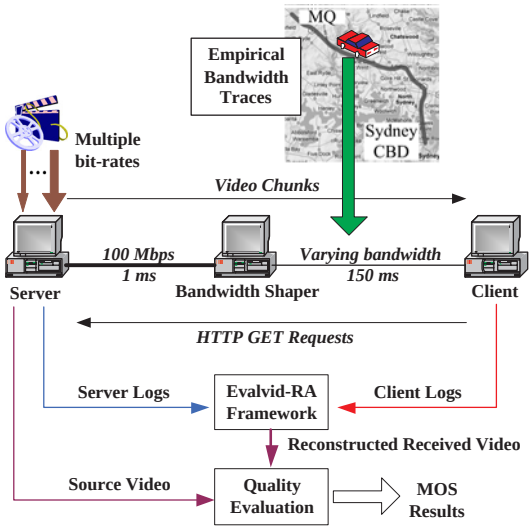


Fig. 3. Experiment setup

Since the wired links in the Internet and the HSDPA core network have sufficiently high bandwidth and small delays as compared to the last-hop HSDPA link, the server and the bandwidth shaper are connected via a static 100 Mbps Ethernet link with a 10 ms propagation delay to represent the wired Internet. The client and the bandwidth shaper are also physically connected with a 100 Mbps Ethernet link. However, in order to emulate the bandwidth changes as the vehicle travels along its route, we use the system utility *tc* at the bandwidth shaper to throttle the bandwidth of the link between the bandwidth shaper and client. In each experiment run, we emulate one driving trip, wherein the bandwidth shaper varies the bandwidth at each location according to the corresponding empirical bandwidth trace for the trip (collected in Section 3).

For the video clip, we create a looping video (using the medium motion QCIF sequence "Foreman" [10]) that lasts for 40 minutes, which is sufficiently longer than the duration of all trips. Before the experiment starts, the video is pre-

encoded at 31 different quantization qualities (corresponds to bit-rates from 80 Kbps to 1 Mbps) using MPEG-4 codec. Note that, commercial products normally use smaller number of bit-rates to save disk space and facilitate file management. The encoded videos are partitioned into a series of equal size video chunks as discussed in Section 2.2. Note that, the size of a chunk is configurable [11]. In Section 5.3, we will investigate the effect of using different chunk sizes.

At the beginning of a trip, the client initiates the session with the server. The server sends the manifest file of the entire video to the client. This file contains all (31, in our experiments) available bit-rates for each video chunk, which can be used during streaming. The client always requests for the lowest bit-rate available for the first chunk. On receiving a chunk from the server, the client measures the instantaneous throughput of the chunk to estimate the bandwidth conditions. The client determines the bit-rate (from those listed in the manifest file) to be used for the next video chunk, based on the estimated bandwidth and the occupancy of the playout buffer. Note that, all received video chunks will be first stored into the playout buffer before being played back. Initially, the buffered video needs to reach an initial playout buffer threshold b_i before the playback starts. Note that, rebuffering will occur whenever the playout buffer is underrun. The playback will be paused until the buffer occupancy reaches b_i. To warrant seamless streaming, the simplest bit-rate selection strategy is to always choose a chunk with a bit-rate that is lower than the current estimated bandwidth. However, this can result in constant increase in the buffer occupancy, if video chunk arrival rate is constantly greater than the playback rate. Excessive buffering is not desired in adaptive streaming, as all un-viewed video in the buffer is discarded, if the user terminates the session. This not only wastes the available bandwidth for transferring the buffered data but also misses the opportunity to deliver better quality video to the client, i.e., by using higher bit-rates. Hence, commercial HTTP adaptive streaming implementations are known to apply some flow control mechanisms to maintain a reasonable buffer occupancy [4, 11]. However, the detailed mechanisms are proprietary. In this paper, we have used a simple threshold-based scheme for this purpose. During streaming, the client consults the buffer occupancy before selecting the bit-rate of the next video chunk. When the buffer occupancy is lower than the threshold b_f, the client selects the highest bit-rate that is lower than the current estimated bandwidth. Otherwise, the client selects the lowest bit-rate that is higher than the current bandwidth. In this case, the arrival rate of the video chunks is expected to decrease, since the selected bit-rate is greater than the available bandwidth. We have assumed the buffer control threshold, $b_f = 1.5 \times b_i$. The coefficient of 1.5 was selected based on our pilot experiments (excluded for reasons of brevity).

For evaluating the video quality, we use the Evalvid-RA framework [10], which is well-accepted for evaluation of the quality of video transmitted over a real or simulated communication network. During streaming, we log the necessary information at the server and client according to the framework. After the experiment is finished, the log files are used to reconstruct the received video sequence. Recall that, during instances of rebuffering, the playback is paused. For

those instances, we fill in the paused periods by copying the last played frame. The similar approach is often used by media players to deal with lost frames. We use the tools available from Evalvid-RA to process the source and received videos and generate the Peak Signal-to-Noise Ratio (PSNR) [10] for each frame. To get better insights about the actual viewing experience, we estimate the Mean Opinion Score (MOS) from the PSNR value for each frame. In particular, we employ the empirical model [13] obtained for the specific Foreman sequence used in our experiments,

$$\hat{MOS} = k \times (a - \frac{b}{PSNR}), \tag{1}$$

where $k = 0.56$, $a = 14.2$ and $b = 280.5$. Note that, MOS ranges from 0 to 5. A MOS of 5 suggests an "excellent" viewing experience, while 0 being completely unacceptable. Hence the MOS was set to 0 for the frozen frames during buffering.

5 Experimental Results

In this section, we discuss our findings from the trace-driven experiments. As discussed in Section 4, we calculate MOS to evaluate the viewer streaming experience. It is reported that humans can perceive a drop in the streaming quality, when the MOS remains below 3 consistently for one second or longer [15]. We refer to such an event as a *video glitch*. Clearly, reducing the number of glitches experienced by a viewer directly improves the QoS of a video streaming session. For understanding the effect of glitches on the viewing experience, we define a metric, *glitch duration*, which measures the cumulative time over which glitches occur during a session. Another important aspect of the user viewing experience is the stoppages encountered due to buffering events. As such, we define two metrics to evaluate the impact of buffering. The first metric is the *total number of buffering events* encountered during a viewing session. The second metric is the *buffering duration*, which measures the cumulative time when buffering occurs during a session. Note that, the lower the values of the above metrics (glitch duration, number of buffering events and buffering duration), the better is the viewing experience. In the following, we first compare the performance of the non-adaptive and the adaptive HTTP streaming schemes under vehicular mobility. Further, we study the effect of important streaming parameters, i.e., the buffer threshold and video chunk size, on the adaptive streaming performance.

5.1 Adaptive vs. Non-adaptive

We first investigate how HTTP adaptive streaming can cope with the bandwidth fluctuations under high-speed vehicular mobility. For comparison, we also present the results for non-adaptive streaming scheme. The non-adaptive scheme streams at a constant target bit-rate of 420 Kbps, which is approximately equal to the empirical mean HSDPA bandwidth observed from our traces. For the HTTP adaptive scheme, we set both chunk size and buffer threshold b_i to be 2 seconds, which are the recommended settings in [11].

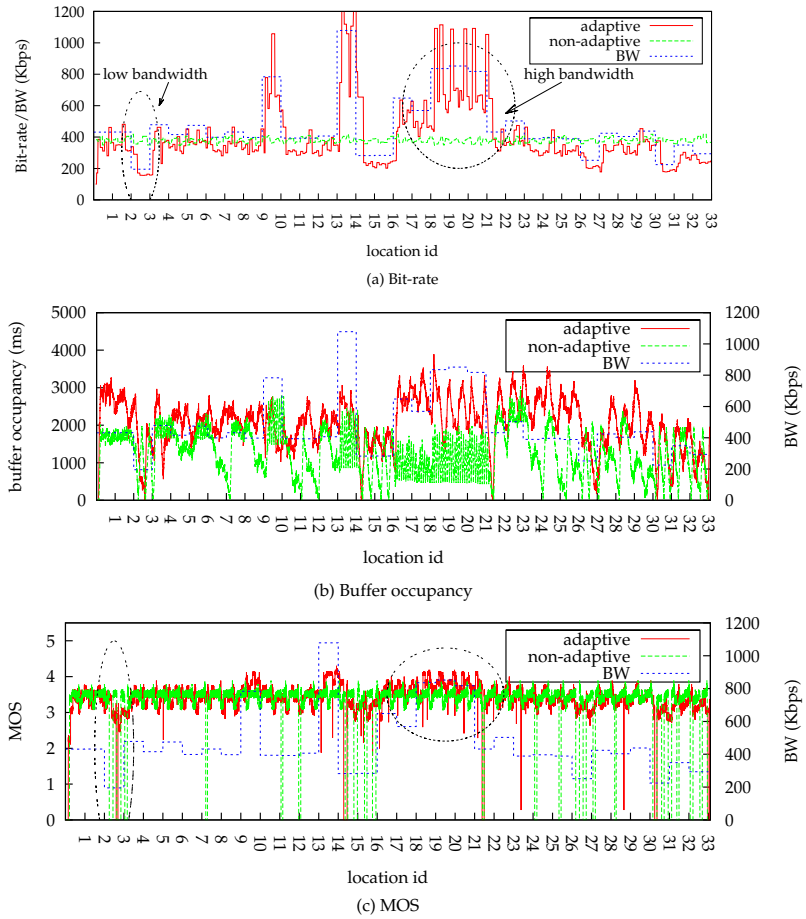
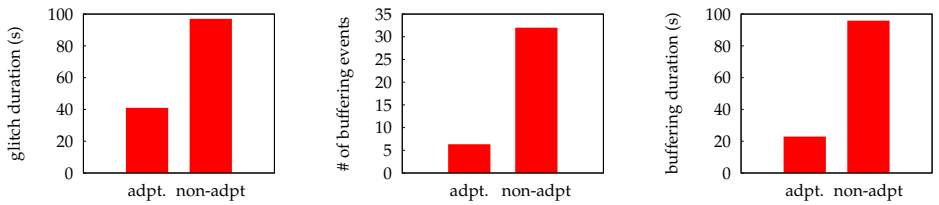


Fig. 4. Microscopic behavior of adaptive and non-adaptive schemes during trip #65.

To gain insights into the behavior of HTTP adaptive streaming, we first present the instantaneous bit-rate, buffer occupancy and MOS results as the vehicle travels along the route during one particular trip in Fig. 4. We observe similar results with the traces from other trips. Fig. 4(a) shows that the HSDPA bandwidth keeps fluctuating along the route. For example, the bandwidth drops to 200 Kbps when the vehicle enters location #3, whereas in location #19-#21, the bandwidth increases to 800 Kbps. As is evident, despite the variations in the bandwidth, the non-adaptive scheme keeps sending at near constant bit-rate. At location #3, the steady bit-rate stream overloads the link, which in turn leads to congestion. Fig. 4(b) shows that in this instance, the non-adaptive scheme causes repetitive occurrences of buffer underrun. Hence, a viewer experiences jerky playback as the video pauses to re-buffer frequently. Note that, due to the frozen playback during buffering, the MOS score drops to 0 as shown in Fig. 4(c). In addition, when the HSDPA bandwidth increases between location #19-#21, the non-adaptive scheme is not able to stream the video at a higher

bit-rate, as shown in Fig. 4(a). This wastes the opportunities of utilizing the extra bandwidth available to achieve better picture quality. On the other hand, Fig. 4(a) demonstrates that the adaptive scheme varies its bit-rate in accordance to the bandwidth variations. This avoids congestion and draining the playout buffer, when the bandwidth drops, which in turn results in minimal re-buffering. For example, we only observe 1 instance of buffering in location #3, whereas the non-adaptive scheme leads to 3 such instances. When the available bandwidth is high, the adaptive scheme is able to increase its bit-rate to achieve the best quality streaming possible (e.g., in location #19-#21). Fig. 4(c) shows that in these instances, the MOS of the adaptive scheme is significantly higher than that of the non-adaptive. Note that, the higher MOS directly implies better picture quality. Observe that, occasionally the bit-rate of the adaptive scheme exceeds the available bandwidth. This is due to the buffer control mechanism used in our implementation (i.e., increase the bit-rate when the buffer reaches b_f).



(a) glitch duration (b) # of buffering events (c) buffering duration

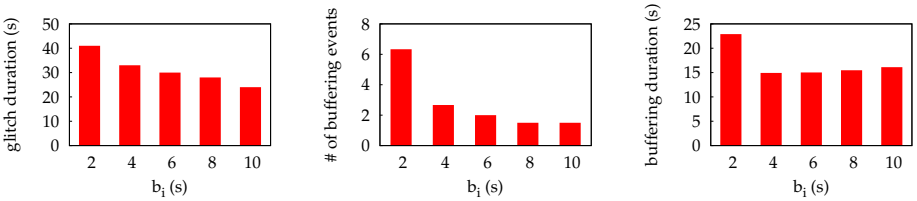
Fig. 5. Comparing adaptive and non-adaptive streaming.

The above microscopic analysis of a particular trip reveals that the adaptive approach can effectively reduce the buffering when bandwidth drops and utilize the available bandwidth when bandwidth increases. The mean results over all 75-trip experiments are shown in Fig. 5. Note that, the average duration of a session is about 25 minutes. Fig. 5(a) shows that the adaptive scheme effectively reduces the glitch duration by over 50% as compared to the non-adaptive scheme. More significantly, the number of buffering occasions (Fig. 5(b)) and the buffering duration (Fig. 5(c)) is reduced by 80% and 70%, respectively.

5.2 The Effect of Playout Buffer Threshold

In this set of experiments, we study the effect of playout buffer threshold b_i on HTTP adaptive streaming. Recall that, in the previous experiments (Section 5.1), b_i was set to 2 seconds. Fig. 6 shows the mean results for all three metrics averaged over all 75 trips. Observe that, the glitch duration decreases when a larger b_i is used. This is because a larger b_i allows storage of more video into the buffer, hence better absorbing the bandwidth variations along the route. Fig. 6(b) shows that by setting b_i greater than 6 seconds, the number of buffering events reduces to about only 1 per trip, which is simply due to the initial buffering at the start of the session. This shows that, by using a small buffer of less

than 10 seconds, HTTP adaptive streaming can achieve a near un-interrupted streaming experience during a vehicular trip. However, even though the number of buffering event reduces, Fig. 6(c) shows that using a b_i greater than 4 seconds does not further reduce the buffering time. This is due to the fact that by using a larger b_i, a viewer generally spends more time for the initial buffering at the beginning of the trip. This effect is shown in Fig. 7. Further, using a larger play-out buffer can also be an issue for memory constrained mobile devices, such as mobile phones. As a result, the buffer threshold needs to be carefully tuned, so that it would not lead to lengthy buffering and memory issues, while effectively smoothing out the bandwidth variations under vehicular mobility.



(a) glitch duration (b) # of buffering events (c) buffering duration

Fig. 6. The impact of using different buffer threshold.

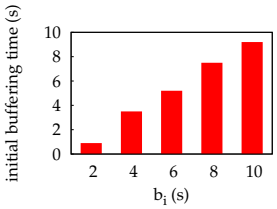
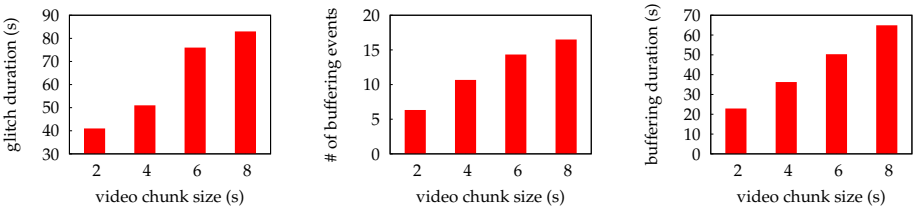


Fig. 7. Initial buffering time under different buffer threshold.

5.3 The Effect of Video Chunk Size

The video chunk size is also an important parameter in HTTP adaptive streaming. Recall that, each video chunk is stored as an individual file on the HTTP server. Since multiple bit-rates are available for each video chunk, a small chunk size such as 2-second can result in tens of thousands of small files for an hour-long video. This can pose file management issues for video content distributors [11]. Thus, larger chunk sizes are recommended [4]. However, the large chunk size setting may not be sufficient to adapt to the rapid bandwidth fluctuations encountered in vehicular mobility. To understand the effect, Fig. 8 plots the mean results for all the 3 metrics as a function of the video chunk sizes (with a 2-second buffer threshold). Clearly, the glitch duration (as in Fig. 8(a)) increases significantly with the increase in the chunk size. Note that, using a larger chunk size requires more time to transfer each chunk. Since the streaming bit-rate can be only varied once a new chunk is fully received, the larger chunk size reduces the agility of the streaming client in tracking the underlying mobile bandwidth. This

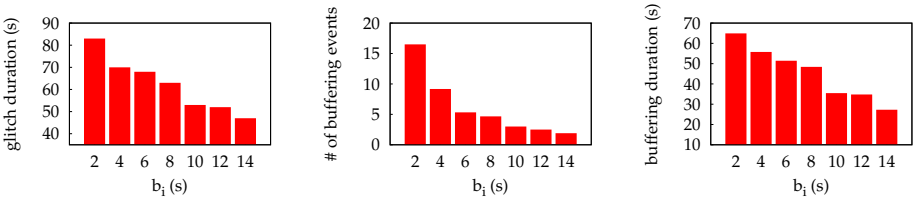
further leads to the increase in the number of buffering events and buffering time as shown in Fig. 8(b) and (c). For example, even when a 8 second video chunk is used, both the number of buffering events and the buffering time increase by nearly three-fold, as compared to a 2-second chunk size.



(a) glitch duration (b) # of buffering events (c) buffering duration

Fig. 8. Impact of using different video chunk sizes.

Recall that, in Section 5.2, we have observed that using a larger buffer threshold achieves better streaming performance in the face of frequent bandwidth fluctuations. Fig. 9 presents the experiment results when incorporating the large video chunk size (8 seconds) with different buffer threshold settings. As is evident, a b_i as large as 14-second is required to effectively reduce glitches and buffering. This highlights that it is important to configure the buffer threshold in accordance with the video chunk size in use, particulary for the scenario under consideration where the bandwidth fluctuates frequently and rapidly. Table 1 lists our recommendations on the buffer threshold for different chunk sizes based on the evaluations.



(a) glitch duration (b) # of buffering events (c) buffering duration

Fig. 9. Impact of larger chunk size (8 s) with different buffer threshold.

Table 1. Recommended buffer thresholds for different video chunk sizes

video chunk size (s)	2	4	6	8
recommended b_i (s)	*8*	*10*	*12*	*14*

6 Conclusion

In this paper, an empirical evaluation of HTTP adaptive streaming under vehicular mobility has been presented. We have implemented a HTTP adaptive

streaming prototype and conducted evaluation experiments using "real" WWAN bandwidth traces that are empirically collected from a moving vehicle. We have investigated the performance of HTTP adaptive streaming under the effect of different parameters, such as buffer threshold and video chunk size. Our results have shown that HTTP adaptive streaming is effective in responding to the widespread fluctuations that are inherent in vehicular mobility and ultimately achieving an improved viewer experience. We have highlighted that it is possible to achieve a smooth and high-quality streaming experience, by using appropriate streaming parameter settings.

References

1. 3GPP: High Speed Downlink Packet Access - Overall description - Stage 2
2. 3GPP TS 26.234: Transparent end-to-end packet switched streaming service (PSS); Protocols and codecs (2010)
3. Adobe Systems Incorporated: Dynamic Streaming
4. Akamai: Akamai HD for iPhone Encoding Best Practices - Akamai HD Network
5. Apple Inc.: HTTP Live Streaming Overview
6. Arracomm Inc.: iBurst Broadband Wireless - System Overview
7. Derksen, J., Jansen, R., Maijala, M., Westerberg, E.: HSDPA Performance and Evolution. Ericsson Review (03), 117–120 (2006)
8. Floyd, S., Handley, M., Padhye, J., Widmer, J.: TCP Friendly Rate Control (TFRC): Protocol Specification. RFC5348 (Sep 2008)
9. ISO/IEC CD 23001-6: Information technology – MPEG systems technologies – Part 6: Dynamic adaptive streaming over HTTP (DASH) (2011)
10. Lie, A., Klaue, J.: Evalvid-RA: Trace Driven Simulation of Rate Adaptive MPEG-4 VBR Video. ACM/Springer Multimedia Systems Journal 14 (Nov 2007)
11. Microsoft Corporation: IIS Smooth Streaming Technical Overview
12. Move Networks: `http://www.movenetworks.com/`
13. Nemethova, O., Ries, M.: Quality assessment for h.264 coded low-rate and low-resolution video sequences. In: Proc. of Conf. on Internet and Inf. Tech. (2004)
14. OIPF: Specification Volume 2a - HTTP Adaptive Streaming (2010)
15. Orlov, Z.: Network-driven Adaptive Video Streaming in Wireless Environments. In: Proc. of IEEE PIMRC08. Cannes, France (Sep 2008)
16. Schierl, T., Wiegand, T., Kampmann, M.: 3GPP compliant adaptive wireless video streaming using H.264/AVC. In: Proc. of IEEE Intl. Conf. on Image Processing. Rio de Janeiro, Brazil (2005)
17. Schulzrinne, H., Rao, A., Lanphier, R.: Real Time Streaming Protocol (RTSP). RFC2326 (Apr 1998)
18. Singh, V., Ott, J., Curcio, I.: Rate Adaptation for Conversational 3G Video. In: Proc. of IEEE INFOCOM 2009 Workshop on Mobile Video Delivery. Rio de Janeiro, Brazil (Apr 2009)
19. The Insight Research Corporation: Streaming Media, IPTV, and Broadband Transport: Telecommunications Carriers and Entertainment Services 2009-2014
20. Wenger, S., Chandra, U., Westerlund, M., Burman, B.: Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF). RFC 5104 (Feb 2008)
21. Yao, J., Kanhere, S.S., Hassan, M.: An Empirical Study of Bandwidth Predictability in Mobile Computing. In: Proc. of ACM WinTech. SF, CA, USA (Sep 2008)