# A Performance Evaluation Framework for Video Streaming

Florian Metzger, Albert Rafetseder and Kurt Tutschku
{florian.metzger, albert.rafetseder kurt.tutschku}@univie.ac.at
Chair of Future Communication (endowed by A1 Telekom Austria)
Faculty of Computer Science
University of Vienna, Austria

*Abstract*—**In this paper, we present a generic framework for performance evaluation of video streams. We embrace the perspective of a streaming application, and model the playback buffer level at given network and playback conditions. That way, our model enables flexible, adaptable evaluation of a wide variety of streaming methods and protocols. We test our model on HTML5 and Flash video as served by the YouTube web site. We find universal achievable limits in any streaming process, and explore quality trade-offs every application has to decide on.**

## I. Introduction

Video streaming resonates well with web users, and streaming traffic makes up an ever-growing share of network traffic. At the same time, multiple streaming methods exists, resulting in a multitude of protocols, codecs, and their variety increases at an astonishing rate. Furthermore, the current boom in smart-phones creates an increasing plurality of access network technologies across which media are streamed.

This poses a problem to traditional analytical approaches like source-traffic modeling: Such models are complex to develop, and hard to adapt to new streaming mechanisms. Often, they deliberately omit details for reasons of analytical tractability, or only look at single layers of the network stack.

The method presented in this paper rather aims to evaluate performance by capturing generic behavioral patterns of streaming mechanisms, embracing the perspective of a streaming application. Specifically, we model the level of the playback buffer, and thus can subsume both network and playback behavior, while maintaining flexibility and adaptability with regards to the actual streaming server implementation, type of transmission network, protocol stack, and codec. Our model reports perceivable artefacts of buffer underruns, e.g. skips or stalls, which could then be fed into a QoE model to yield actual user QoE values.

The remainder of this paper is structured as follows. Section II reviews related work. Section III discusses issues arising from different (and multiple) parts of the network stack. In Section IV, we take a closer look at the buffering and playback behavior on both the theoretical and practical side of things. Our testbed approach is described in Section V and evaluated in Section VI, where we conduct an exemplary measurement and modeling campaign on the popular YouTube platform. Section VII wraps up the paper.

## II. Related Work

The technical fundamentals for video streaming have existed for a sufficiently long time so that there is a large body of existing work. We focus on TCP and especially HTTP streaming to which [1] and [2] give an introduction and overview the mechanics involved in streaming, e.g. flow control mechanisms in the video delivery.

The authors of [3] propose an analytical model for TCP-based video streaming, differentiating between live ("constrained") and prerecorded videos. In our approach, analytical tractability is not an issue as we perform actual measurements and decoding, so we are not limited to constant bit-rate video streams, constant packet sizes, or single playback strategies.

The importance of packet loss for an H.264 SD video stream is studied in [4]. Packet loss on the link is also investigated in our comparison of playback behaviors. However, in our first evaluation we analyze TCP as the transport protocol for streams. So, from the perspective of the application, packets are not lost, but delayed.

In [5], the authors present a quality-assessment model for video streaming services, with the quality features derived from the actual video. The model does not include the network behavior, but focuses on the codec performance instead.

A metric termed "application comfort" is calculated from YouTube videos in [6] to monitor live network conditions in realtime. While this approach is in effect similar to our evaluations, it is geared towards a very specific implementation of streaming, whereas we believe our methodology is more generic.

## III. Network Stack Layers and Streaming

In network layering models, it is often assumed that the layers are independent, or at least strongly decoupled, and present only narrow interfaces to each other. From a conceptual point of view, media streaming is a process governing the application layer. Thus, the application and its behavior might be thought to dominate the overall streaming process and associated quality. In this Section, we will show that this is not necessarily the case.

Figure 1 overviews the approximate time scales on which activities on different layers may take place, spanning a remarkable range of twelve orders of magnitude. Multiple layers might implement the same or similar functionality, e.g.

flow control in the application and on transport layer, resulting in nested control loops, which might be coupled due to the timing constraints.

## A. Network Layer

As seen in Figure 1, the time constants found in different network implementations range from nanoseconds (for Gigabit Ethernet) to seconds (for UMTS networks), depending on the technology used. This also influences the achievable round-trip time across such networks, which directly affects the performance of higher-layer protocols: IP, ICMP, UDP, TCP, and subsequently all application-layer protocols are subject to these timing constraints.

In the case of wireless networks, typical effects of wireless connectivity relating to physical phenomena, e.g. fading and interference, come into play. Flaky radio connectivity is a major source of packet loss and excessive delay. Certain cellular mobile technologies like UMTS and its evolutions implement loss concealment themselves, confounding IP's assumption of a host-to-network layer lacking guaranteed delivery. Other peculiarities of cellular mobile networks include a maximum transmission unit (MTU) opaque to IP, and delay variances as functions of packet sizes [7] and radio access technologies [8].

## B. Transport Layer

The two most widely used transport protocols are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). As is widely known, TCP implements a number of elaborate mechanisms to establish and tear down connections, deliver data to the application in sequential order, conceal loss on the network layer, adapt its bandwidth usage to the capabilities of the other endpoint (flow control) and the network (congestion control), and share bandwidth fairly through a distributed control algorithm.

UDP does not include any of mentioned mechanisms TCP has. This spurs the common misconception that UDP is the faster transport protocol. In fact, all packet types are subject to the same round-trip time, independent of the transport protocol used. Delays in the delivery of data to the upper layer occur in TCP when segments are considered lost in transmission (via timeouts or gaps in the range of acknowledged segments). TCP retransmits the lost segments, causing the round-trip time to spike temporarily. In the case of UDP, the application layer would need to handle packet loss.

As indicated in the previously, mobile cellular networks often conceal packet loss, which is used by TCP as an indication for network congestion. Rather than lost, packets are highly delayed, which can cause sub-optimal bandwidth usage. Mobile networks also show artifacts relating to port and network address translation, firewalling, and middleboxes interfering with TCP timeout on long-lived connections [9].

## C. Application Layer

There exists a diversity of streaming applications and associated application-layer protocols, each one supporting to a different degree certain types of streaming, and each having its own set of requirements, depending on the content type (pregenerated or live), the codec and its bit-rate, and playback control and quality feedback.

One classification for streaming protocols might be their body of standardization: There are many proprietary protocols with undisclosed or legally restricted standards documents, e.g. RTMP (Real Time Messaging Protocol) and RTMPCS (RTMP Chunk Stream), MMS (Microsoft Media Server), and WMSP (Windows Media HTTP Streaming Protocol). Other protocols and protocol families are standardized by open bodies such as the Internet Engineering Task Force (IETF). In our work, we focus on these "open" protocols.

In the latter category are two well-known protocol families for media streaming, Real-time Transport Protocol (RTP) and Hypertext Transfer Protocol (HTTP). RTP sees most of its use in walled garden IPTV services. As RTP is designed for media transport, a companion protocol suite consisting of RTCP for control information exchange, RTSP for streaming control, and SDP for session management is often used. RTP is mostly transported using UDP.

HTTP is the single most common application layer protocol on the Internet, owing its popularity to the ubiquity of web browsers. In contrast to RTP, HTTP was not designed for specific payload types apart from HTML. The actual streaming protocol behavior is defined by the application, not by the protocol. Every service is thus free to define its own distinct protocol behavior. For HTTP, many de-facto variants for streaming exist, but many if not most are not formally standardized. It is this fact that makes evaluating HTTP streaming protocols against each other especially and we attempt to solve with our framework approach.

The time scale on which streaming applications buffer content lies in the range of seconds. This is a necessity in a best-effort network, as the available network bitrate might drop unexpectedly and could drain a shallower buffer quickly. The application behavior represents a trade-off between different types of perceivable artifacts – initial startup delay, stalls, (partial) media skips (e.g. continuous audio, but skipping video), and quality adaption. The next Section will specifically look at these issues.

## IV. Application Behavior

To play back a media stream, an application needs to maintain a media buffer of sufficient size to at least gather enough data to reconstruct one single atomic unit of playback, typically a video frame.

The current buffer level at time $t$ is given by the amount of data received from the network so far, diminished by the amount of data already played back. The actual media format used in the stream determines the progress of the playback process, whereas the network conditions and application download strategies yield the overall progress of data reception:

$$buffer(t) = \sum_{0}^{t} data_{\text{received}} - \sum_{0}^{t} data_{\text{played}}$$

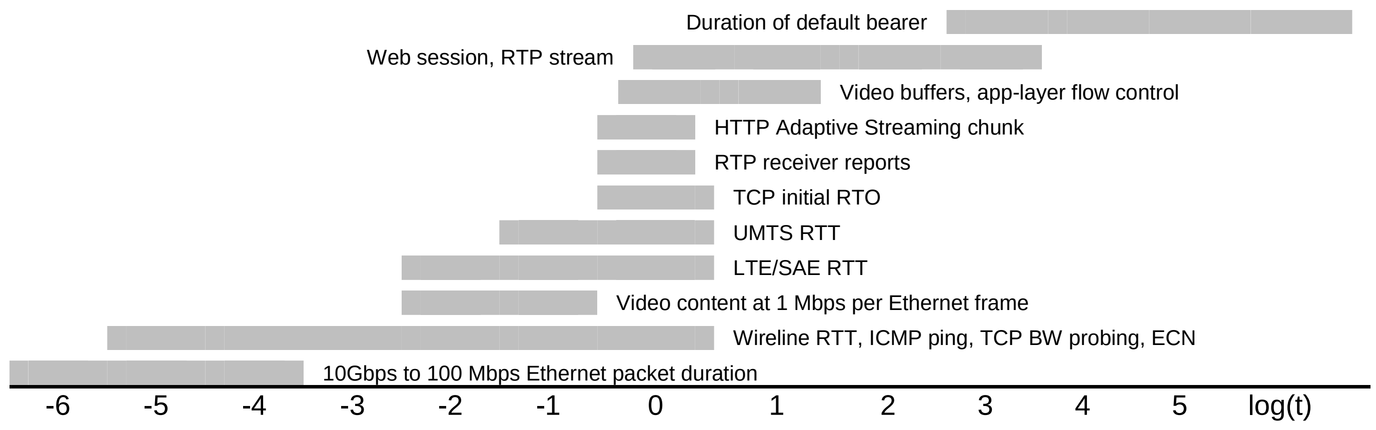


Fig. 1. Relevant time scales in the layers of the stack

The playback buffer level governs how well the player can conduct the playback. If the buffer reaches zero size the playback process stops and stalling occurs. Then a decision is required when to restart the playback process again after a stall, and if the stream should skip forward to a more current playback position. This process is the core part of the playback model for an HTTP streaming service. The model could also be extended to accommodate adaptive streaming mechanisms that use the buffer level as feedback information to influence the download strategy of the player, e.g. send a receiver report to request a lower bit rate stream in the case of RTP.

Playback models need to define the behavior at the occurrence of one of two conditions during the playback process. These are the initial playout delay (the point in time when to start draining the playback buffer), and the buffer fill level at which to start playing again if the buffer had intermittently emptied and the video had to be stopped.

These decisions yield a stalling period distribution for a streamed video. The frequency and the duration of stalls directly relate to the decision function of the playback model. The more frequent the stalls are, the shorter they will be; if the function produces longer stalls, they will be less frequent.

There can be other parameter spaces governed in the model. For example, in a live streaming scenario a user could prefer to always stay at the most current stream position. To enable this, a player would skip older parts of the video. If the user prefers to consume the entire stream, the player would show the video without skipping parts, but pause intermittently.

Another user parameter is the quality level of the video for adaptive streaming. This trades off between maintaining a certain quality level and putting up with increased waiting times, and dropping the quality to a level sustainable at the current transmission rate.

The next subsections present four stalling playback models, ranging from theoretical models that represent boundaries to the values possible in stalling characteristics, to the Firefox and the Flash model, which represent actual player behaviors that can be seen “in the wild”.

## A. *Simple Playback Stalling Model*

The behavior can be summarized as “Whenever anything can be played from the buffer, do so”. This means that, if the player is currently stalling and a complete frame becomes available in the buffer, playback will immediately restart and the frame will be shown even if this means stopping playback after that frame again. This results in the lowest required buffer space. Moreover, it gives an upper limit for the number of stalls occurring.[1]

## B. *Initial Playback Delay Model*

The model will simply delay the initial start of the video until it can be played completely without any buffer underruns occurring. The only stall occurring is the initial waiting period until the video starts. The time spent waiting will also be minimal. The downside of this model is its reliance on knowledge of future network conditions, making it purely theoretical. Actual streaming players need to accurately predict the transmission process, e.g. through moving averages.

The stalling and initial delay models define the maximum achievable upper and lower limits for the stalling parameter space for all possible real models.

## C. *Firefox HTML5 Model*

The algorithm used in the Firefox 4 browser is an approximate realization of the HTML5 video standard [10] which suggests starting the playback only when it can be ensured that the video can be played without interruption similar to the initial playback delay model.

The algorithm shown in Fig. 2 and its variables in Table I demonstrate the exact behavior of this strategy. Firefox 4 uses moving averages to estimate the development of the transmission rate. It does not differentiate between intermittent and initial conditions. As the approach is similar in concept to the initial playback delay model, it sports very few stalling events due to conservatively chosen (i.e., long) buffering times.

[1]As a video frame is atomic, no other model could possibly stop the playback more often.

```
if $s_{MA} > v_{MA}$ then
    $c \leftarrow (b_b = 20s \lor b_T = 20s)$
else
    $c \leftarrow (b_b = 30s \lor b_T = 30s)$
end if
```

Fig. 2. Firefox playback (re-)start decision algorithm.

TABLE I
VARIABLES INVOLVED IN BUFFERING DECISIONS.

| Variable | Explanation |
|---|---|
| $s_{MA}$ | Moving average of the transmission speed. |
| $v_{MA}$ | Moving average of the video bitrate. |
| $c$ | Condition upon which to start/resume playback. |
| $b_b$ | Amount of video data the buffer contains. |
| $b_T$ | Amount of time spent in non-playing buffering state. |

### *D. YouTube Flash Player Model*

This model is facilitated by the Flash Player used by the YouTube website. It will initially start the playback after it has buffered two seconds of video data. If a stall occurs it will restart playing after five seconds of video are in the buffer. The Flash Model assumes sufficient network conditions in the beginning, requiring only a short initial playback delay to pre-fill the playback buffer. If, however, stalling occurs, then it will buffer longer to keep the stalling frequency down.

## V. TESTBED ARCHITECTURE

Testing and comparing new protocols is a complex and time-consuming undertaking if traditional evaluation approaches such as analytical source-traffic models are employed. For today's fast-paced development of streaming protocols and strategies, faster evaluation methods are needed. A testing environment must be simple and sufficiently flexible to allow testing of current and adaptation to future protocols, services, network structures, and associated parameter spaces. At the same time, it must be able to answer specific questions about protocols, applications, and network setups. Finally, the results yielded from analyzing and evaluating a streaming service should be user-oriented, i.e. provide a foundation of data that can be fed into the calculation of a Quality of Experience (QoE) model. We believe our proposed testbed meets these requirements.

Figure 3 depicts the evaluation testbed. Conceptually, it replicates the actual steps a user would perform to consume a media stream on a playback device. Through appropriate configuration different scenarios can be modeled, e.g. network conditions, behavior and specifics of the user device.

In the first pass of the process, the stream data is transmitted from a server to the client. The server could either be an actual streaming service on the Internet, or a local streaming server implementation. The traffic is directed through a network emulation node capable of altering the network QoS parameters, i.e. latency, jitter, and packet loss. The parameters could also be set according to stochastic models derived from actual network traces.

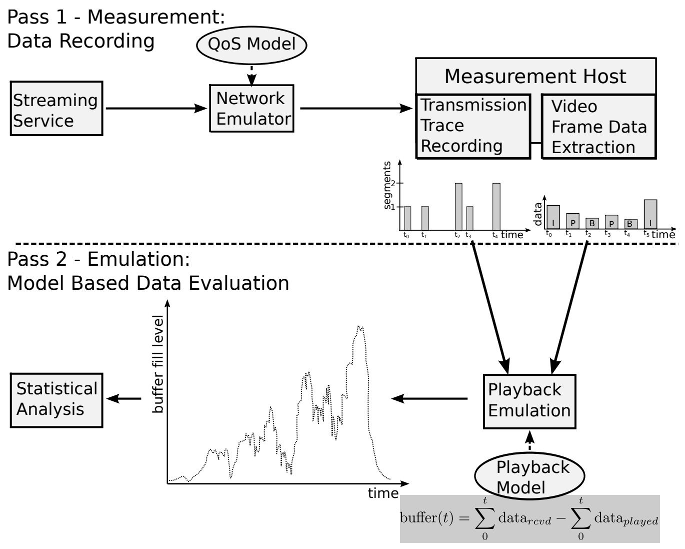


Fig. 3. Testbed Schematic.

The measurement host downloads and records a network trace of the video stream. For HTTP streaming, it issues a single HTTP `GET` request on the video file, and then maintains the TCP connection until the file has fully arrived. Depending on the level of detail of the traces, they could further be used to scrutinize other layers of the connection, e.g. the dynamics of TCP receive window size. The packet trace is then decoded using the open source `ffmpeg` suite, yielding another set of traces consisting of video frame sizes and playout timestamps.

In the second pass, these two traces are then used to feed the playback models described before. The playback emulation process combines the transmission and video frame traces to calculate the playback buffer fill level for every point in time during the playback. It then generates statistics about user-perceivable artifacts such as playback stalls that would occur during the playback. These statistics can then be compared to the results of other models and network QoS. It is worth noting, that the two steps are independent of each other, meaning that all playback strategies can be applied to the same network trace allowing for direct comparison.

Currently not modeled in this framework is any kind of user interaction during playback, i.e. users that skip, pause or stop playback. However, statistics on any such behavior could be easily integrated into the measurement setup.

## VI. EVALUATION

The results presented here show some of the capabilities in comparing play network conditions based on playback models. We compare how the four playback models introduced previously fare against each other in a measurement series featuring emulated transmission latency and packet loss. At the same time, we acknowledge that other specific questions are not touched in this first set of experiments, e.g. the inclusion of a mobile network or handset, or RTP-based streams.

The video used in our measurements was streamed from the YouTube web site. This provides a realistic base for all the

TABLE II
TEST VIDEO PARAMETERS

| | |
|---|---|
| Video Duration | 01:32.536 minutes |
| Size | 9.61 MiB |
| Framerate | 23.976/s |
| Average Video Bitrate | 871 Kbps |
| Codec | AVC+AAC |

experiments. Note that YouTube also employs its own form of application layer flow control in addition to TCP's [11]. Details on the streamed video are available in Table II.

For the loss experiment series, the network emulator was configured to drop a certain percentage of packets based on a normal distribution on both the uplink and the downlink direction. There was no loss correlation involved, the existence of which one would expect, e.g., in wireless networks. One streaming run for every two percentage points of additional loss up to 14% was conducted.

In the latency series, the emulator delayed the delivery of every packet for a symmetrically amongst up- and downlink distributed time. One experiment was conducted for every 100ms of additional delay, up to a total of 5000ms.

After the traces were recorded, every playback model was applied to all runs. For every model, two data points were computed. First, the total stalling time was calculated. This is the time the player keeps buffering and not playing the video, including the initial start delay. To attain results comparable to the other measurements, the stalling time is calculated relative to the total video length instead of an absolute value. The second resulting value is the number of times the video stops playing including the initial delay, i.e. the stalling frequency. Both of them are an indicator how well the playback mechanism can cope with the currently emulated network setup. They could for example be used to either check if a modification to a network has a noticeable impact on streaming experience, or to find an optimized player behavior for a given network.

All models will generally work very similar in good network conditions. If sufficient bandwidth is available, they will play videos with almost no delay and intermediate buffering. If, however, the achievable TCP goodput is close to the average video bitrate, the buffer may be strained by short deviations from the average rates.

High latency can also trigger TCP timeouts and retransmissions, and in turn decrease the congestion window, further impacting the goodput. The latency measurements are depicted in Figure 4. The stalling time increases with the additional latency. The Initial Start Delay model provides the best possible result in terms of pure stalling time. On the other hand, Figure 4b shows the Stalling model provides always the worst result for the number of stalls. Any other model will lie beyond that line. The Flash and HTML5 models both run in just a few buffering events which however tend to increase in length with rising latency. Attributed to the simple and optimistic assumption of the Flash model, stalling time is usually lower than with HTML5, at the cost of slightly more buffering events.

TCP goodput is strongly affected by packet loss. Lost packets result in duplicate acknowledgements, retransmissions, and a decreased congestion window. The connection could stall on missing old segments without which the playback cannot proceed. Figure 5 shows some exemplary measurements for a loss scenario. While additional packet loss of up to four percent seem to have no noticeable impact on streaming quality, the total stalling time suffers a large increase for any model as seen in Fig. 5a. Figure 5b shows the extremity of the Stalling model compared to other models reaching a number orders of magnitude larger than any other model.

Through these to exemplary experiments, we tried to show that network QoS parameters have a direct measurable impact on the application layer, namely on HTTP streaming quality. While the models scale rather well with latency, any HTTP streaming is almost impossible with high packet loss values. Comparing the presented playback models, we conclude that every model represents a trade-off between several parameters, e.g. as measured here, the number and length of stalls. With the knowledge gained from the experiments, playback models could be tailor-made to best suit certain conditions and user requirements. We further conclude, that the evaluated practical playback strategies are rather tailored for fixed networks, not being able to cope very well with high latency or loss. Additionally, both seem to favor keeping the stalling frequency low instead of keeping a short total stalling duration. However, there is still room for optimization.

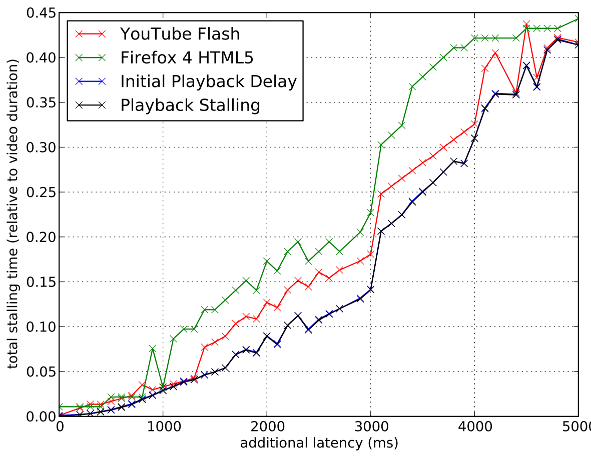
## VII. CONCLUSION

Because of rapid developments in the field of video streaming, full-scale measurement campaigns and analytical modeling might prove too time consuming to test every new protocol. The streaming model framework presented here offers methodologies to quickly evaluate new streaming mechanisms under the influence of network QoS as it decouples the network trace recording and the playback model calculation phase.

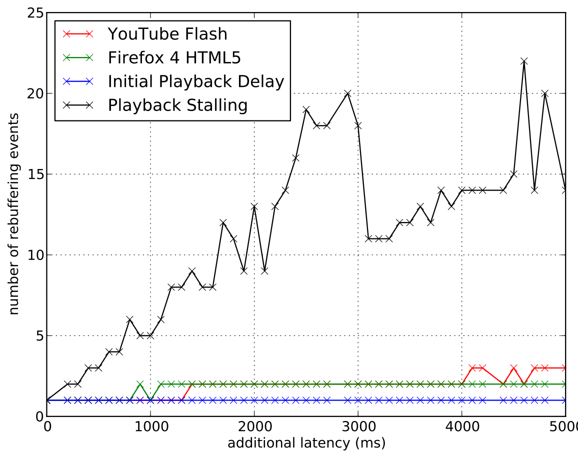
We detailed possible influences of the different network layers on video streaming, This inspired the creation of a generic model, incorporating universal notions on data transport, flow control, and buffering, striving to cover most possible streaming methods. Using this model, we explored the theoretical quality limits for streaming such as limits for the maximum stalling duration, and the user trade-offs incurred by making specific choices on how to treat conditions related to streaming processes. In our evaluation of the model on YouTube we observed the influence of network QoS on playback quality and found that current playback strategies do well under normal circumstances but tend to break down in unexpected scenarios, leaving room for further optimization.

## ACKNOWLEDGMENT

The authors thank Jochen Eisl of NSN Munich for the fruitful discussion, and Ákos Lukovics, Bernhard Gruber, and Stephan Seebacher for assistance in the development and maintenance of the testbed mechanisms.

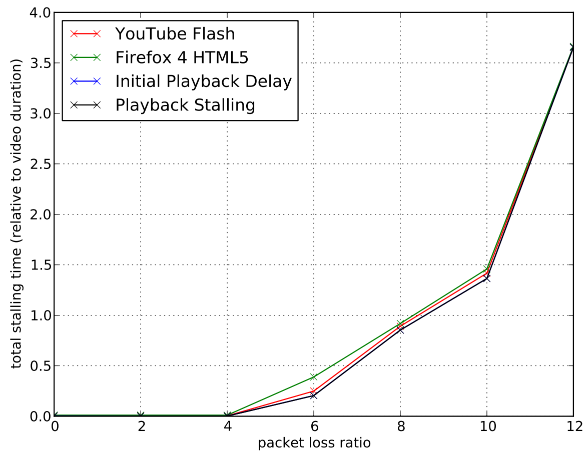


(a) Total stalling time

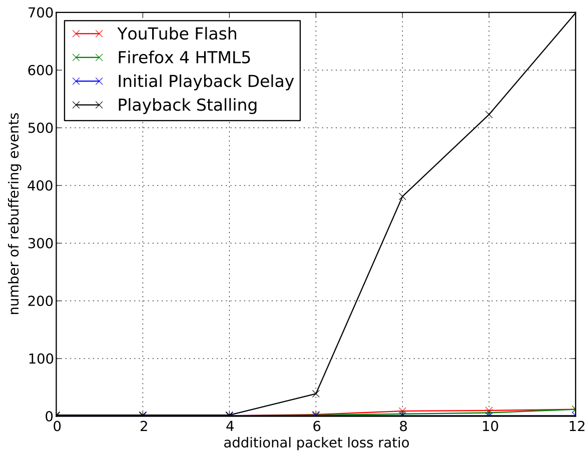


(b) Number of stalls

Fig. 4. Playback model observations on additional latency



(a) Total stalling time



(b) Number of stalls

Fig. 5. Playback model observations on additional packet loss.

## References

[1] A. Begen, T. Akgul, and M. Baugher, "Watching Video over the Web: Part 1: Streaming Protocols," *IEEE Internet Computing*, vol. 15, no. 2, pp. 54–63, Mar. 2011. [Online]. Available: http://dx.doi.org/10.1109/MIC.2010.155

[2] K. Ma, R. Bartos, S. Bhatia, and R. Nair, "Mobile video delivery with http," *IEEE Communications Magazine*, vol. 49, no. 4, pp. 166–175, Apr. 2011.

[3] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "A Model for TCP-based Video Streaming," University of Massachusetts Technical Report, 2003. [Online]. Available: http://lass.cs.umass.edu/papers/pdf/TR03-TCP.pdf

[4] Y.-L. Chang, T.-L. Lin, and P. C. Cosman, "Network-based ip packet loss importance model for h.264 sd videos," in *Packet Video Workshop 2010*, 2010.

[5] T. Kawano, K. Yamagishi, K. Watanabe, and J. Okamoto, "No reference video-quality-assessment model for video streaming services," in *Packet Video Workshop 2010*, 2010.

[6] B. Staehle, M. Hirth, R. Pries, F. Wamser, and D. Staehle, "YoMo: A YouTube Application Comfort Monitoring Tool," in *New Dimensions in the Assessment and Support of Quality of Experience for Multimedia Applications*, Tampere, Finland, Jun. 2010.

[7] P. Arlos and M. Fiedler, "Influence of the packet size on the one-way delay in 3g networks," in *Passive and Active Measurement, 11th International Conference, PAM 2010*, vol. 6032, 2010, pp. 61–70.

[8] M. Laner, P. Svoboda, E. Hasenleithner, and M. Rupp, "Dissecting 3g uplink delay by measuring in an operational hspa network," in *Passive and Active Measurement*. Springer, 2011, pp. 52–61.

[9] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang, "An untold story of middleboxes in cellular networks," in *Proceedings of the ACM SIGCOMM 2011 conference on SIGCOMM*. ACM, 2011, pp. 374–385.

[10] "HTML5: A vocabulary and associated APIs for HTML and XHTML," W3C, 2011. [Online]. Available: http://dev.w3.org/html5/spec/Overview.html

[11] S. Alcock and R. Nelson, "Application flow control in youtube video streams," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 24–30, 2011.