A Comparison of Quality Scheduling in Commercial Adaptive HTTP Streaming Solutions on a 3G Network

Haakon Riiser[1], Håkon S. Bergsaker[1], Paul Vigmostad[1]
Pål Halvorsen[2,3], Carsten Griwodz[2,3]

[1]Netview Technology AS, Norway
[2]Simula Research Laboratory, Norway
[3]Department of Informatics, University of Oslo, Norway

ABSTRACT

There are many available commercial streaming solutions that perform quality adaption. An important issue with respect to users' perceived quality is how the system schedules the quality levels to match the available network resources. In this study, we compare several adaptive media players on the market to see how they perform in challenging streaming scenarios on a mobile 3G network. Bandwidth data collected in real-world field trials is used in all tests. We investigate how the media players respond to fluctuating bandwidth and outages, and how this affects the quality levels used, the bandwidth utilization, and the number and duration of buffer underruns. We found significant differences in performance and optimization goals between the different players' schedulers. We conclude that the quality scheduler is an important factor in providing a satisfying quality of experience when using an adaptive media player.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Video*; C.4 [**Computer Systems Organization**]: Performance of Systems

General Terms

Experimentation, Measurement, Performance

Keywords

adaptive streaming, bitrate adaption, quality scheduling, mobile internet, 3G, wireless, fluctuating bandwidth, Microsoft Smooth, Apple HLS, Adobe Dynamic, Netview

1. INTRODUCTION

Adaptive streaming over HTTP [13, 12] is rapidly becoming popular among commercial vendors of streaming technology. It can be implemented as a combination of simple servers and intelligent clients that make adaptation decisions based on local observations. The technology's versatility and relative simplicity has made it successful on everything from high-speed fixed networks with HD-capable receivers to small handheld devices on mobile wireless networks. As a strictly pull-based approach, adaptive HTTP streaming is quite different from early video streaming techniques that relied on server-side decisions and multicast. Pull-based streaming has become viable because the development of the Internet infrastructue has made it possible to take cheap server capacity and backbone network capacity for granted. Under these conditions, providing good quality of experience on a fixed high-speed network is not very difficult any more, but it is considerably more challenging when the client's access network is a mobile wireless network with severe and frequent bandwidth fluctuations and outages. Such network conditions result in recurring buffer underruns and frequent quality switches, both harmful to the viewer's quality of experience.

Under such circumstances, proper configuration of a media player's quality scheduler is both challenging and important. The overall aim is to provide the best possible viewing experience. Important goals include:

1. Avoid buffer underruns, as they cause interruptions in video playback.
2. Avoid rapid oscillations in quality, as this negatively affects perceived quality [8, 15].
3. Utilize as much of the potential bandwidth as possible to give the viewer a higher average video quality.

While actively using several commercial video systems in our other projects, we have experienced large differences in performance with regard to the goals stated above. This prompted us to conduct a more rigorous performance comparison, where we investigate how the players perform and adapt in challenging mobile streaming scenarios using typical bus, ferry, metro and tram commute routes in Oslo, Norway. In particular, we want to know how the different systems perform in terms of robustness against underruns, stability in quality, and bandwidth/quality utilization.

To the best of our knowledge, one similar study has been performed before [6], but they have used synthetic bandwidth data which looked nothing like the strongly fluctuating bandwidth curves we have observed in real mobile wireless networks [11, 10].

MoVid'12, February 24, 2012, Chapel Hill, North Carolina, USA.

Because several of the compared systems are closed source, we cannot know the exact algorithms, but we can still report their behavior. In earlier related work [6], this has been done by looking at the consumed bandwidth and buffer sizes, which is not necessarily closely related to video quality and user perception. In our experimental testbed, we have support for Adobe, Apple, Microsoft and Netview's media players from a single video representation. This means that we encode the content once for each quality level, and use that exact stream on all media players to be compared. Consequently, we can actually compare the resulting video playback quality.

From our experimental results using Telenor's 3G mobile network, we observe large differences in the rate adaption policies. Apple and Adobe's players represent two opposites in that Apple focuses more on stable quality at the expense of high average quality, whereas Adobe does the opposite. Microsoft's Smooth Streaming [14] and Netview's Media Client [4] try to achieve good stability as well as high bandwidth/quality utilization, but the Netview solution seems to offer better protection against buffer underruns and better bandwidth utilization.

2. EXPERIMENTS

2.1 Tested Systems and Video Formats

Several competing HTTP streaming systems exist, and it is out of the scope to compare all. The Motion Pictures Experts Group is working on standard formats under the label DASH [13], which will standardize formats, but not the way in which clients adapt quality. We have therefore selected four different, representative media players to investigate how they make adaptation decisions:

- Adobe's Strobe Media Playback (v1.6.328 on Flash 10.1 on a Windows 7 PC) using its native HTTP Dynamic Streaming format [2]. Note that this player is also known as OSMF, short for the Open Source Media Framework on which it is built.
- Apple's iPad player (iOS v4.3.3 (8J2)) using its native HTTP Live Streaming (HLS) format [9].
- Microsoft's Silverlight (v4.0.60531.0 on a Windows 7 PC) using its native Smooth Streaming format [14].
- Netview's Media Client (v2011-10-10 on a Linux PC) using Apple's HLS format. Netview's client offers multiple schedulers; the scheduler studied in this paper is designed for streaming in 3G networks.

We do know several other pull-based approaches like Netflix (using Microsoft Smooth Streaming) and Move Networks. These systems are closed, meaning that we could not support them in our testbed. Netflix was however tested more generally in [6] – its aggressive adaption mechanism targeted for fixed networks would place it between Microsoft and Adobe in terms of performance. Thus, in the context of this paper, we test the four listed systems as representatives for existing commercial systems.

We verified that each system could play all quality levels with no issues. For the Adobe, Apple and Microsoft players, we used their native formats. Netview's media client supports all of the above formats and we decided to use Apple's HLS format – its overhead was never more than 12 % higher than the other formats, and by choosing the HLS format for the Netview client, we could show that Apple's quality adaption results are due to their implementation, not the format itself.

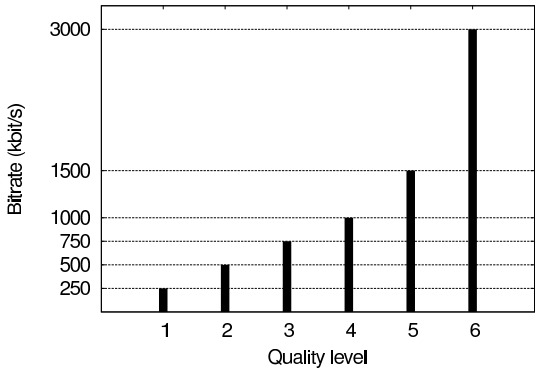


Figure 1: Relation of video quality levels and their average bandwidth consumption.

On the server side, we used CodeShop's Unified Streaming Platform v1.4.25 [7] (integrated into the Apache v2.2.14 web server [3]) to support all streaming formats without requiring different video encodings; i.e., the only differences between the tests of the different systems were the protocol and media container formats. Differences in bitrates between formats (due to different container overhead) was always less than 12 %, meaning that the results were not significantly affected by the chosen streaming format[1]. The web server ran on a dedicated Linux box with 2 GB RAM and an Athlon 64 3200+ CPU. Both the receiver and the web server were on the same 100BASE-TX Ethernet, using our custom-made throttling module to reproduce the behavior of a real 3G network (see section 2.3).

2.2 Video Stream Configuration

Ensuring fairness in the video stream means that the video segmentation has to be as equal as possible (duration and bits per segment) for all the different media formats used. We encoded video (a European football match) with a fixed segment duration of two seconds and six quality levels. The bitrates used for the six quality levels are shown in figure 1. Bitrates at the lower end were chosen based on Akamai's recommendations [1] for small handheld devices on mobile networks. Bitrates for the higher qualities were chosen based on subjective testing, trying to achieve a linear scale in perceived quality. The reason for the larger leaps in bitrate between the highest three levels is diminishing returns in quality – it was necessary to double the bitrate to make level 6 visibly better than level 5.

2.3 Realistic and Equal Network Conditions

To perform a fair and realistic comparison of the different streaming systems, we need equal network conditions for

[1]We could have encoded an extra representation for HLS to compensate for this overhead, but sacrificing quality to achieve the same overall bitrate means that the quality levels would not be identical in all tests. Because the relative bitrate difference was less than 12 % even for the lowest quality level (where the relative overhead is highest), we deemed it negligible and decided instead to use the same representation for all tests.

each test which match the observed bandwidths in real mobile 3G networks. We have previously [10, 11] performed a large number of real-world experiments, and while we found the bandwidth (as a function of geographical location) to be fairly deterministic, this study requires identical results on each run to achieve a fair comparison.

For this reason, we created an advanced throttling module for our Apache web server. This module takes as input a bandwidth log (from a real-world test) that contains a single kbit/s number for every second of the session. After loading the bandwidth log, the first HTTP request starts the session. At time t after the session starts, the web server's maximum throughput for the next second will be $B(t)$, where $B(t)$ is the bandwidth at time t in the log that was used as input to the throttling module. In addition to bandwidth throttling, our Apache module also adds a small one-way delay of 40 ms to simulate the average round-trip latency as experienced and measured in our real wireless 3G network.

This approach means that each media player can get exactly the same conditions, ensuring both fairness and reproducibility in our experiments, while at the same time being nearly as realistic as a field trial.

We selected four representative bandwidth logs from our database of measurements[2]. Each log represents a typical run in its respective environment. The four streaming environments are popular commute routes in Oslo (Norway) using ferry, metro, bus and tram. The travel routes with the corresponding bandwidths are shown in figure 2. We can see that they represent different challenges with respect to both achieved rates and outages.

2.4 Logging a Segment's Video Quality

While streaming, we used `tcpdump` [5] on the server to log every packet transmitted to the receiver, so that we could measure the actual achieved throughput (which might be less than the bandwidth cap set by the throttling module, e.g., if the client consumes the available network resources inefficiently). The packet dump contains every HTTP GET request for every downloaded segment, so it also contains the information we need to plot the quality level as a function of playout time. However, because we are testing proprietary media players where we do not know the state of their internal buffers, buffer underruns were logged manually by actually watching the video in every test, and registering the times when the video stopped and resumed.

3. RESULTS AND ANALYSIS

In this section, we present our results. In particular, we look at 1) the achieved segment quality along the routes according to time, 2) the amount of video data presented in each quality level including buffer underruns, and 3) the length of each playout interval at a given quality to give an indication of the quality switching pattern. These properties are plotted in figures 3, 4 and 5, respectively, for all four routes.

3.1 Adobe

Comparing Adobe's quality level plots in figure 3 with the bandwidth plots, one can clearly see that their shapes

[2]These streaming environments were used extensively in our previous paper on predictive quality scheduling algorithms [10].

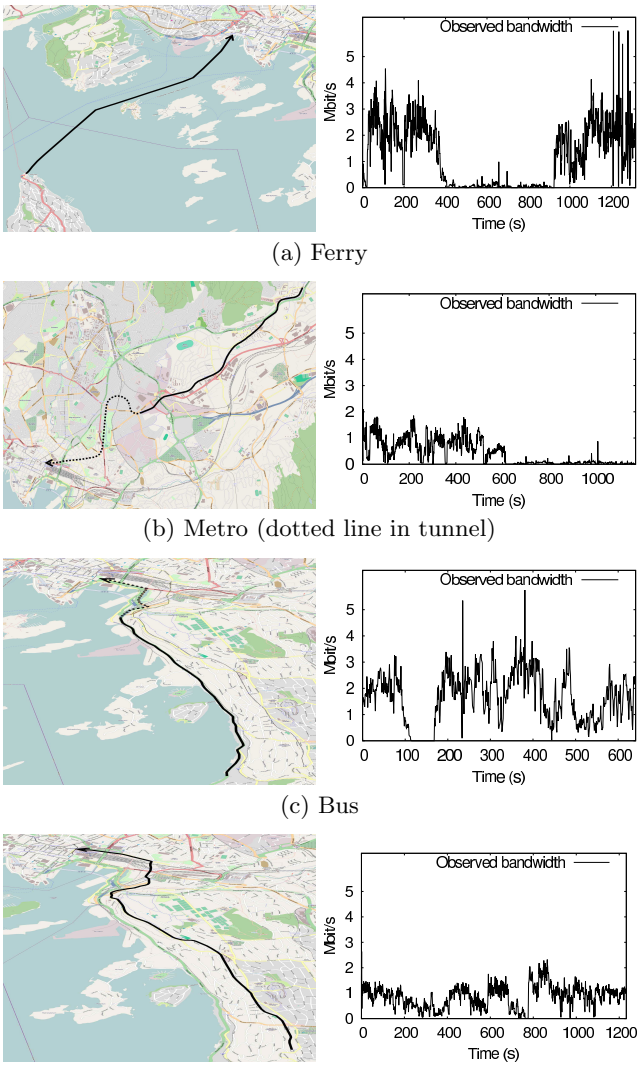
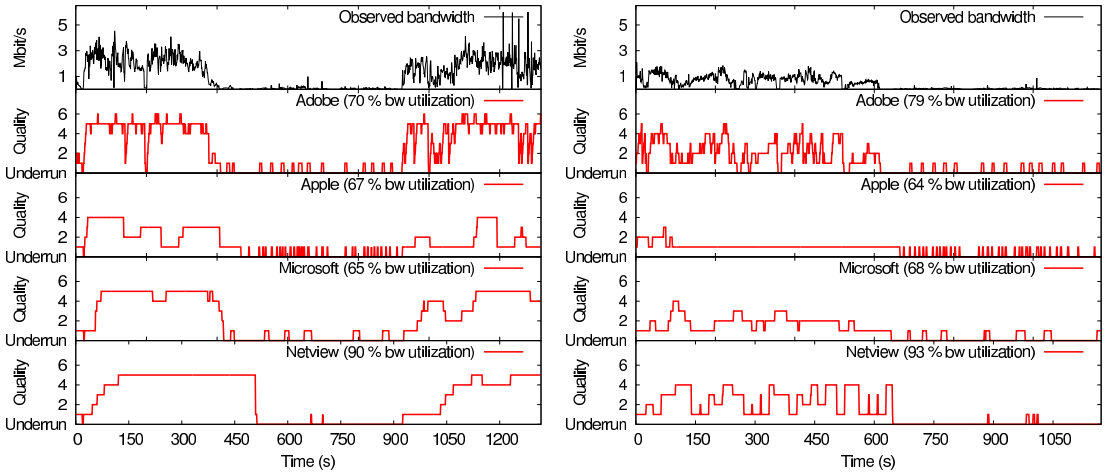


Figure 2: Test environments and the observed bandwidths used in our experiments.

are almost identical. From this, we conclude that the quality scheduler in Adobe's Strobe player bases its decisions almost exclusively on the most recent bandwidth numbers. The next segment to be downloaded is the one whose bitrate is closest to the current bandwidth, with no considerations to stability or safety margins. As a result, the users' quality of experience suffers due to buffer underruns and too frequent oscillations in quality (figure 5). Despite minimal use of buffering, the scheduler achieves decent bandwidth utilization, mainly because high bitrate segments were downloaded quite often (even when unsafe to do so), meaning more bytes per download request, and thus, less wasted bandwidth.

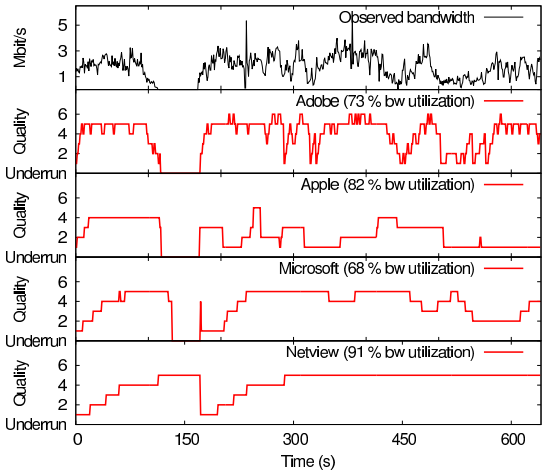
3.2 Apple

The quality scheduler in Apple's iPad player stands out from the others by being more careful about increasing quality. Its frequent use of low quality segments produces stable quality (figure 5) with long intervals in the same quality. The

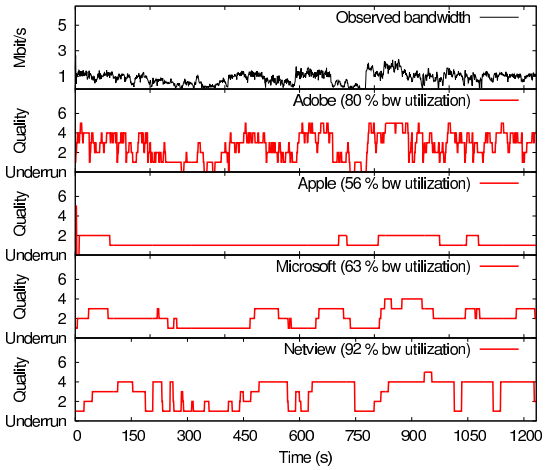


(a) Ferry

(b) Metro



(c) Bus



(d) Tram

Figure 3: Comparison of quality scheduling for the four commute routes. Quality is plotted at playout time. This implies that effects of network outages are reflected with a delay when buffers run dry.

tendency to pick low quality levels is clearly seen in figure 4. Despite having lower quality on average than the other players, Apple's bandwidth utilization is higher than one would expect. The reason for this is that the bandwidth utilization number does not take into account whether the downloaded video data was actually used. Unlike the other players, the iPad player often re-downloads segments – sometimes the same segment is downloaded two or more times, usually in different qualities, but not always. This means that some of the downloaded video data is never used, and bandwidth is wasted.

3.3 Microsoft

Microsoft's Silverlight player achieves a nice balance between bandwidth utilization and stability. Compared to Apple's player, it uses more of the available qualities (figure 4), while still achieving a fairly stable viewing experience (figure 5), setting it apart from Adobe's player. We observe that Microsoft fills its buffers when the bandwidth is high, progressively increasing quality, instead of instantly jumping to the quality level whose bitrate is closest to the current bandwidth. However, we observe that the quality increases just as quickly when the bandwidth is poor, indicating that filling its buffers is not a priority.

Microsoft's biggest problem is that the buffer tends to be very small, especially considering that it is designed for PCs with plenty of memory. This limitation makes it more vulnerable to buffer underruns than Netview's player (figure 3(c)). Average quality is better than Apple's, but not quite as high as Adobe's numbers. Microsoft's player would have achieved better results with a better buffering strategy, as it often wastes bandwidth by idling, even when its buffers are almost empty. On the other hand, the bandwidth utilization is quite stable at about 65 % and the Silverlight media player is intended to run as a browser plugin, so it might be a design goal to leave some bandwidth available for other services.

3.4 Netview

To efficiently use available resources and provide a good viewing experience, Netview's reactive scheduler uses several techniques, described in [10] with the only modifica-

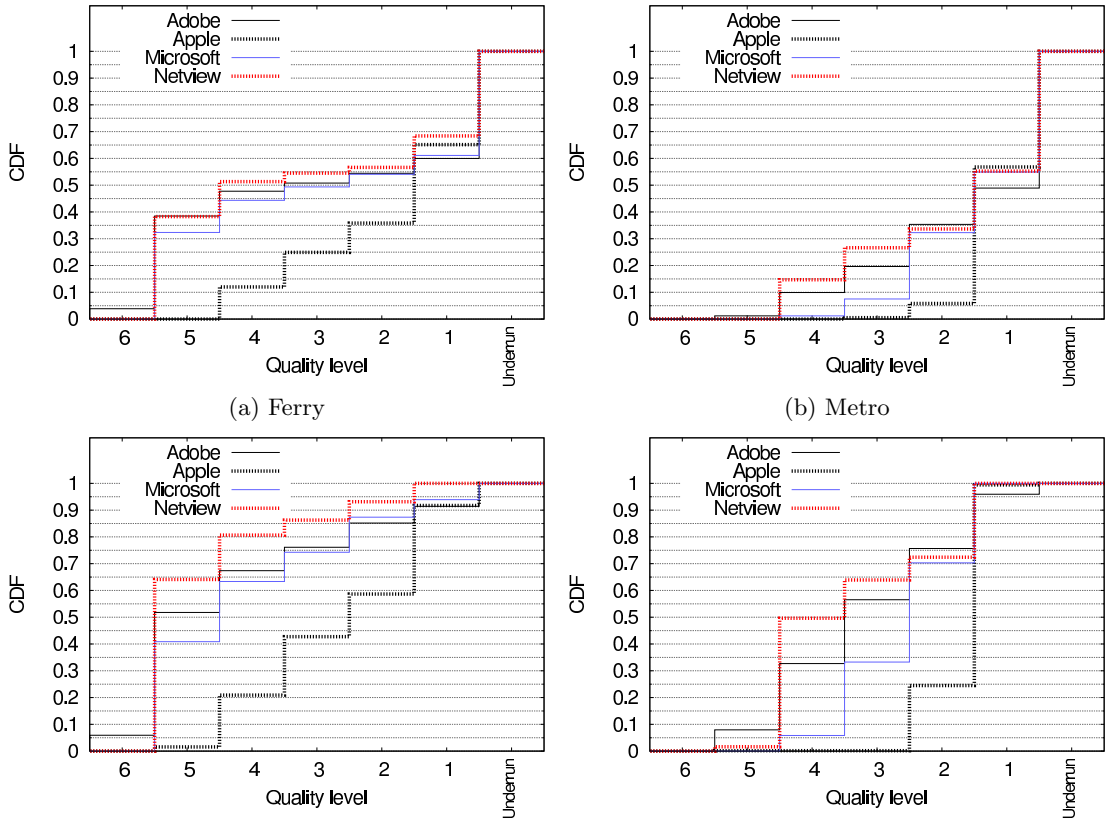


Figure 4: Cumulative distribution function of quality. The height of each step represents the total amount of time that a video is played out at a given quality level (and the last step shows the cumulative duration of playout interruptions). Higher towards top-left corner is generally better, but the frequency of quality changes, which is visible in figure 3 and which can be visually disruptive, is not represented.

tion that bandwidth capping is now only enabled when the buffer has less than 30 seconds of video. The results show that the Netview scheduler is most successful in scenarios with periodically high bandwidth, such as figures 3(a) and 3(c)). Like the scheduler in Apple's iPad, it is designed for the needs of a product utilizing wireless networks. Thus, Netview's player is conservative when the buffer is small, increasing quality slower than the other players, and using extra bandwidth to increase its buffer. When the buffer is sufficiently large, it prioritizes stability over buffer fullness, making the quality very stable (figures 5(a) and 5(c)). In scenarios with poor bandwidth, it sacrifices stability (figures 5(b) and 5(d)) to achieve more robustness against buffer underruns. However, unlike Adobe's player, it always takes precautions to avoid switching too often, avoiding the rapid fluctuations that are known to annoy users [8, 15]. Similar to Microsoft's player, the Netview player usually achieves a progressive increase and decrease in quality, i.e., avoiding large jumps (except when performing bandwidth capping to preserve the buffer fullness). Note that Netview's player is more aggressive when dealing with the lower quality levels, as there are diminishing returns in quality as bitrates increase. The bandwidth utilization of Netview's scheduler is good in all tested scenarios, both due to buffer strategy and by frequently downloading high bitrate video segments.

4. CONCLUSION

We found large performance differences between the various systems, even though none of them have advantages over each other in terms of what scheduling information is available. Apple and Adobe's players represent two opposites in that Apple sacrifices high average quality for stable quality, whereas Adobe does the opposite. Microsoft's Silverlight player falls in between, but without compromising too much on either parameter. Netview's scheduler is similar to Microsoft's, but offers better protection against buffer underruns and better bandwidth utilization. From our experiments, we conclude that the quality scheduler has a large impact on the quality of experience in adaptive HTTP solutions and that several products on the market have a definitive potential for improvement when streaming in mobile networks.

Acknowledgements

This work has been performed in the context of the *HyStream* project (project number 176847) and the *iAD* centre for Research-based Innovation (project number 174867) – both funded by Norwegian Research Council.

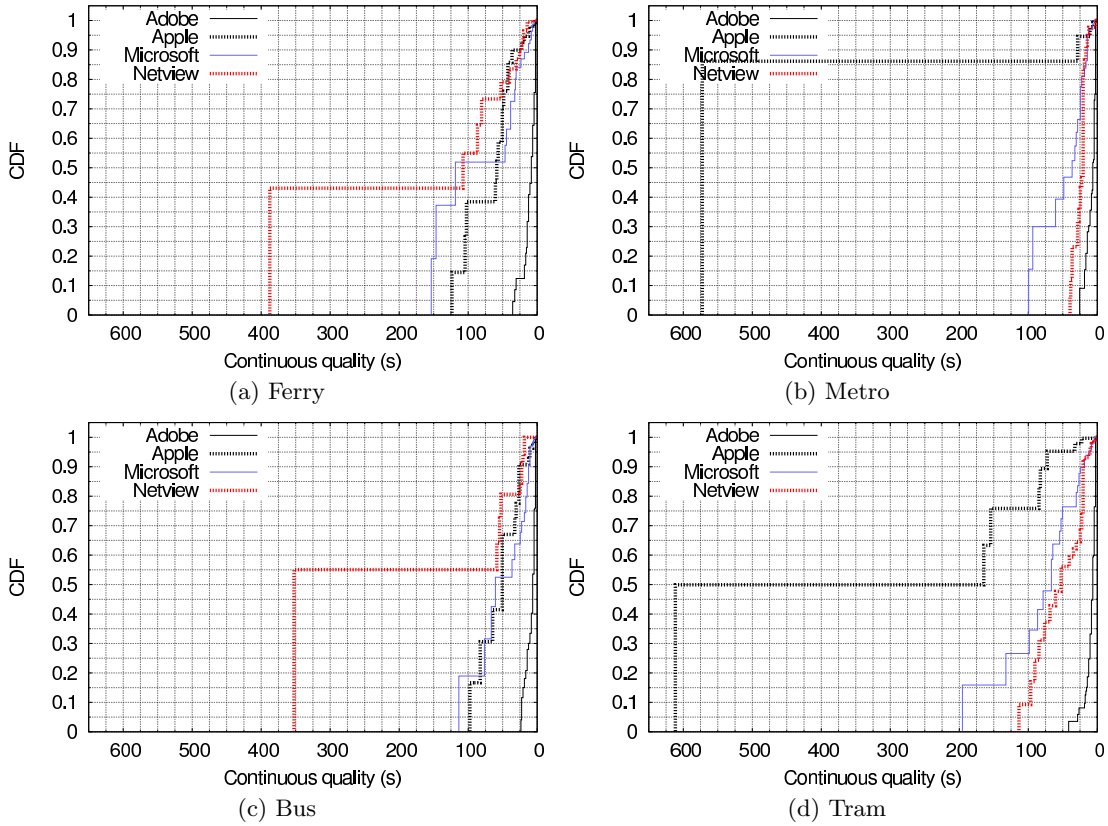


Figure 5: Cumulative distribution function of quality stability on the four commute routes. A point in the plot represents how much of the playout time (y-axis) has been played out with stable quality intervals larger than a given length (x-axis). Buffer underruns are not counted. *Note:* This plot considers only stability but ignores the quality of the playout interval (the long intervals of the Apple player in figures 5(b) and 5(d) are playouts at quality level 1 as seen in figures 3(b) and 3(d)).

5. REFERENCES

[1] Akamai HD for iPhone encoding best practices. `http://www.akamai.com/dl/whitepapers/Akamai_HDNetwork_Encoding_BP_iPhone_iPad.pdf`, 2010.

[2] HTTP dynamic streaming on the Adobe Flash platform. `http://www.adobe.com/products/httpdynamicstreaming/pdfs/httpdynamicstreaming_wp_ue.pdf`, 2010.

[3] The Apache HTTP server project. `http://httpd.apache.org/`, 2011.

[4] Netview Media Client. `http://www.netview.no/index.php?page=downloader`, 2011.

[5] tcpdump. `http://www.tcpdump.org/`, 2011.

[6] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proc. of ACM MMSys*, pages 157–168, Feb. 2011.

[7] CodeShop. Unified Streaming Platform. `http://www.unified-streaming.com/`, 2011.

[8] P. Ni, A. Eichhorn, C. Griwodz, and P. Halvorsen. Fine-grained scalable streaming from coarse-grained videos. In *Proc. of ACM NOSSDAV*, pages 103–108, June 2009.

[9] R. Pantos and W. May. HTTP live streaming. `http://tools.ietf.org/html/draft-pantos-http-live-streaming-07`, 2011.

[10] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. Video streaming using a location-based bandwidth-lookup service for bitrate planning. *TO APPEAR IN ACM TOMCCAP*, 2011.

[11] H. Riiser, P. Halvorsen, C. Griwodz, and B. Hestnes. Performance measurements and evaluation of video streaming in HSDPA networks with 16QAM modulation. In *Proc. of IEEE ICME*, pages 489–492, June 2008.

[12] H. Riiser, P. Halvorsen, C. Griwodz, and D. Johansen. Low overhead container format for adaptive streaming. In *Proc. of MMSys*, pages 193–198, Feb. 2010.

[13] T. Stockhammer. Dynamic adaptive streaming over HTTP: Standards and design principles. In *Proc. of ACM MMSys*, pages 133–144, Feb. 2011.

[14] A. Zambelli. Smooth streaming technical overview. `http://learn.iis.net/page.aspx/626/smooth-streaming-technical-overview/`, 2009.

[15] M. Zink, O. Künzel, J. Schmitt, and R. Steinmetz. Subjective impression of variations in layer encoded videos. In *Proc. of IWQoS*, pages 137–154, 2003.