Enhancing Video-on-Demand Playout over Multiple Heterogeneous Access Networks

Dominik Kaspar*† Kristian Evensen*† Paal Engelstad†*‡ Audun F. Hansen* Pål Halvorsen*† Carsten Griwodz*†
*Simula Research Laboratory, Norway †University of Oslo, Norway ‡Telenor R&I, Norway
email:{kaspar, kristrev, paalee, audunh, paalh, griff}@simula.no

***Abstract*—Multimedia streaming is increasing in popularity and has become one of the dominating services on the Internet today. Even though user devices are often equipped with multiple network interfaces and in reach of several access networks at the same time, media streams are normally communicated over only one of the available Internet connections.**

In this paper, we explore the challenges and potential benefits of using multiple access networks simultaneously. Exploiting HTTP's capability of handling requests for specific byte ranges of a file, we present the implementation of a lightweight, application-layer, on-demand streaming service that requires no changes to existing servers and infrastructure.

Based on real-world experiments with a multihomed host, we investigate the potential performance gains of video-on-demand playout. We achieve a bandwidth aggregation efficiency of 90% when downloading over 3 heterogeneous access networks in parallel. In addition, we analyze the effect of file segmentation on the buffer requirements and the startup latency.

I. Introduction

Today, mobile devices are typically equipped with multiple network interfaces supporting different technologies. For example, many smart phones and laptops can connect to both 3G and WLAN networks. Provided that a multihomed device is within coverage range of more than a single access network, any secondary interface can potentially be used for increased performance. Aside from enhanced connectivity and possibilities of seamless handover, aggregating the bandwidth of multiple interfaces is becoming increasingly interesting for providing high-quality video-on-demand (VoD) services to mobile devices. There are even claims that in 2013, almost 64% of the world's mobile data traffic will be video [1].

A major hurdle in the deployment of a multilink solution is the lack of server-side support. Although there have been suggested modifications to TCP (e.g., [10]) and SCTP (e.g., [4]), standard transport protocols are unable to provide host-based bandwidth aggregation. A common approach is therefore to provide specialized libraries (such as PSockets [7]) that transparently partition application-layer data into multiple transport streams. However, the implementation of such middleware requires software modifications to clients and servers.

In order to provide easy deployment and interoperability with existing server infrastructure, the Hypertext Transfer Protocol (HTTP) [2] can be exploited to download a single file over multiple links. Using the popular and widely supported HTTP protocol allows for a lightweight and client-based implementation. As illustrated in Figure 1, HTTP supports so-called *range retrieval requests* (range requests in short), which are commonly used to allow a halted download to proceed with the outstanding part at a later time. In this paper, we suggest the use of range requests to download unique segments of a file over multiple links available at the host.

The retrieval of video segments from one or more servers with HTTP range requests is a commonly used technique in commercial content distribution. For example, *Move Networks* [6] provides a delivery service to VoD providers that imports content into a delivery system and distributes it to clients. Each client implements an HTTP-based pull approach that makes transparent use of replicated servers. Move Networks supports the adaptation of video quality to available bandwidth by creating several versions of the content, allowing the client to choose the best version of a video segment at playback time. However, Move Networks does not provide support for hosts with multiple network interfaces and strongly focuses on high-speed Internet connections.

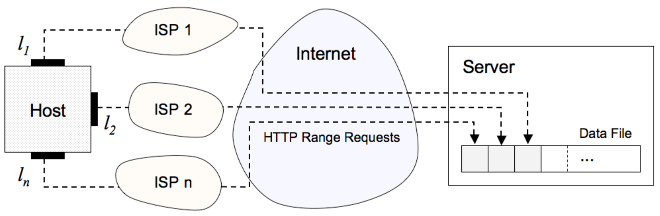


Fig. 1. General scenario of a multihomed host connected to n different access networks (i.e., ISPs). The host utilizes the links l_1, ..., l_n to simultaneously download a file from a server in many fixed-size segments.

HTTP range requests are also frequently used by *download managers* to achieve a larger throughput by opening multiple transport flows over a single network. Searching the Internet for existing download managers, we have found over 40 such tools [8]. Most of them are advertised with a multitude of features, such as the ability to connect to multiple mirror sites and multi-protocol support. However, to the best of our knowledge, not a single existing download manager actively supports data transfer over multiple access links.

Apart from implementation challenges, the main difficulty of efficiently combining multiple interfaces is the instability of the different wireless links and the heterogeneity of the different access networks [5]. While wired networks are usually very stable and predictable in terms of throughput, latency and loss, wireless links exhibit severe dynamics. A typical example of throughput variances in wired and wireless networks is depicted in Figure 2. While the used Ethernet connection exhibits a stable throughput, both WLAN and HSDPA fluctuate significantly over short periods of time. An important goal is to achieve an aggregated throughput that equals the sum of all links individually, regardless of variations.

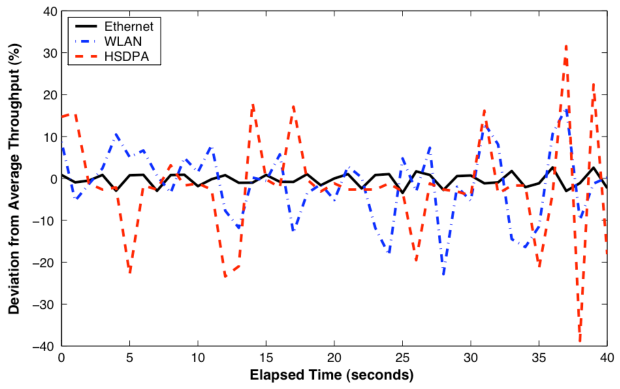


Fig. 2. Short-term throughput variances of different access technologies: Ethernet, Wireless LAN, and High-Speed Downlink Packet Access (HSDPA).

In this paper, we investigate the playout performance of multimedia files that are progressively downloaded over multiple wireless access networks in parallel. Section II explains and illustrates the challenges associated with progressive download. After introducing our method of performance analysis in Section III, we use Section IV to demonstrate the potential of our multilink approach in terms of increased throughput, reduced buffer requirements and startup latency. From this exploration, we identify several exciting topics for future work. Finally, Section V concludes the paper and discusses the numerous opportunities for future improvements.

II. Multilink Progressive Download

The term *progressive download* describes the method of sequentially transferring a media file from a server, allowing data playout on the client while the download is still in progress. Initially coined as "Fast Start" in Apple's QuickTime Streaming Server [3], progressive download is also supported by several other commercial streaming products, including Microsoft's Internet Information Services [9].

A particular property of progressive download is the use of client-side buffering for achieving smooth playback during periods of congestion. If the buffer is emptied too fast, the playout pauses until sufficient data is available again. A startup latency for filling the buffer prior to playout is therefore required to reduce the chance of an interrupted video experience. The tolerable startup latency is subjective and dependent on the media content, but should be as small as a few seconds.

Enabling progressive download over multiple network interfaces is very promising in terms of bandwidth aggregation and potentially higher quality multimedia streams. In addition, several independent and collaborating interfaces are more robust against link variances, congestion and failures. However, due to heterogeneity, progressive download over multiple links raises new challenges for efficient file segmentation. Dividing a media file into larger segments poses the risk of long startup latencies and large buffer requirements. On the other hand, creating small segments allows a smooth playout at the cost of reduced throughput due to segmentation overhead.

A. Segmentation Overhead

Analogous to Figure 1, linear playout over heterogeneous and dynamic links can be achieved by dividing a requested file into n fixed-sized segments. Whenever a link has finished downloading a segment, it sends an HTTP request for the next in sequence. Thus, for large numbers of segments, the overhead caused by file segmentation is dominated by the time T_{HTTP} it takes to request the server for a new byte range, which amounts to 1 round-trip time. The effect of T_{HTTP} depends on the delay heterogeneity. In general, the total time overhead T_{tot} of linear segmentation is a combination of several factors and can be summed up as:

$$T_{tot} = T_{TCP} + n * T_{HTTP} + T_{CPU} + T_{\Delta} \qquad (1)$$

The time T_{TCP} is the overhead caused by TCP's three-way handshake (2 RTTs) and slow-start mechanism when establishing a new connection. This happens once for each interface before the first segment can be transferred and is therefore not avoidable. For common hardware, the processing overhead of the application itself, T_{CPU}, is several orders of magnitude lower than T_{HTTP} and therefore negligible. Other delays (T_{Δ}) may be caused by packet loss. Although we have discovered a few cases of lost SYN packets causing TCP timeouts (3 s is a common default value), these occurrences have insignificant impact on our overall analysis.

B. An Example Playout Timeline

Based on a field experiment using a WLAN and an HSDPA network simultaneously, Figure 3 illustrates the characteristics of progressive download when multiple interfaces are used.

Even though segments are requested sequentially over the available links, incoming packets at the client may contain data that is not linear with respect to the playout order. In other words, the use of multiple links causes gaps in the received file. Most of the time during a transfer, some data must therefore be buffered, because it is received and available, but blocked by gaps and therefore not ready for playout.

In order to compensate for the stepwise increase in ready data and to allow a playout bitrate equivalent to the total aggregated throughput, a startup latency must be introduced. Under the assumption of constant-bitrate video encoding, the long-term sustainable playout rate can be represented as a line parallel to the total aggregated data received, with an x-axis offset equal to the startup latency.

III. Performance Metrics

For measuring the performance of progressive download, there are three general metrics that also apply in the scenario of simultaneously utilizing multiple networks:

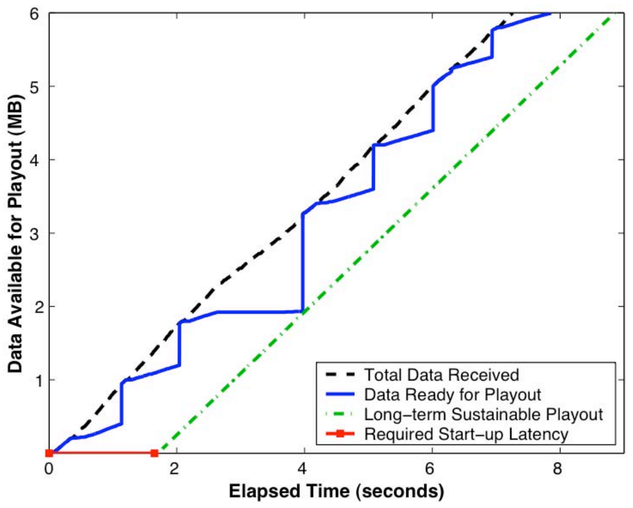
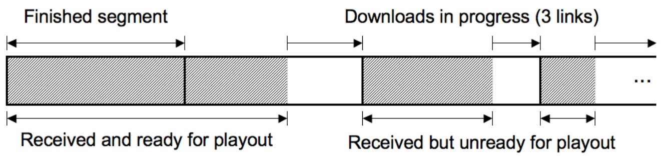


Fig. 3. Playout timeline of progressive download: the use of heterogeneous access networks (in this case WLAN and HSDPA) causes gaps in the received file and results in a step-wise increase of data ready for playout.

1) Playout bitrate: the bitrate at which a movie is played out is highly relevant for the user experience in terms of image resolution or frame rate. When using multiple links, efficient bandwidth aggregation is necessary for maximizing the playout bitrate.
2) Startup latency: in order not to exhaust the viewers' patience, it is important to minimize the startup waiting time. As shown in Figure 3, for every playout timeline, there exists an optimal startup latency. Due to unknown future link variances, it is a challenge to predict an appropriate startup latency.
3) Buffer requirements: for low-power and memory-constraint devices, it is crucial to minimize the buffer occupancy.

Due to network heterogeneity and dynamically changing link conditions, the maximum achievable playout rate, the minimum possible startup latency and the buffer requirements are unknown until a download is complete. Therefore, to investigate what the optimal values of these metrics are in reality, we conducted our analysis based on log files that include a precise history of the download progress. Figure 4 illustrates an example snapshot of the received byte ranges during a progressing download.



Finished segment
Downloads in progress (3 links)
...
Received and ready for playout
Received but unready for playout

Fig. 4. Time snapshot of a download in progress. The shaded areas represent currently received data. Data that follows a gap is unready for playout and must be buffered.

Once a file download with a given segment size has completed, the average aggregated throughput can be calculated by dividing the total file size by the transfer duration. Following the goals of maximizing the movie quality while ensuring a long-term sustainable playout, we set the desired playout bitrate equal to the average aggregated throughput.

After the desired playout $bitrate$ is decided, the optimal startup latency and the required buffer requirements can be calculated from the history of received bytes. By virtually playing back the entire download, it is possible to find out the number of bytes that are ready for playout at any given point in time ($bytes\_ready$). Knowing the exact received byte ranges, it is also possible to determine the amount of data that must be buffered because it is blocked by gaps in the playout order ($bytes\_unready$).

In order to obtain the minimum required startup latency, it is possible to calculate for every time t how many bytes are needed ($bytes\_needed$) for achieving a uninterrupted and sustainable playout at the desired bitrate:

$$bytes\_needed(t) = bitrate * t - bytes\_ready(t) \tag{2}$$

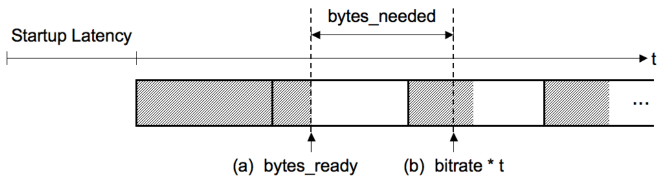


Fig. 5. The required startup latency can be illustrated as a race between (a) the number of bytes ready for playout and (b) the desired constant-bitrate playout. The startup latency must be able to compensate for the maximum difference between the two pointers (a) and (b).

As depicted in Figure 5, the required startup latency is the maximum number of bytes needed to satisfy the desired playout bitrate. It can be calculated as follows:

$$startup\_latency = \frac{\max_t(bytes\_needed(t))}{bitrate} \tag{3}$$

The required buffer size can generally be understood as the maximum capacity needed to store those packets that were received during the startup waiting time. However, in a scenario of sending file segments over multiple links, the buffer size is additionally influenced by received data that is not in correct playout order. In other words, the buffer occupancy at a given point in time is the number of received but unready bytes plus the number of playout-ready bytes needed to guarantee smooth playout. The maximum buffer requirements buf_req can thus be calculated as:

$$buf\_req = \max_t(bytes\_unready(t) + bytes\_needed(t)) \tag{4}$$

In consideration of these three metrics, Section IV will explore the potentials of using multiple links for enhancing the performance of on-demand progressive download. Rather than introducing a complete multilink solution, our current intention is to gain insight from real-world experiments and to identify the most promising opportunities for further optimizations.

IV. Experimental Results

In the following subsections we discuss the performance of progressive download in a real-world scenario with a multihomed host that is simultaneously connected to a WLAN and up to two independent HSPDA networks. The measured link characteristics between the used client and server are summarized in Table I.

TABLE I
OBSERVED CHARACTERISTICS OF USED LINKS

	WLAN	HSDPA
Maximum achievable throughput	640 KB/s	320 KB/s
Average experienced throughput (Φ)	600 KB/s	250 KB/s
Average RTT for header-only IP packets	20 ms	100 ms
Average RTT for full-size IP packets	30 ms	220 ms

All the presented results are based on downloading a 25.2 MB large file from a geographically local web server (the path lengths from the client to the server are in the range of 7 to 9 IP hops).

A. Bandwidth Aggregation

The most obvious advantage of downloading a file over multiple network interfaces in parallel is the expected throughput gain. When simultaneously transferring data over n different links l_i ($i \in \{1, ..., n\}$) with average throughput $\Phi(l_i)$, the main objective is to maximize the bandwidth aggregation efficiency η, which we define as:

$$\eta = \frac{\Phi(l_1 + ... + l_n)}{\Phi(l_1) + ... + \Phi(l_n)} \tag{5}$$

In other words, for achieving ideal bandwidth aggregation, each link should transfer data at its full capacity at all times; idle periods should be kept as rare as possible.

Figure 6 compares the throughput gain of combined heterogeneous networks to the traditional method of using the fastest interface alone. For optimal segment sizes (around 1000 KB in these experiments) the aggregation efficiency is η = 90%.

When choosing small segment sizes, the utilization of the links is often interrupted due to frequent requests to the server. This causes a significant loss in bandwidth aggregation efficiency, which indicates that there exists a tradeoff between the segmentation overhead and the maximum achievable throughput. Choosing a small segment size increases the overhead and therefore reduces the aggregation efficiency. For very small segment sizes, it might even be advantageous to solely use the highest capacity link.

On the other hand, the use of large segments is problematic for two reasons. First, there is the *last segment problem*, which is most obvious when a file is divided in two segments. Since the networks are heterogeneous, one interface will finish first and remain idle until the entire download has finished. Second, as will be shown later in Sections IV-B and IV-C, the use of large segment sizes creates a significant increase in the required startup latency and buffer requirements.

From the results presented in Figure 6, it becomes evident that an optimal segment size must exist that maximizes the

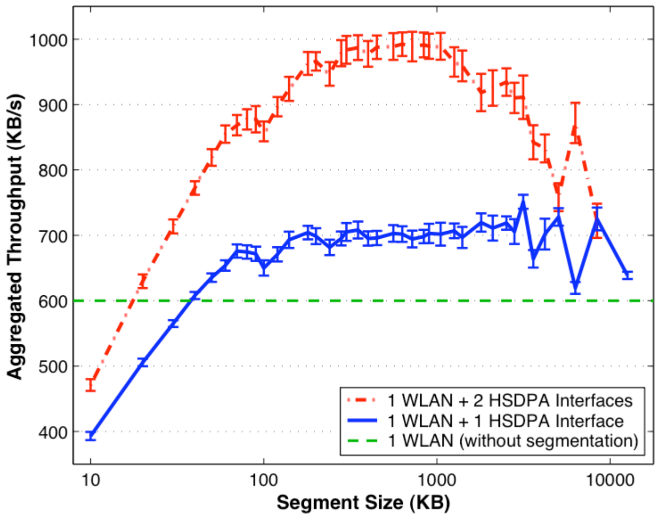


Fig. 6. The efficiency of bandwidth aggregation heavily depends on the segment size. In this progressive download scenario, a 25.2 MB large file was downloaded 50 times using up to 3 combined WLAN and HSDPA networks.

aggregated throughput. The optimal segment size depends primarily on the delay heterogeneity of the used links. It should therefore be possible to dynamically adjust the segment size according to predictions based on link monitoring feedback.

B. Startup Latency

The required waiting time before the playout of an on-demand video is an important factor for the user-perceived quality of service. For example, when starting a full-length movie of two hours, users might be willing to wait a minute before show time, but for shorter clips, the startup should be perceived as instantaneous.

Due to link variations, the optimal startup latency is unknown in advance. Additionally, the required startup latency depends on the desired playout bitrate, which is bounded by the aggregated throughput over the combined links. If the desired playout bitrate is well below the aggregated throughput, then the startup latency could be reduced. Our goal is, however, to gain from multiple links and to achieve a playout rate that approaches the average aggregated throughput.

Figure 7 reports the startup latency that is required to obtain a playout bitrate equal to the average aggregated throughput. Not only for bandwidth aggregation, but also for the required startup latency, file segmentation plays a crucial role. When the segment size is small, it takes less time for transferring a full segment, which reduces the startup latency. From a consumer's perspective, choosing the segment size translates to the decision between a shorter startup latency or a higher playout quality.

In addition to the segment size, the startup delay also depends on the throughput heterogeneity of the links. The larger the throughput difference between the slowest link and the others, the larger the amount of bytes needed to satisfy the desired playout rate (see $bytes\_needed$ in Figure 5).

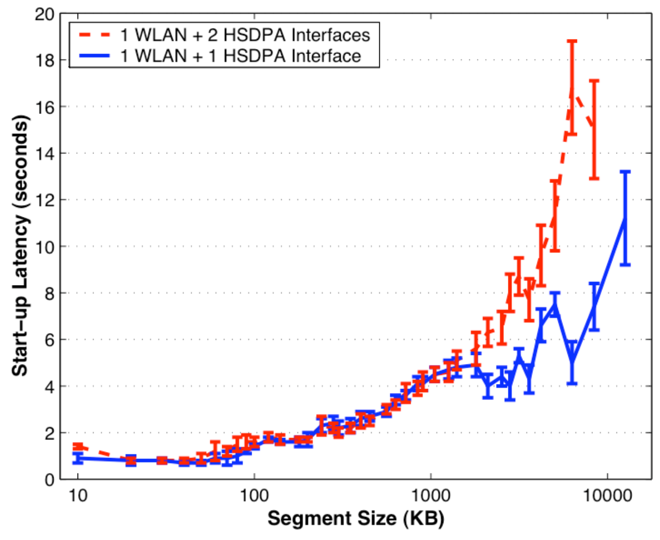


Fig. 7. The required startup latency grows with increasing segment size.

In a deployable implementation, a precise estimation of the startup latency would be necessary, for instance by using the startup phase itself for link monitoring purposes. Another approach of reducing the startup latency could include a TCP-like slow growth in segment sizes.

C. Buffer Requirements

In order to store received data that is not ready for playout, buffering is needed. From a consumer's perspective, the required buffer size is of little relevance. However, for the development of multilink on-demand streaming solutions on devices with scarce memory, the maximum required buffer capacity represents an important design factor.

The maximum expected buffer size increases with the chosen segment size and is in close correlation to the required startup latency. The only major difference between the two metrics is their dependence on the number of used interfaces. As the startup latency is dominated by the slowest link used, the aggregation of additional faster interfaces will not have any significant impact on the startup latency. On the other hand, the buffer capacity increases with the number of used network interfaces, because each interface causes additional out-of-order data to be received and buffered.

In a solution for devices with memory constraints, it will be important to find an optimal balance of the playout bitrate and the startup latency, given a limited buffer capacity.

V. Conclusions and Future Work

In this paper, we have analyzed the utilization of multiple wireless networks for improving on-demand progressive download. The presented proof-of-concept implementation operates in the application layer and uses HTTP range retrieval requests to sequentially transfer a file in segments from a server to a multihomed host. Even though it is common for video-on-demand solutions to employ file segmentation techniques, there has so far been a lack of studies that demonstrate how progressive download performs when multiple heterogeneous wireless networks are used simultaneously.

From field experiments with combined HSDPA and WLAN networks we reached the conclusion that there exists an optimal segment size for which the aggregation efficiency is maximized. In reality, however, the application does not know the segment size that achieves the highest aggregated throughput. Therefore, our future plans include several optimizations for the prediction of the optimal segment size and the required startup latency. Both properties are heavily dependent on the delay heterogeneity and variances in throughput, which necessitates the use of link monitoring and dynamic resizing of segments. It is also conceivable that a proportional adjustment of segment sizes to each link's capacity will decrease the number of necessary requests, reduce the periods of underutilized links, and therefore improve the aggregation efficiency.

As a concluding summary, Table II aligns all challenges that have been identified during our current research with possible solutions for future work.

TABLE II
Problems with segmented file download over multiple links

	Problem	Possible solution
P1	The time overhead of frequently requesting small segments causes inefficient throughput aggregation.	Utilize a mechanism for pipelining requests and permanently keeping the server busy.
P2	It is a challenge to predict the optimal segment size that maximizes the aggregated throughput and minimizes the startup latency.	Before the actual download begins, employ a short phase of link monitoring to gain knowledge of the delay and throughput characteristics.
P3	In highly dynamic networks, fixed-size segments increase the required startup latency and the buffer load.	Employ continuous link monitoring and adjust the segment sizes in proportion to the links' throughput.
P4	The last segment is often downloaded by the slowest interface, causing idle time on all others.	The last segment may be split among all interfaces in a divide-and-conquer fashion.

References

[1] Cisco Systems, I. Cisco visual networking index: Forecast and methodology, 2008–2013. Tech. rep., Cisco Systems, Inc., June 9 2009.

[2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext transfer protocol – HTTP/1.1. IETF, 1999.

[3] Inc., A. Mac OS X server – QuickTime streaming and broadcasting administration, 2007.

[4] Iyengar, J. R., Amer, P. D., and Stewart, R. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Trans. Netw. 14*, 5 (2006), 951–964.

[5] Kaspar, D., Evensen, K., Hansen, A. F., Engelstad, P., Halvorsen, P., and Griwodz, C. An analysis of the heterogeneity and IP packet reordering over multiple wireless networks. In *IEEE Symposium on Computers and Communications (ISCC)* (2009).

[6] Networks, M. Internet television: Challenges and opportunities. Tech. rep., Move Networks, Inc., November 2008.

[7] Sivakumar, H., Bailey, S., and Grossman, R. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. *Supercomputing, ACM/IEEE 2000 Conference* (Nov. 2000), 38–38.

[8] Wikipedia. Comparison of download managers, June 2009.

[9] Zambelli, A. IIS smooth streaming technical overview. Tech. rep., Microsoft Corporation, March 2009.

[10] Zhang, M., Lai, J., Krishnamurthy, A., Peterson, L., and Wang, R. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *ATEC '04: Proceedings of the USENIX Annual Technical Conference* (2004).