# Using HTTP Pipelining to Improve Progressive Download over Multiple Heterogeneous Interfaces

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

***Abstract*—Today, mobile devices like laptops and cell phones often come equipped with multiple network interfaces, enabling clients to simultaneously connect to independent access networks. Even though applications, such as multimedia streaming and video-on-demand delivery systems, could potentially benefit greatly from the aggregated bandwidth, implementation and standardization challenges have so far hindered the deployment of multilink solutions.**

**Previously, we have explored the benefits of collaboratively using multiple Internet connections to progressively download and play back large multimedia files. In this paper, we present an improved version of our approach that utilizes HTTP's capability of request pipelining in combination with range retrieval requests. While, in our earlier work, the optimal choice of file segmentation size presented a tradeoff between throughput and startup latency, the enhanced solution is able to overcome this tradeoff. The use of very small segments no longer impairs the efficiency of throughput aggregation, which additionally makes our solution robust against link variances and agnostic to network heterogeneity.**

## I. Introduction

Multimedia streaming is increasing in popularity and has become one of the dominating services on the Internet today. At the same time, there is an ongoing trend of integrating multiple wireless network interfaces into user devices, such as laptops and cell phones. Such mobile devices, which are becoming increasingly popular for downloading and viewing multimedia content [1], are often located in the coverage area of multiple wireless networks, such as wide-area 3G access networks and local-area WiFi networks. Especially in environments with weak signal reception, the simultaneous utilization of all available network interfaces on multihomed clients has a great potential of improving the service quality and the user experience.

A major hurdle in the deployment of a multilink solution is the lack of server-side support. Although there exist suggested modifications to TCP (e.g., [2]) and SCTP (e.g., [3]), standard transport protocols are unable to provide host-based aggregation of individual flows. Thus, a common approach is to provide specialized libraries (such as PSockets [4]) for transparent partition of application-layer data into multiple independent transport streams. However, the implementation of such middleware requires software modifications to all involved clients and servers.

In order to provide easy deployment and interoperability with existing server infrastructure, we have previously proposed a purely client-based solution for progressively downloading a single large file over multiple interfaces [5]. As illustrated in Figure 1, this application-layer method is utilizing the *range retrieval request* feature of HTTP [6], allowing logical file segmentation and cooperative download over multiple access networks.

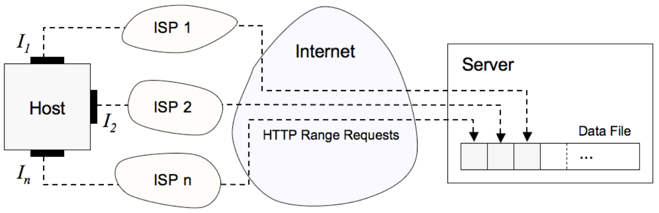

Fig. 1. General scenario of a multihomed host connected to $n$ different access networks (i.e., ISPs). The host utilizes the interfaces $I_1$, ..., $I_n$ to simultaneously download a file from a server in many fixed-size segments.

Our previous work demonstrated the tradeoff of achieving a high-bitrate multimedia playback versus the required waiting time (i.e., startup latency) before playback [5]. In other words, dividing the requested file into small-sized segments leads to a small startup latency and smooth playback, but requesting many segments introduces additional time overhead, which significantly reduces the efficiency of throughput aggregation. In addition to this tradeoff, we have identified numerous problems and potential solutions, as listed in Table I. The identified problem space can be translated into four main requirements for an improved solution (see Table II).

TABLE I
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 sending data. |
| 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 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. |

The remainder of this paper is organized as follows: in Section II, related work on media content distribution using

TABLE II
Requirements for Multilink Progressive Download

| | Requirement |
|---|---|
| R1 | The throughput of each additional interface must be efficiently aggregated to allow an increased throughput and a better playback quality. |
| R2 | For achieving a smooth playback experience, the startup latency and the number of playback interruptions should be minimized. |
| R3 | The solution should be agnostic to the heterogeneity of the used access networks. |
| R4 | The solution should dynamically adapt to link variations. |

file segmentation is described and compared to our ongoing research. After introducing our experimental testbed in Section III, the advantages and challenges with HTTP request pipelining are described and analyzed in Section IV. This paper's main contribution is the use of pipelining to eliminate idle time overhead, allowing a fine-grained file segmentation that ensures both a smooth progressive playback and efficient throughput aggregation of multiple interfaces. In Section V, it is discussed how this approach meets all the requirements listed in Table II. Before the final conclusions are made, Section VI demonstrates the performance of our system by means of a real-world experiment.

## II. Related Work

Application-layer techniques of file segmentation in content distribution systems have been suggested for two main reasons. First, the placement of multimedia file segments on replica proxies is a commonly used technique to improve the response time of video-on-demand distribution systems [7]. Second, dividing video files into segments and creating several versions of the content allows a smooth adaptation of video quality to the available bandwidth [8]. However, the schemes proposed in the literature fail to leverage the potential of hosts with multiple network interfaces and tend to ignore the variances and heterogeneity of wireless Internet connections.

In addition, the retrieval of video segments from multiple servers is a widely used practice in commercial content distribution. For example, *Move Networks* [9] offers a delivery service to video-on-demand 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.

HTTP range retrieval 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 [10]. 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.

## III. Testbed

The testbed used for the experiments presented in this paper is shown in Figure 1. All results were obtained in a real-world scenario with subscriptions to independent Internet service providers. The characteristics of the used wireless access networks are shown in Table III.

TABLE III
Observed Characteristics of used Links

| | IEEE 802.11b | HSDPA |
|---|---|---|
| Maximum net bandwidth | 1375 KB/s | 450 KB/s |
| Typical experienced throughput ($\Phi$) | 600 KB/s | 300 KB/s |
| Average RTT for header-only IP packets | 20 ms | 110 ms |
| Average RTT for full-size IP packets | 30 ms | 220 ms |

We had full control over both the client and the server in the experiment and we configured the server as a common HTTP Web server running Apache 2.2 on a Linux platform (Ubuntu 9.04). The server was assigned a globally reachable IP address.

## IV. HTTP Pipelining

HTTP pipelining is, according to the protocol specification [6], a method that "allows a client to make multiple requests without waiting for each response, allowing a single TCP connection to be used much more efficiently, with much lower elapsed time". As depicted in Figure 2, in the absence of pipelining (as implemented and analyzed in our previous work [5]), each range retrieval request must be sequentially handled by the server before the client can send the next request. Thus, for each request, an average time overhead of one round-trip time is incurred. For a large number of small file segments, this overhead significantly impairs the throughput of high-latency connections (such as HSDPA).

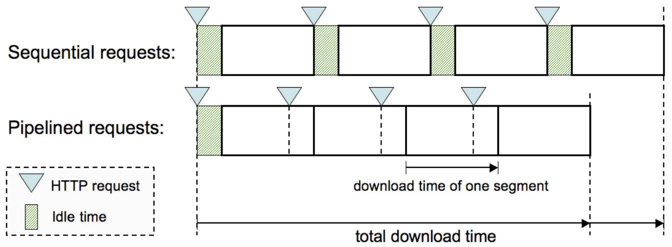

Fig. 2. From a client-side perspective, HTTP pipelining eliminates the time overhead incurred by a sequential processing of requests and replies.

If pipelining is not enabled, any attempt to achieve smoother playback by reducing the segment size is bound to increase the server's idle time and results in overall decreased throughput aggregation. However, if the client already sends a request for the next byte range before the currently downloading segment has completed, this time overhead can be eliminated. As shown in Figure 3, using HTTP pipelining helps to increase the aggregated throughput for small segment sizes.

### A. Interleaved Startup Phase

If pipelining is enabled, the optimal point in time for sending the next request is not evident (illustrated with small triangles in Figure 2). Requesting the next segment too late obviously degrades the performance of pipelining, while requesting it too early increases the probability of two

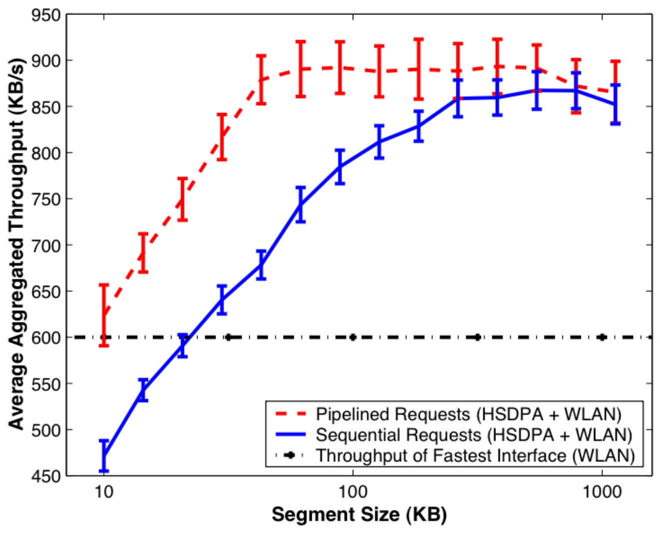

Fig. 3. The benefit of HTTP request pipelining – Using a pipelining mechanism eliminates the waiting time between successive requests and effectively increases the aggregated throughput. These results were obtained by simultaneously downloading a 50 MB large file over HSDPA and WLAN.

segments being assigned side-by-side, effectively resulting in double-size segments and one superfluous request. In order to overcome this dilemma, we suggest to schedule a predefined pipelining pattern during the startup phase (see Figure 4).

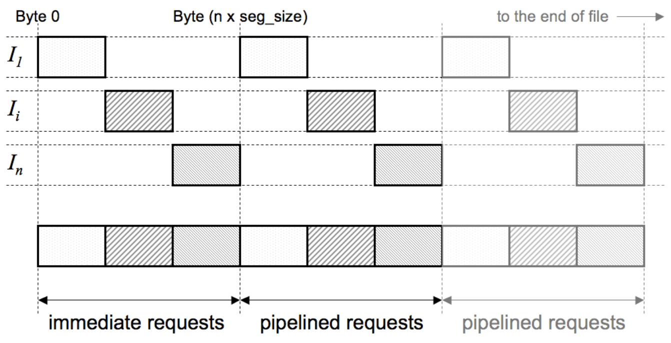

Fig. 4. Startup phase – Requesting segments over interfaces $I_1$, ..., $I_n$ in an interleaved pattern helps to provide a smooth playback and reduces the initial response time.

During the interleaved startup phase it is guaranteed that byte ranges requested over the same interface are not assigned side-by-side. After the interleaved startup phase, the dilemma of when to send the next request is eliminated. Each interface may simply send a new request right after it has completed a segment and the server's pipeline will always contain at least one request to be processed.

## B. The Minimum Segment Size Problem

In the attempt of achieving a smooth playback experience by employing a very fine-grained segmentation, there exists a risk of choosing the segment size too small. In this case, the server spends such a small fraction of time to process a request that the client has no time pipelining the next request. This happens if the one-way transmission delay $d_i$ over interface $I_i$ is larger than the time $t_i$ it takes for the HTTP server to send out a segment, which is given by the segment size $S$ and the average throughput $\Phi_i$ experienced over interface $I_i$:

$$t_i = S/\Phi_i \tag{1}$$

For example, for a 10 KB file segment over HSDPA, the server requires 33 ms (i.e., 10 KB / 300 KB/s) until the last byte of the segment is sent away. The problem here is that the 33 ms of sending out the data is much less than the average one-way transmission delay $d_i = 110$ ms experienced over the HSDPA link. Therefore, pipelining is not as efficient as expected, because the server has sent a segment and wants to send another one, but no other request is yet pipelined in the server's request queue.

For pipelining to work efficiently over a path with one-way delay $d_i$, a new request has to be sent from the client to the server at time $t_{req}$ after the current time $t_{now}$:

$$t_{req} = t_{now} + S/\Phi_i - d_i \tag{2}$$

In the example of sending 10 KB segments over HSDPA, $t_{req}$ is $-77$ ms. Thus, while the client is downloading a segment, the next segment must be requested *before* the current segment's download has even started. In other words, the segment size has been chosen too small.

The minimal segment size $S_{min}$ required for fully efficient pipelining, can be calculated by setting $t_{req} = 0$ and solving Equation 2 for the segment size $S$:

$$S_{min} = d_i * \Phi_i \tag{3}$$

The minimum segment size for the HSDPA path in our scenario is therefore 110 ms * 300 KB/s = 30 KB, which is equivalent to the bandwidth-delay product of the path.

Figure 5 illustrates the experimental determination of the $S_{min}$ value. Almost precisely at 30 KB, the graph reaches the typical experienced throughput. This figure also clearly shows that choosing too small segments causes a significant loss in throughput.

## C. Multi-Segment Pipelining

In order to alleviate the minimum segment size problem, it must be guaranteed that the amount of pipelined data always exceeds the path's bandwidth-delay product, which can either be solved by increasing the segment size or by pipelining multiple segments. In our implementation, the most elegant way of filling the server queue with multiple requests was to increase the length of the interleaving pattern during the startup phase.

The performance gain of multi-segment pipelining is shown in Figure 6. While single-segment pipelining suffers from a rapid decrease in throughput for segment sizes below the bandwidth-delay product, multi-segment pipelining achieves close to optimal throughput with file segments sizes in the order of a few IP packets.

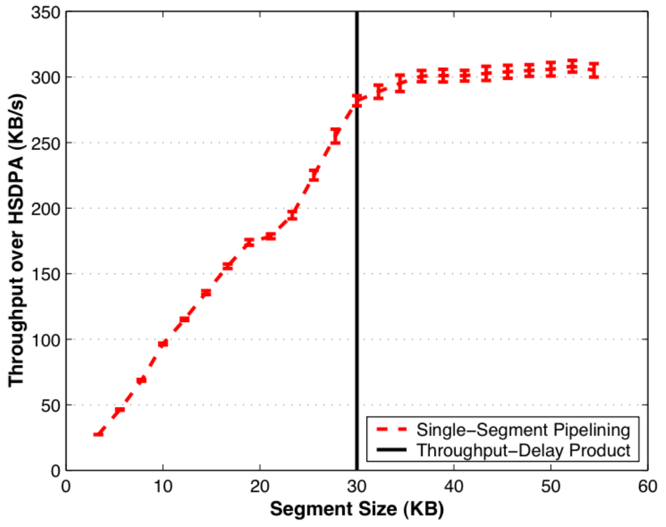

Fig. 5. The minimum segment size problem – pipelining a single segment is only useful for segment sizes larger than the bandwidth-delay product. Too small proportioned segments cause a significant performance penalty.

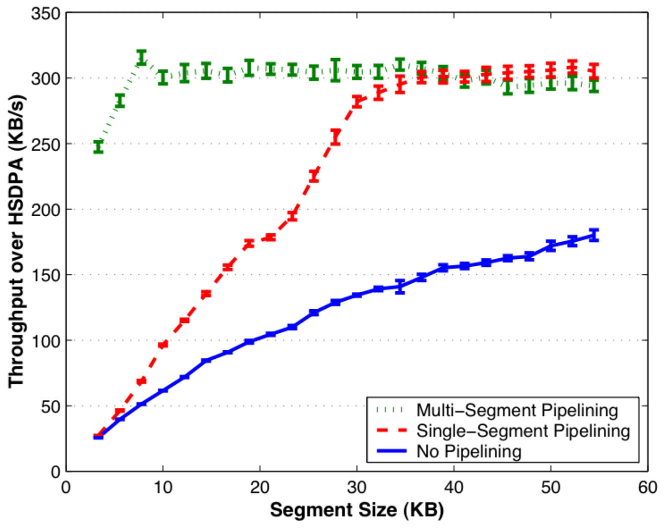

Fig. 6. By pipelining ("pre-requesting") multiple segments, the minimum segment size problem can be alleviated without the need for an increased segment size. This allows a high throughput aggregation efficiency at very small segment sizes and increases the chance for a smooth playback quality.

## V. Discussion

Looking back at Table II, this section evaluates how a pipelining-based approach is capable of satisfying the requirements of progressive video playback over multiple concurrent Internet connections.

### *A. Overhead Elimination*

The primary purpose of concurrently utilizing multiple network connections is to aggregate their throughput and provide a multimedia stream of higher quality. Section IV demonstrated how HTTP pipelining can be used to increase the throughput by eliminating any time overhead between consecutive requests. However, while the impact of byte overhead caused by HTTP header specifications has so far been ignored, such metadata noticeably reduces the throughput aggregation efficiency when the segmentation approaches a granularity of single IP packets. This effect is clearly visible in Figure 6 for the case of multi-segment pipelining when the segment size is less than 10KB.

In the approach described in this paper, each request includes the URL of the file to be retrieved, the server's IP address, and a byte range specification. For short URLs, the byte overhead of each request is therefore around 100 bytes. If a byte overhead of 1% is acceptable, the minimum segment size should be set to at least 10 KB. Despite this negligible tradeoff, the requirement R1 is fulfilled.

### *B. Smooth Playback*

The main challenge associated with playing back a video file, while progressively downloading it over multiple network interfaces, is to provide a smooth viewing experience. For an optimal collaboration of heterogeneous connections, it is desirable to provide a segmentation of optimal granularity. Additionally, using smallest possible segments minimizes the startup latency and the probability of blocking the playback by requesting a too large first segment over a slow link. Due to the fact that our solution allows arbitrarily small segments – only at the cost of reduced throughput aggregation efficiency – the requirement R2 is met, as well.

However, due to the subjectivity of perceived video quality, there will never be a clear answer to the question of what the best choice of the segment size is. Some viewers might prefer a reliably smooth playback, while others put more value on high-definition video with occasional re-buffering.

### *C. Network Heterogeneity Independence*

A frequent assumption in the literature on bandwidth aggregation and multihoming is the use of multiple interfaces of the same kind, such as multiple WLAN adapters (e.g., [11]), multiple 3G modems (e.g., [12], [13]), or otherwise homogeneous simulation parameters (e.g. [3]). In contrast, our implementation handles any type of heterogeneity and requires no a priori knowledge of the used interfaces before a file transfer starts. The reason is that a large number of small, fixed-size segments are naturally distributed over all links in proportion to their throughput.

Simply put, our brute-force scheme of using the smallest possible segment sizes trades off the complications of link monitoring and dynamic adaption at the cost of a negligible amount of byte overhead due to frequent HTTP requests. Requirement R3 can therefore be ticked off.

### *D. Robustness against Link Variances*

While a pipelining scheme based on small, fixed-size segments is agnostic to link heterogeneity and robust to frequent fluctuations in throughput and delay, the probability of link failure increases with the number of interfaces.

However, our current prototype is yet unable to detect link failure and does not support failover. Thus, segments pipelined over a broken connection would never complete. In the case of extreme throughput variances, multi-segment pipelining has the disadvantage that it is impossible to cancel already pipelined requests without closing the TCP connection. One solution to this problem is to detect broken connections and request segments redundantly. In the worst case, the amount of data pipelined over a broken link is lost. However, due to the smallness of the segments, this should not create any severe re-buffering during the movie experience. The smaller the segment size, the quicker is the download of a segment and the smaller the impact of a link that slows down.

Although certain optimizations are pending, from various successful experiments in our wireless testbed, we consider requirement R4 as satisfied.

## VI. Live Media Playback

As a concluding example, we present the results from a real-world experiment within the previously introduced testbed. In this experiment, a 50 MB large file was downloaded over multiple interfaces and, after a short startup delay of 3 seconds, sequentially read through at a constant bitrate.

Rather than analyzing complex video metrics that depend on the type of content and codec used, this experiment is based on a much more demanding metric: pure success vs. any type of failure. Each occurrence of the constant-bitrate file reader seeking ahead of the received data is counted as a failure. A success is only recorded when the entire file was received and played back without any need for data re-buffering.

Figure 7 depicts a direct performance comparison of a single WLAN interface versus the use of an additional HSDPA interface. The simultaneous use of multiple interfaces increases the probability of an interruption-free playback experience at a given media bitrate. Another way to interpret these results is that the unreliable WLAN link, which performs poorly at any bitrates above 600 KB can be made 100% reliable by adding an additional HSDPA link.

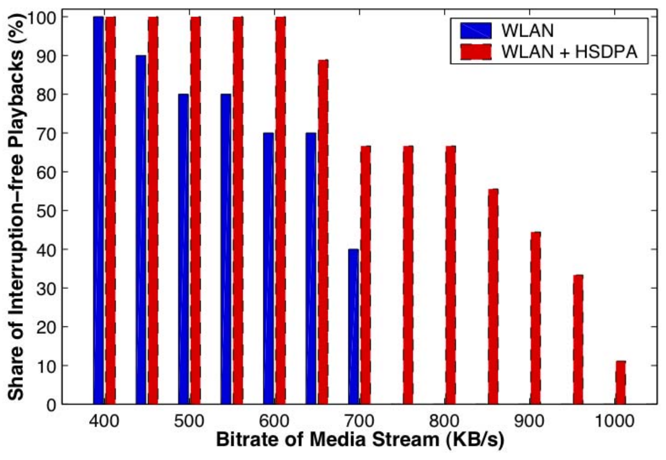

Fig. 7. Live media playback – Pipelining-based progressive download over multiple heterogeneous interfaces allows media streams of higher quality.

## VII. Conclusions and Future Work

In this paper, we have experimentally analyzed a novel approach to achieve high-quality and reliable progressive file download over multiple heterogeneous wireless networks. The introduced proof-of-concept implementation operates in the application layer and exploits HTTP's features of range retrieval requests and request pipelining to optimize the aggregated throughput, while at the same time ensuring a smooth and robust multimedia playback.

From field experiments with HSDPA and WLAN networks, we have gained insight into the advantages and challenges of pipelining requests over network paths that vary in their throughput and latency characteristics. Using a fine-grained segmentation and always pipelining enough data to fill the bandwidth-delay product allows a natural adaption to network heterogeneity and increases the probability of an interruption-free playback experience at a given media bitrate.

In the future, we plan to explore options for failover support to fully secure our solution against extreme conditions and broken network connections. One idea would be to adapt to a changing bandwidth-delay product by dynamically adjusting the number of requests in the pipeline, similar to TCP's AIMD scheme. In addition, we plan to catch some fresh air by testing our system in mobile, outdoor scenarios.

## References

[1] I. Cisco Systems, "Cisco Visual Networking Index: Forecast and Methodology, 2008–2013," tech. rep., Cisco Systems, Inc., June 9 2009.

[2] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang, "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.

[3] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent Multipath Transfer using SCTP Multihoming over Independent End-to-end Paths," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 951–964, 2006.

[4] H. Sivakumar, S. Bailey, and R. Grossman, "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks," *Supercomputing, ACM/IEEE 2000 Conference*, pp. 38–38, Nov. 2000.

[5] D. Kaspar, K. Evensen, P. Engelstad, A. F. Hansen, P. Halvorsen, and C. Griwodz, "Enhancing Video-on-Demand Playout over Multiple Heterogeneous Access Networks," in *Consumer Communications and Networking Conference (CCNC)*, 2010.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1." IETF, 1999.

[7] C. Griwodz, *Wide-area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure*. PhD thesis, Darmstadt University of Technology, 2000.

[8] D. Johansen, H. Johansen, T. Aarflot, J. Hurley, Å. Kvalnes, C. Gurrin, S. Sav, B. Olstad, E. Aaberg, T. Endestad, H. Riiser, C. Griwodz, and P. Halvorsen, "DAVVI: A Prototype for the Next Generation Multimedia Entertainment Platform (demo)," in *ACM Multimedia*, 2009.

[9] M. Networks, "Internet Television: Challenges and Opportunities," tech. rep., Move Networks, Inc., November 2008.

[10] Wikipedia, "Comparison of Download Managers," June 2009.

[11] S. Shakkottai, E. Altman, and A. Kumar, "The Case for Non-Cooperative Multihoming of Users to Access Points in IEEE 802.11 WLANs," in *INFOCOM*, 2006.

[12] A. Qureshi, J. Carlisle, and J. Guttag, "Tavarua: Video Streaming with WWAN Striping," in *ACM international conference on Multimedia*, (New York, NY, USA), pp. 327–336, ACM, 2006.

[13] K. Chebrolu and R. R. Rao, "Bandwidth Aggregation for Real-Time Applications in Heterogeneous Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 5, no. 4, pp. 388–403, 2006.