Server-Efficient High-Definition Media Dissemination

Philip W. Frey, Andreas Hasler, Bernard Metzler
Systems Department
IBM Research GmbH
8803 Rueschlikon, Switzerland
{phf,aha,bmt}@zurich.ibm.com

Gustavo Alonso
Systems Group
Department of Computer Science, ETH Zurich
8092 Zurich, Switzerland
alonso@inf.ethz.ch

ABSTRACT

Internet usage has changed dramatically in the past few years. Content is no longer dominated by static websites, but comprises an increasing number of multimedia streams. With the widespread availability of broadband connections, the quality of the media provided by video-on-demand as well as streaming services increases constantly. Even though today most videos are still encoded with a rather low bit rate, large Internet service providers already foresee high-definition media becoming the predominant format in the near future. However, a larger number of clients requesting media at high bit rates poses a challenge for the server infrastructure. Conventional stream dissemination methods, such as RTP over UDP or HTTP over TCP, result in high server loads due to excessive local data copy, context switching, and interrupt processing overhead. In this paper, we illustrate and discuss this problem in detail through extensive experiments with existing solutions. We then present a new approach based on zero-copy protocol stack implementations in software as well as dedicated RDMA hardware. Our performance experiments indicate that these optimizations allow servers to scale better and remove most of the overhead caused by current approaches.

Categories and Subject Descriptors

B.4.3 [**Input/Output and Data Communications**]: Interconnections (subsystems)—*Asynchronous/Synchronous Operation, Parallel I/O*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms

Performance, Human Factors, Reliability

1. INTRODUCTION

Today, about 23% of the world's population are connected to the Internet — most of them through broadband access. This number as well as the bandwidth offered by the Internet service providers (ISPs) are increasing rapidly [10]. In addition, web content is no longer static but has become dynamic and far richer. According to Comcast [17], the next step in the evolution of Internet content will be a shift from *standard-definition* (SD) towards *high-definition* (HD) media as well as from *broadcast* towards *unicast* services.

1.1 Problem Statement

According to Plagemann et al. [15], *high-definition video-on-demand* (HD VoD) poses two orthogonal service requirements:

1. a large enough data throughput to deliver the high-definition content, and
2. a low latency as well as a high responsiveness to support convenient media control.

Transmitting HD media using unicast to an increasing number of users renders meeting these requirements quite complex for a number of reasons. First, the content servers must be able to sustain a higher aggregate data throughput (the bit rate of a H.264-compressed HD video is roughly 10x larger than that of its SD equivalent). Second, the unpredictable behavior of each client makes it virtually impossible for the server to prefetch data efficiently and predict the aggregated service rate. The clients expect control over the content at all times (pause, skip or rewind as well as switching to another movie). The problem is amplified by the variable bit rate (VBR) typically used to encode the data.

In this paper, we show that traditional video server architectures are inadequate for HD VoD given the large overhead caused by redundant data copy operations, excessive interrupt handling, and context switches. Figure 1 illustrates the process followed to send data stored on a hard drive: The data is first copied from disk through a temporary kernel buffer into a user buffer; then it is copied back into another kernel buffer and finally copied to the network interface card (NIC). Overall, the process involves 2 DMA copies, 2 CPU copies and several context switches.

Scalability in existing systems is achieved by adding more servers. While that approach is rather simple, it increases the running costs of the server infrastructure due to higher server and network maintenance costs, as well as increased power consumption and cooling demands.

The problem we address in this paper is how to improve the scalability of a single video server (i.e., its ability to serve a larger number of clients) so that the demands of HD VoD

NOSSDAV'09, June 3–5, 2009, Williamsburg, Virginia, USA

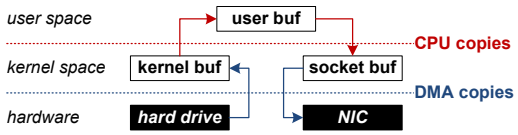

Figure 1: Copy Overhead for Sending File Content

on the overall server infrastructure are minimized. Such an improvement involves removing the overhead encountered in current systems. For this purpose, we propose a novel protocol based on *Remote Direct Memory Access* (RDMA [16]).

The contributions of this paper are three-fold. First, we analyze the performance of existing solutions and identify their key bottlenecks. Second, we propose a new protocol based on iWARP and prove through extensive experiments that it improves server-side scalability while offering efficient VCR-like media control, specially when compared with existing solutions and with using the kernel `sendfile()` mechanism. Third, we outline an extension to our new protocol to support real-time live media streams.

The paper is structured as follows: Section 2 introduces iWARP and presents related work, Section 3 compares UDP- with TCP-based application protocols that enable VoD services. Section 4 presents a novel iWARP/RDMA-based protocol for VoD as well as streaming services. Section 5 evaluates the proposed solutions and identifies their tradeoffs. Section 6 concludes our work.

2. BACKGROUND

2.1 iWARP: RDMA over Ethernet

Remote Direct Memory Access is a mechanism whereby, without CPU involvement, network data is directly fetched from local application memory and placed into the application memory of a remote computer. This placement is typically performed by an RDMA-enabled NIC (*RNIC*). In bypassing the operating system and eliminating intermediate copying across buffers, RDMA reduces the CPU cost of large data transfers as well as the end-to-end latency. This makes it very attractive for transferring HD media across the network. RDMA technology was initially restricted to specific, often proprietary technology, such as InfiniBand [8]. With the advent of 10 Gb Ethernet and the TCP-based Internet Wide Area RDMA Protocol (iWARP), a special infrastructure is no longer necessary. RDMA can now be applied to applications communicating over the Internet.

A detailed description of the RDMA semantic can be found in the protocol specification [7]. We focus our discussion on the aspects relevant for this paper. The RDMA data transfer operations include *Send*, *Receive*, *RDMA Read* and *RDMA Write*, which are all invoked and completed asynchronously. The Send and Receive operations are similar to the Socket API, but allow multiple outstanding operations, whereas RDMA Write and RDMA Read are semantically different. These so-called *one-sided* operations restrict active application involvement to the side issuing the write or read operation. At the other side, the data transfer is handled solely by the RNIC fetching or placing data. Neither the operating system nor the application is notified. To allow the RNIC to fetch and place data without operating system involvement, appropriate memory buffers have to be preregistered by the applications as *Memory Regions* (MR). The associated memory must be pinned to physical memory by the operating system to be instantly available for the RNIC if memory access is required. After local registration, the MRs are referenced by the RNIC and the peer application using unique *steering tags* (STags).

2.2 Related Work

Already in the early 90s, Fall and Pasquale suggested solutions to shortcut data paths within the operating system [4]. They proposed `splice()`, a new UNIX system call, which allows moving data between two file descriptors while avoiding copies between kernel- and user address space. Using `splice()` helps to reduce CPU load as well as the number of context switches. The same authors showed the applicability of the `splice()` system call to continuous-media playback over UDP transport [5]. As we will see, UDP used to be a valid choice for media transmissions at those low data rates, but it has to be reconsidered when moving to the high bit rates required by HD media content.

Another copy avoidance solution for on-demand media servers is presented by Halvorsen et al. [6]. A shared memory region is used between the hard drive where the media data reside and the network interface in order to create a specialized zero-copy data path from the storage medium to the communication system. This shared buffer is created at stream setup time and remains statically allocated throughout the transmission. The proposed zero-copy mechanism only assures that the data is not copied until it is handed over to the network subsystem. The further steps are outside the scope of this work. Although their goal is similar, our solutions are fundamentally different. They suggest UDP as a suitable transport layer and restrict their applicability to non-live media content. Furthermore the data is sent out only in fixed periodic intervals, whereas we offer a true on-demand solution with a VCR-like media control.

Accelerating HTTP by using RDMA technology has been proposed in [2]. An iWARP Apache module was presented that extends the HTTP protocol with RDMA support. While HTTP is push-based, the completely new protocol we propose is pull-based and independent of HTTP.

Further extensive research has been conducted on how to distribute media over the Internet based on the assumption that the available bandwidth is very limited [13, 18]. Detailed discussions of the different challenges posed by designing mechanisms and protocols for Internet streaming services can be found in [21, 15]. In our work, we assume the bandwidth to be sufficient and investigate server-side limitations. Extending our client-server solution to a peer-to-peer overlay is planned for the future.

3. CURRENT SOLUTIONS

We have examined the performance of existing VoD solutions through a series of experiments on a 10 Gb Ethernet fabric. To avoid disk access overheads, we preloaded the complete test data set into main memory. Our test bed consists of an IBM BladeCenter containing six HS21 BladeServers. Each of them is equipped with a quad core Intel Xeon CPU (2.33 GHz), 8 GB of main memory and a Chelsio T3 RDMA-enabled 10GbE NIC. The BladeServers are running a Fedora Linux 2.6.27 kernel with the OpenFabrics Enterprise Distribution v1.4 [14] for iWARP support. One BladeServer acts as the server (media source) and the

other five are connecting to it as clients (media sink). The streamed videos are encoded with the predominant codec for HD media (H.264). We compare different bit rates from SD quality with an average of about 1 Mbps up to full HD (1080p) requiring about 8.7 Mbps on average. Our main criterion of media distribution efficiency is the maximum number of clients the single server is able to serve concurrently without service degradation (frame loss or late frames). We have investigated the influence of other service parameters, such as offering varying numbers of movies, differently sized movies, movie access through various patterns as well as changing the client-side cache size. Compared with the actual bit rate, none of the other parameters had a significant impact on scalability. Therefore, they are not discussed further.

3.1 RTP-based Solutions

The motivation for using unreliable connectionless services such as UDP for video streaming is that media streams typically tolerate data loss better than delay and, therefore the higher reliability of TCP is not needed. UDP is most often used in combination with the *Real Time Transport Protocol* (RTP), which defines a standardized packet format for delivering audio and video. As UDP streams are unidirectional, RTP is typically accompanied by the *RTP Control Protocol* (RTCP), which monitors the transmission and feeds out-of-band control information back to the streaming source, and by the *Real Time Streaming Protocol* (RTSP), which provides control over the stream (i.e., play, pause, stop, etc.).

In our first test, we explore RTP scalability for various bit rates with special focus on the high bit rates needed for HD media content. We have conducted the experiments with the two most prevalent media servers for Linux that offer VoD services using RTP: The open source VideoLAN Server [3] and the Darwin Streaming Server [1].

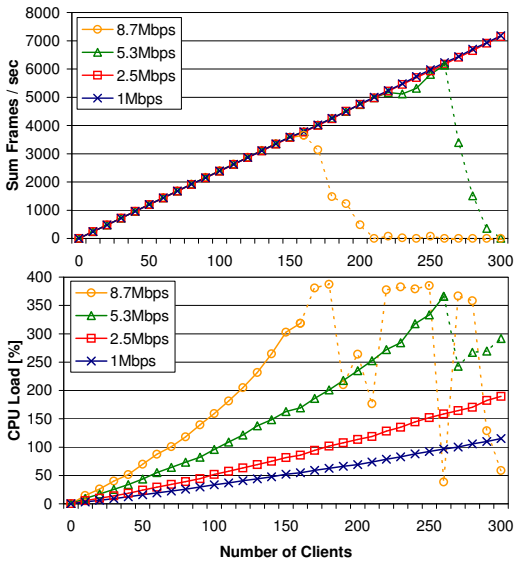

Figure 2: RTP Scalability for Various Bit Rates

The upper chart of Figure 2 shows the cumulative rate of frames which all clients received successfully. We have conducted the experiment with up to 300 clients, which the server can handle with moderate CPU load when using bit rates up to 2.5 Mbps (see lower chart of Figure 2). When streaming at 5.3 and 8.7 Mbps, the maximum number of serviceable clients drops to 260 and 160, respectively. Note, that since we are streaming over a 10 GbE link, the theoretical maximum of clients for the 8.7 Mbps stream is about 1100, a factor of 7 larger than what we experience. The lower chart of Figure 2 clearly reveals that the server CPU utilization is the limiting factor. When investigating the reason for the high CPU load using *oprofile* [9], we found that most of the CPU cycles are spent in copying data and RTP packetizing.

The erratic shape of the curves beyond the scalability limit (dashed parts of the plots) are caused by nondeterministic, nonreproducible behavior of the server due to the high CPU load. They indicate severe service degradation.

3.2 HTTP-based Solutions

Streaming media over HTTP assures good interoperability and server efficiency. YouTube as one of the main sources of current Internet media streams, for example, offers HTTP-based video dissemination. Even though HTTP is based on connection-oriented TCP and was not specifically designed to meet media-streaming requirements, it offers some advantages.

First of all, no special software is needed; any state-of-the art web browser will do. Furthermore, HTTP port 80 is allowed on most firewalls, whereas other ports potentially used by RTP are often blocked. Also, the TCP reliability feature can be desirable for highly compressed media where the loss of a key frame can cause severe playback disruption. Furthermore, the TCP flow control mechanism implicitly allows client-driven media control. Not reading new data and thus delaying TCP `Window` updates causes the sender to stall the stream (see Figure 3). Seeking is done by sending a new `HTTP GET` request with the desired offset.

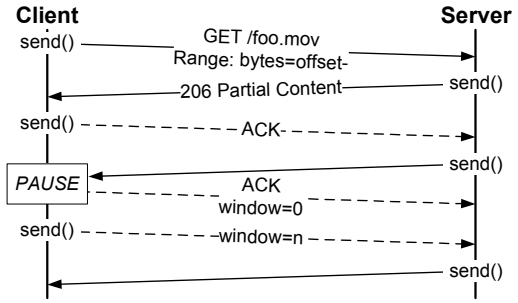

Figure 3: HTTP-Based Stream Control

In the preceding section, we have shown that RTP/UDP does not manage to utilize the 10 GbE link fully because of the resultant high CPU load when streaming at high bit rates. We now repeat the above experiment with the 8.7 Mbps data rate to compare our findings with HTTP-based services. Again, the key performance measure is how well an HTTP server scales in terms of the number of concurrent HD streams it can provide. For our experiments, we have chosen the prevalent HTTP server by Apache with the multi-processing worker module which is needed to be able to stream to more than 150 clients in parallel as each data stream needs to be handled by its own thread. To obtain a fair comparison with RTP, the `sendfile()` mechanism is

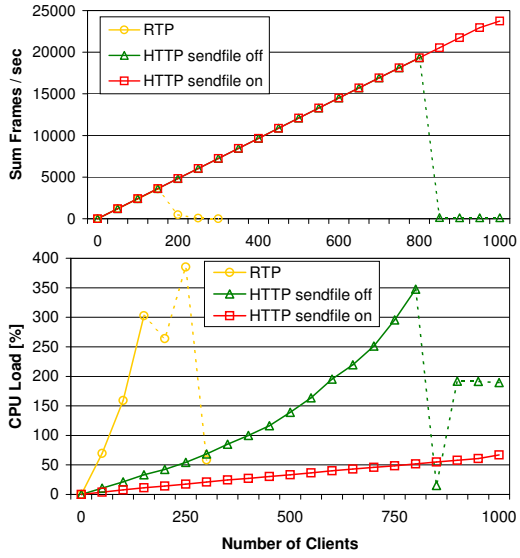

Figure 4: HTTP vs. RTP (8.7 Mbps)

disabled for this experiment (see next section for `sendfile()` tests).

As can be seen in Figure 4 the CPU limit (where all 4 CPUs are completely busy — hence 400%) is reached much later with HTTP than with RTP because of the reduced packetizing overhead and simpler connection management. However, the link still cannot be saturated: 30% of the 10 GbE bandwidth remain unused, and the server CPU load also increases exponentially, indicating bad scalability properties.

3.3 Kernel Sendfile Support

Since version 2.2, the Linux kernel offers the `sendfile()` system call. In contrast to `read()`/`write()`, it allows the data to be sent directly from the temporary kernel buffer onto the network. As a modern NIC (such the Chelsio T3) is equipped with a DMA engine, the CPU copies are avoided altogether. Furthermore, the number of context switches and of necessary system calls are reduced.

The Apache web server can directly apply this mechanism to reduce the CPU load and improve scalability. Figure 4 shows its benefit: The number of clients the server can handle is now limited by the link capacity while inducing about 70% CPU load on one core. Although the `sendfile()` approach is already a large improvement, we can do even better by using iWARP/RDMA.

4. IWARP-BASED SERVER-EFFICIENT VOD

While the aforementioned `sendfile()` mechanisms significantly improves server performance, an RDMA-based approach potentially eliminates all server CPU involvement during video data dissemination. To make efficient use of RDMA semantics, we first had to define a new video streaming application protocol, which we will introduce next.

4.1 Client-Driven Protocol

The purpose of our protocol is to reduce the server load to the minimum while fulfilling the client-side VoD requirements in terms of bandwidth and media access semantics. For that, we use the one-sided RDMA Read operations which enables a completely client-driven protocol offering efficient VCR-like media control at minimum server load.

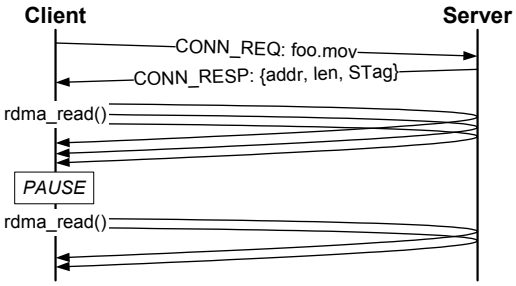

Figure 5: Client-Driven RDMA Protocol

Our communication protocol depicted in Figure 5 starts with the client opening an iWARP connection to the server. The connection setup allows a small amount of private data to be attached to both, the connection request and the connection response message. The client uses the private data of the connection request to select the movie, whereas the server attaches a buffer descriptor of the requested movie to its connection response. The buffer descriptor is a triplet consisting of the starting address of the buffer holding the movie, the length in bytes and an RDMA steering tag (STag). The client now has all the information needed to fetch the movie using RDMA Read operations. Since the entire movie is statically available in the buffer advertised by the server, the client simply needs to issue continuous RDMA Read operations from different source offsets in order to get the data required for playback. We have implemented this client-server protocol as an RDMA server application talking to a VideoLAN client module.

4.2 Server-Efficient Data Dissemination

Figure 6 shows the performance results of our protocol compared with plain HTTP and HTTP with `sendfile()` support. We have limited the experiment to 1000 clients, as this is enough to fill the 10 GbE link. The top chart indicates that our solution is capable of saturating the link by serving the required data to all clients. The chart in the middle reflects the server CPU load. During data transfer, plain HTTP induces an exponential load and HTTP with `sendfile()` a linear load, whereas our RDMA protocol induces no load at all, indicating good scalability. Independent of the local CPU performance, the RNIC itself is able to saturate the link by processing the RDMA Read requests in hardware. In addition to avoiding data copying, using an RDMA protocol with an RNIC completely avoids interrupts from the NIC since the CPU is not involved in protocol processing (see bottom chart of Figure 6). This is highly desirable as it reduces the context switching rate significantly and therefore leads to less cache pollution [11].

4.3 In-Band VCR-like Media Control

Our fully client-driven video streaming protocol is a very efficient way of providing VCR-like media control. All efforts to control the connections is done at the client side, thus freeing the server from almost all application protocol processing. The advantages of the simple protocol are three-fold:

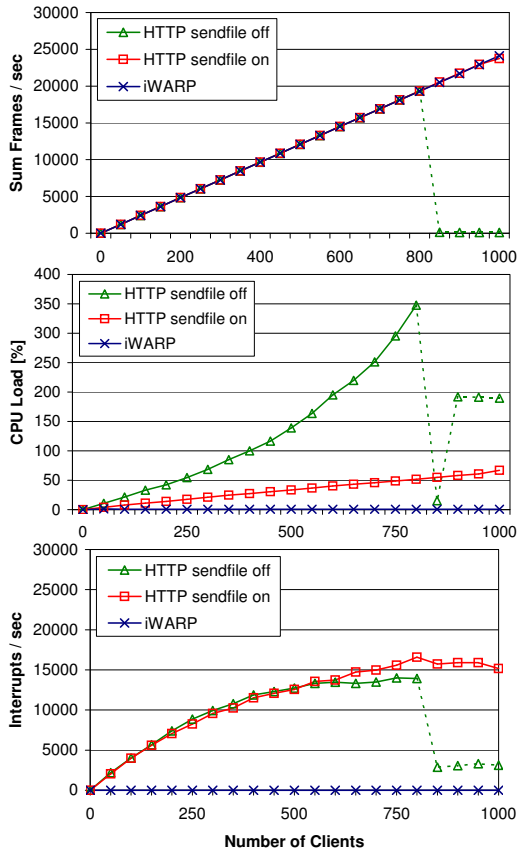

Figure 6: iWARP vs. HTTP (8.7 Mbps)

First, each client is free to read any amount from any position within the advertised buffer whenever new data for playback is needed. If several movies are available on a server, a client can watch any of them by simply switching to another buffer. The server does not need to keep track of which client is watching which movie. Thus, an expensive stream control protocol with feedback loop as well as synchronization or packetizing overhead (as in RTP) are avoided altogether, thereby reducing not only the server load but also the overhead on the network itself. This is reflected in Figure 6: The CPU load induced on the server is negligible (middle chart) while all clients receive their requested data in time (top chart).

Second, the server-side overhead is minimal because the RNIC hardware takes care of the data transfer. It processes the inbound Read Requests and sends back the requested data through corresponding Read Responses without requiring operating system intervention (see bottom chart of Figure 6). The CPU of the server is only needed for connection establishment and tear down. This cost is amortized quickly with the high data rates.

Third, in contrast to the `sendfile()` approach, copies are avoided not only on the server but also on the clients. This feature is particularly interesting as it allows us to extend our protocol to support even real-time streams.

5. DISCUSSION

A VoD server offering HD media based on RTP suffers from a high interrupt- and context switching rate as well as a CPU overhead which is exponential to the number of clients served. By using HTTP instead, the overhead can be reduced by about an order of magnitude (see Figure 7). Applying the `sendfile()` mechanism to HTTP brings the CPU load down to a linear increase with the number of clients, which is efficient enough to saturate the 10 GbE link as long as the NIC is equipped with a true DMA engine. The advantage of `sendfile()` is that it does not require changes in the software or application protocols used. Furthermore, since it is based on files, a massive storage cluster (e.g., RAID) can be used to store all the content, and it is still possible to do zero-copy data transmission on the server side.

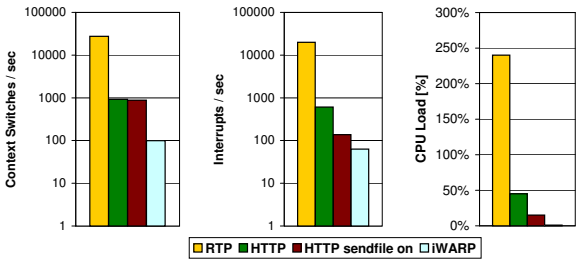

Figure 7: Direct Protocol Comparison (8.7 Mbps)

To allow a parallel serving of all clients, HTTP requires each stream to be processed by a separate thread. This results in a potential waste of system resources and necessitates an increased number of context switches. In contrast, when using RDMA, a single thread is sufficient as the connection multiplexing is performed by the RNIC. With our iWARP-based protocol, we can reduce the context switching overhead again by an order of magnitude compared with HTTP and bring the CPU load down to a small constant.

Our iWARP-based VoD solution using an RNIC is able to significantly reduce the interrupt- and the context switching rate not only on the server but also on the client. This is highly desirable as it leads to a much lower cache pollution, thus improving local application processing performance. Furthermore, a fully offloaded RDMA stack eliminates the copy overhead on both sides, which is important when streaming data at even higher rates (e.g., at several Gbps per client). The drawback of the RDMA solution is that the server must be equipped with an RNIC. In addition to that, also the clients must have RDMA stack support — but here a software RDMA stack may be sufficient (e.g., IBM SoftRDMA driver [12]). As the RDMA semantic is different from the common Socket API, major application adaptations are needed. Furthermore, the physical memory is the limiting factor for the total size of the data to be transmitted. However, this limitation can be circumvented by applying local buffer replacement strategies (e.g., a local pyramid broadcast [20]) or by attaching the server to a RamSan system [19], which offers up to several hundred gigabytes of DDR memory accessible through RDMA. An important advantage of using RDMA is the possibility to combine client-server control type interaction with the data transfer operation itself. By issuing RDMA Reads at appropriate offsets, the client is able to seek through the data set without frequent tear down and re-establishment of the data channel. The server application is not even involved when the client changes the movie playback position. Using

a Socket-based approach, each seek would close the current TCP stream on both sides and re-open the media with the new offset. Another strong server scalability advantage of the RDMA approach is the complete avoidance of a dedicated control channel between the server and each client, which is otherwise typically implemented by just another peer-to-peer Socket connection.

Besides VoD, the other popular media dissemination scenario is live streaming, where the video content is not prerecorded but generated in real time, for example by a video camera that continuously writes its output to local memory. The key difference is that in a VoD environment the client is free to choose when to fetch the next part of the video since it does not change on the server. Live streaming on the other hand is driven by the media source at the server, and the media buffer is continuously overwritten. In that context, a high server-side CPU availability is desirable since the data typically is processed (e.g., encoding/compression) before it is transmitted over the network. The `sendfile()` approach cannot directly be applied to this scenario as the data would have to be written to a file first. With our iWARP-based VoD protocol on the other hand, we can provide an efficient zero-copy solution for live streams. For such an extension of the protocol, we look at live streaming as a special case of VoD where the user does not interact with the stream apart from starting and stopping it. The server's video data production must now be synchronized with client data consumption (RDMA Read): The server may do that by sending periodic notification messages. Not sending that data itself but information about data availability has a number of advantages from a server perspective. As these notifications are sent to all clients, each client can choose the stream it currently wants to receive without inducing any coordination or tracking overhead at the server. The amount of link bandwidth wasted is very small as the size of the notification messages sent is negligible compared to the payload of the stream. A push-based scheme, on the other hand, would require the server to keep track of which client is receiving which streams and transmit the payload data accordingly. Sending all streams to all clients using unicast is clearly not an option.

6. CONCLUSION

We have analyzed and shown by experiment why server-side copy avoidance is key to achieving good scalability when offering HD media content from a single server to many clients. By proposing an iWARP-based application protocol, we have shown how an efficient VRC-like media control can be implemented for video-on-demand as well as real-time streaming services and demonstrated its significant performance improvements by experiment. Finally we have highlighted its advantages and tradeoffs as compared to the `sendfile()` zero-copy mechanism offered by the Linux kernel.

7. REFERENCES

[1] Apple Inc. Darwin Streaming Server. `http://developer.apple.com/opensource/server/streaming/`.

[2] D. Dalessandro and P. Wyckoff. Accelerating web protocols using RDMA. In *Proceedings of the 6th IEEE International Symposium on Network Computingand Applications*, 2007.

[3] Ecole Centrale Paris. VLC media player. `http://www.videolan.org`.

[4] K. Fall and J. Pasquale. Exploiting in-kernel data paths to improve I/O throughput and CPU availability. In *Proceedings of the Winter 1993 USENIX Conference*, pages 327–333, 1993.

[5] K. Fall and J. Pasquale. Improving continuous-media playback performance with in-kernel data paths. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pages 100–109, 1994.

[6] P. Halvorsen, E. Jorde, K.-A. Skevik, V. Goebel, and T. Plagemann. Performance tradeoffs for static allocation of zero-copy buffers. In *Proceedings of the 28th Euromicro Conference*, 2002.

[7] J. Hilland, P. Culley, J. Pinkerton, and R. Recio. RDMA Protocol Verbs Specification, Version 1.0. `http://www.rdmaconsortium.org/home/draft-hilland-iwarp-verbs-v1.0-RDMAC.pdf`.

[8] InfiniBand Trade Association. InfiniBand architecture specification. `http://www.infinibandta.org`.

[9] J. Levon. OProfile - A System Profiler for Linux. `http://oprofile.sourceforge.net`.

[10] Miniwatts Marketing Group. World internet stats. `http://www.internetworldstats.com/stats.htm`.

[11] J. C. Mogul and A. Borg. The effect of context switches on cache performance. In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 75–84, 1991.

[12] F. Neeser, B. Metzler, and P. W. Frey. SoftRDMA. `http://www.zurich.ibm.com/sys/servers/rdma_soft.html`.

[13] T. P. Nguyen and A. Zakhor. Distributed video streaming over internet. In *Proceedings of Multimedia Computing and Networking*, pages 186–195, 2002.

[14] OpenFabrics Alliance. OpenFabrics Enterprise Distribution. `http://www.openfabrics.org/`.

[15] T. Plagemann, V. Goebel, P. Halvorsen, and O. Anshus. Operating system support for multimedia systems. *The Computer Communications Journal*, 23:267–289, 2000.

[16] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia. A Remote Direct Memory Access Protocol Specification, 2007.

[17] V. Saxena. Bandwidth drivers for 100 G Ethernet. `http://www.ieee802.org/3/hssg/public/jan07/Saxena_01_0107.pdf`.

[18] K. Stuhlmüller, N. Färber, M. Link, and B. Girod. Analysis of video transmission over lossy channels. *IEEE Journal on Selected Areas in Communications*, 18:1012–1032, 2000.

[19] Texas Memory Systems. Ramsan-5000. `http://www.superssd.com/products/ramsan-5000/`.

[20] S. R. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4:197–208, 1996.

[21] D. Wu, Y. T. Hou, W. Zhu, Y. qin Zhang, and J. M. Peha. Streaming video over the internet: Approaches and directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11:282–300, 2001.