iWARP-based high-definition media dissemination

Master Thesis

Author(s):
Hasler, Andreas

Publication date:
2009

Permanent link:
https://doi.org/10.3929/ethz-a-005772969

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master Thesis

iWARP-based High-Definition Media Dissemination

Andreas Hasler
ahasler@alumni.ethz.ch

Systems Group
Department of Computer Science
Swiss Federal Institute of Technology (ETH)

Systems Department
IBM Zurich Research Laboratory

Supervisor: Prof. Dr. Gustavo Alonso
Thesis advisor: Philip Frey

March 13th, 2009

Abstract

In recent years, Internet usage has changed dramatically. The bandwidth of Internet connections offered by Internet Service Providers (ISPs) as well as the number of households with a broadband connection has grown rapidly. Based on this development, large ISPs already foresee high-definition (HD) media with high bit rates becoming the predominant format for media streaming in the near future. Compared to standard definition this causes about a 10 times higher bit rate. Additionally, there is a new trend in disseminating the content: instead of streaming it by broadcast after a fixed schedule, it is expected that in the near future the more flexible video on demand, which requires one separate unicast stream per user, will be the predominant way. As a result, the aggregate throughput on the server side will increase dramatically and will challenge the server infrastructure. Our experiments show that conventional stream dissemination methods such as RTP over UDP or HTTP over TCP result in high server load due to excessive local data copy operations, context switching and interrupt processing overhead.

This work presents a new approach based on zero-copy protocol stack implementations in software as well as using dedicated RDMA hardware. Performance experiments indicate that these optimizations allow servers to scale better and remove most of the overheads caused by current approaches.

Kurzfassung

In den letzten Jahren hat sich die Nutzung des Internets stark verändert. Sowohl die Anzahl Haushalte mit einem Breitbandinternetanschluss als auch die Bandbreite, welche die Internet Service Provider (ISP) ihren Kunden anbieten, ist stark gewachsen. Aufgrund dieser Entwicklung gehen grosse ISPs davon aus, dass High-Definition (HD) Media mit hoher Bitrate in naher Zukunft das vorherrschende Format für Media-Streaming sein wird. Verglichen mit Standard-Definition (SD) führt dies zu einer ca. 10-fach höheren Bitrate. Zusätzlich gibt es einen neuen Trend in der Art wie Inhalte verbreitet werden: Anstatt sie per Broadcast nach einem festen Zeitplan zu senden, wird erwartet, dass schon bald das flexiblere Video on Demand, welches einen separaten Unicast-Stream pro Nutzer benötigt, die beliebtere Variante sein wird. Die Folge wird eine enorme Erhöhung des gesamten Datendurchsatzes auf Server-Seite sein, was eine ernst zu nehmende Herausforderung für die Serverinfrastruktur darstellt. Unsere Experimente zeigen, dass konventionelle Streaming-Methoden wie RTP über UDP oder HTTP über TCP hohe Serverlast, hervorgerufen durch mehrfaches redundantes Kopieren von Daten, Context Switches und Interrupt-Processing Overhead, verursachen.

Diese Arbeit präsentiert einen neuen Ansatz basierend auf zerocopy Protokoll-Stack Implementationen in Software und dedizierter Hardware. Performance Tests zeigen, dass mit diesen Optimierungen die Server besser skalieren und die meisten Overheads von heutigen Systemen eliminiert werden.

Contents

1 Introduction

1.1 Motivation

Since 2002, the number of households accessing the Internet through broadband connections has increased by roughly 7 millions every year (USA only). By December 2008, about 23.5% of the world's population was connected to the Internet. Not only the number of users increases rapidly but also the bandwidth offered by Internet Service Providers (ISPs) [34]. Driven by this development the bit rate of video streams is constantly increasing. According to Comcast [39], the largest cable television company and second largest ISP in the United States, the next step in the evolution of the Internet will be a shift from standard-definition (SD) towards high-definition (HD) media content with an improved media quality which requires about 10 times higher data rates.

Furthermore, Comcast predicts a new trend in video dissemination from broadcast to unicast requiring one stream per user instead of one stream per video. Various video on demand platforms providing a wide range of videos have become popular. E.g. YouTube which provides short clips with rather low bit rate or Hulu.com providing movies in higher resolutions. Unicast streaming as well as video-on-demand over the Internet is typically offered either over the connectionless UDP transport with the Real-time Transport Protocol (RTP [40]) on top or over the widely used Hypertext Transport Protocol (HTTP [8]) which is based on the connection-oriented TCP transport. The TCP protocol needs several data copies on its way through the protocol stack from the application memory until it is finally sent over the network. Experiments of a research group from the Intel Corporation with 1 Gbps Ethernet links presented in [16] suggest that the 1 GHz / Gbps rule of thumb is still valid, which means that 1bps of network link requires 1Hz of CPU processing. Our experiments show that a 10 Gbps link can be saturated with TCP only under high CPU load. The bandwidth can not be fully saturated when the standard system settings are used.

Therefore, it is desirable to have a more efficient way to stream HD media. The RDMA protocol is especially designed for minimal demands on memory bus bandwidth and CPU processing overhead during data transfers. iWARP is a standardized protocol stack of the RDMA protocol for Ethernet. RDMA not only solves performance problems, but it also offers one-sided operations allowing for the design of a communication model with a fundamentally different new semantic. This work will present a new approach based on zero-copy protocol stack implementations in software as well as using dedicated RDMA hardware. Performance experiments indicate that these optimizations allow servers to scale better and remove most of the overheads caused by current approaches.

1.2 Goal and Scope

The goal of this master thesis is to answer the following questions: Is iWARP/RDMA suitable for implementing a High Definition Video on Demand service? And how does an iWARP-based system scale compared to current systems?
It analyzes the suitability of iWARP as a transport for High Definition VoD using unicast. High Definition videos pose serious requirements to traditional streaming systems:
First, the data throughput must be large enough to serve the clients with a massively increased bit rate compared to standard definition streaming. Second low latency and high responsiveness must be supported for convenient media control.
For this reason, the work focuses on analyzing the existing streaming systems, comparing them with detailed performance measurements and exploring how an iWARP enabled streaming system can be implemented. Subsequently, a fully functional iWARP-based streaming system offering VCR-like media control will be compared to traditional systems. It is assumed that the video data is already encoded in the memory and that there is either enough memory available to store all video data or an appropriate caching algorithm is used to provide the required data.

1.3 Structure of the Thesis

First, some background information about RDMA, zerocopy mechanisms and video streaming related subjects is given in chapter 2. Related work from other research projects is summarized in chapter 3. Next, in chapter 4, a detailed evaluation based on experimental results of the performance of traditional streaming systems is given and it is shown why they are inadequate for HD VoD.
Based on these findings and the RDMA background, the benefits RDMA offers are analyzed and a RDMA based protocol is proposed in chapter 5. In the same chapter the implementation of the protocol is compared through extensive experiments to the previously discussed systems. Finally the 6th chapter gives an outlook about what is to come and discusses future work.

2 Background

2.1 RDMA

In order to understand the new possibilities offered by RDMA, we give a short introduction to the basics. The main purpose of RDMA is to address the issue of network I/O in high speed networks which causes significant overhead of host processing. This is referred to as the I/O bottleneck. As we will see later on in the performance measurements, it turns out that not only local data copies, but also TCP/IP processing consume a considerable amount of system resources when large amounts of data are transmitted (an estimate can be seen in Figure 2.1). Looking at the procedures involved in receiving and sending data with TCP the high processing overhead can be explained as follows: When data is received with TCP, first, it is copied from the NIC to the socket buffer pool via DMA. From there it is copied again within the kernel space with the CPU while processing the TCP/IP stack. Finally, it is copied into the application memory (user space) also via the CPU. After all, there are three copies involved and two of them are done by the CPU.

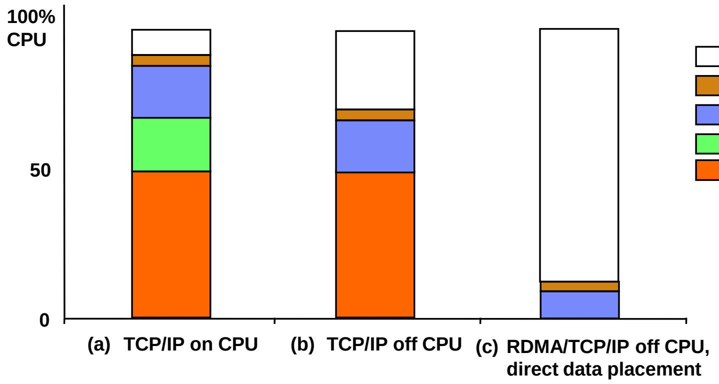

Figure 2.1: I/O Bottleneck

As depicted in Figure 2.2 sending data stored on the disk involves even more overhead: The data is copied from the disk through a temporary kernel buffer into a user buffer; then it is copied back into another kernel buffer; and finally it is copied to the network interface card (NIC). Overall the process involves 2 DMA copies, 2 CPU copies and several context switches. With RDMA in contrast data on the disk must only be copied once into the main memory. Afterwards, RDMA allows fetching data directly from local memory and placing it into the application memory of a remote host, without involving the operating system. This

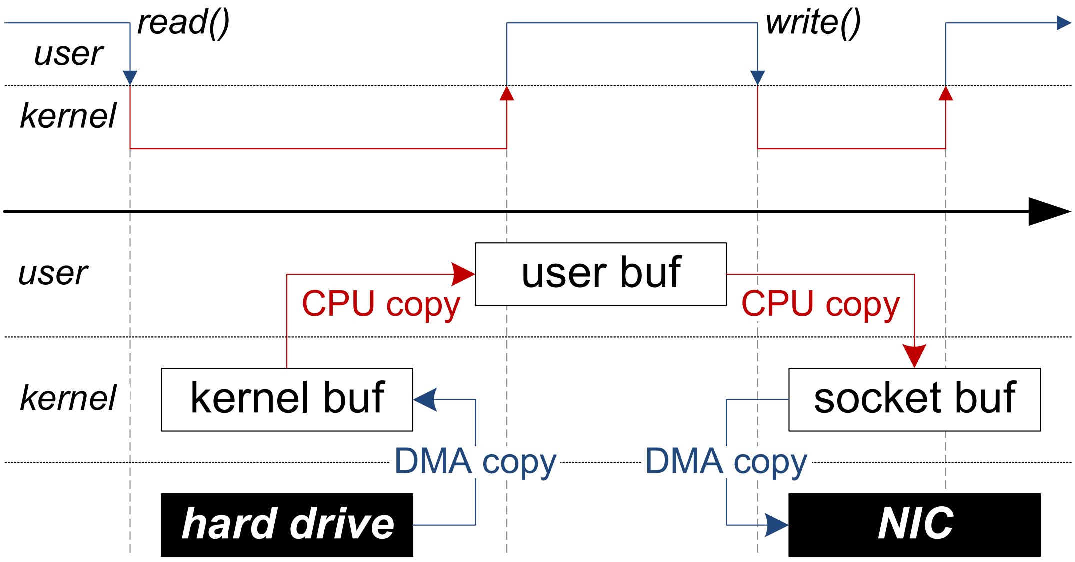

Figure 2.2: I/O Bottleneck - TCP copies

is typically performed by a specialized RDMA enabled NIC (RNIC).
With this mechanism, the CPU load, the end-to-end latency and the memory bus load is reduced significantly due to the elimination of redundant copy operations. Especially with a network bandwidth of 10 Gbit/s this can result in considerable performance enhancements as shown in [17].

2.1.1 RDMA Operations

Besides mechanisms for data transmission with less CPU load, RDMA gives also a new communication semantic. Additional to the RDMA Send and Receive which have only little differences to the traditional `send()` / `receive()` operations, it provides the two so called one-sided operations RDMA Write and RDMA Read. Send and Receive are similar to the Socket API, but allow for multiple outstanding operations. The one-sided operations restrict application involvement to the side issuing the Write or Read operation. At the remote host, the data transfer is handled completely by the RNIC, fetching or placing the data with Direct Memory Access (DMA) and thus without involving the OS or the application. As a result, the data transfer is executed asynchronously which is key to enhance application performance. A further difference between the RDMA operations and the traditional operations is that RDMA does not create a continuous data stream. With TCP you can simply read an arbitrary amount of data from the socket and process it as long as the server keeps sending data. With RDMA the Receive operations must know how big the incoming data chunk will be at most. Otherwise, the connection will terminate with an error. RDMA Read/Write do not allow a continuous stream either. The reading/writing host has to specify a read/write size. The data does not necessarily arrive in order at the data sink. This is why it should wait until the whole operation is completed before it begins to read the data from the memory. This means you either have to issue many data transmissions of small size to simulate a continuous stream, which could defeat the performance advantage of RDMA or you issue few bigger data transmissions which could pose a problem for a streaming solution. This problem will be discussed in detail later on in section 5.2.3.

2.1.2 Memory Management

To use RDMA and profit from its performance increase, a RDMA enabled Network Interface Card is required. It is accessed over the RNIC Interface (RI), specified in [20]. The RNIC has to be able to read data directly out of the application memory. A special mechanism is required to specify areas in the application memory which an RNIC can access: The memory regions (MR). A MR describes a set of locations in the main memory represented by a Physical Buffer List (PBL) which has been registered at the RNIC. The MR is identified by a STag, a base Tagged Offset (TO) and a length. It can be assigned different access rights such as local read/write and remote read/write. Registered MRs are pinned into the main memory which prevents the OS from swapping it out on the disc. This is necessary to have the data instantaneously available for the RNIC and to prevent data corruptions due to reading a memory location which is currently swapped out.

2.1.3 Queue Pairs

In order to interact with a RDMA Stream, a queue pair is needed. It consists of a send and a receive work queue and a posting mechanism to submit RDMA operations to the RI. The RDMA Stream can only be accessed over the QP. The send and receive queue store Work Queue Elements which describe RDMA operations as introduced in 2.1.1. WQEs can be posted by submitting Work Requests to the QP. Before data transmission is possible the QP must be set with the appropriate connection setup (described in 2.1.6) to the RTS state (ready to send). If an error is generated the QP will terminate the connection and go back to the idle state.

2.1.4 Protection Domain

Since a data transfer coming from a QP may only be placed into a memory region associated with the QP, it is necessary to have a mechanism to track these associations. The Protection Domain (PD) is dedicated for this purpose. When creating a memory region or QP a Protection Domain Identifier (PD ID) must be specified. When a data transfer takes place, the RNIC can validate the PD ID of the memory region, which the Steering Tag (STag) refers to against the PD ID of the QP.

2.1.5 Completion Queue

The purpose of the completion queue (CQ) is to indicate the status of posted work requests upon completion. A work request can either be signaled or unsignaled. Signaled work requests create upon completion a work completion (WC), which contains information about the status (e.g. success, error) of the completed WR. The completion queue consists of entries to hold this WCs. When a WC is added to the CQ an event can be triggered to inform the application.

2.1.6 Connection Management

In order to initialize a RDMA connection the transport connection has to be established first. The passive host acting as server has to create an event channel and a connection ID. Afterwards, the RDMA ID has to be bound to the source address, before it can listen for incoming connections. After having received an inbound connection request, the server creates a PD and a completion channel for completed work requests. After this, the CQ and QP for the connection can be created. At this point, the RDMA connection can be accepted. It is important that the client cannot send a connection request before the server is in this state. Any connection request would result in an error on the client side.

The active host taking the role of the client also has to create an event channel and a connection ID. Afterwards, the server address and route must be resolved. Finally, the PD, CQ and QP have to be created. At this point the client can connect to the server.

Contrary to socket communication the active host (client) must wait until it received at least one RDMA message from the passive host. Otherwise, the host could go into the error state. Any error results in a termination of the connection. It is also important to notice that posted RDMA work requests must be completed and cannot be discarded without a transition to the error state which leads again to the tear down of the connection. These restrictions will become important while designing the communication protocol later on.

The RDMA connection setup is much more complex than setting up a traditional TCP connection and thus takes more time.

2.1.7 iWARP

The RDMA protocol to be used in Ethernet networks is provided by the Internet Engineering Task Force (IETF) through the Internet Wide Area RDMA Protocol (iWARP). As can be seen in Figure 2.3 the iWARP stack contains 3 protocols: RDMAP over DDP on top of MPA/TCP or on top of SCTP. An important component is the Data Direct Placement Protocol (DDP) [33], which provides information to place incoming data directly into the main memory. On top of DDP there is the RDMA protocol (RDMAP) [37] which enables the RDMA Read and Write operations. For the data transmission itself usually the MPA (Message PDU Aligned) [11] in combination with the Transport Control Protocol (TCP) is used. It is also possible to replace MPA/TCP with Stream Control Transmission Protocol (SCTP) [31]. As stated in [38]

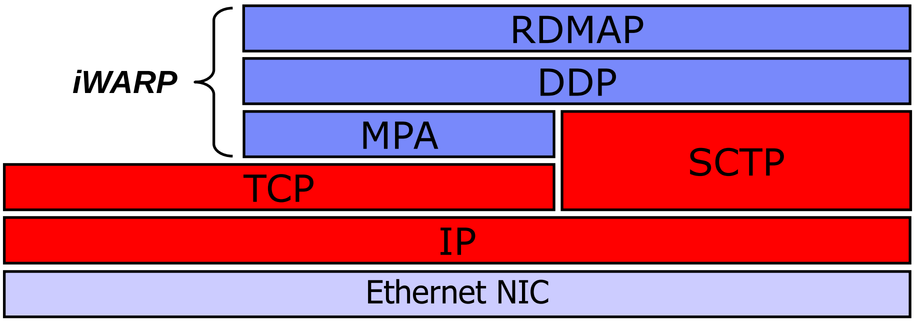

Figure 2.3: The iWARP protocol stack

SCTP is the better architectural choice since SCTP has frames and RDMA requires frames, but TCP is byte oriented. However, TCP is more widely deployed and RDMA protocols can be added to TOE NICs more easily.
Several implementations of the iWARP protocol are available for Microsoft Windows and Linux operating systems. An implementation for Linux offered by the OpenFabrics Enterprise Distribution (OFED) [3] is used in this thesis.

2.2 Sendfile

Besides RDMA there is another way to reduce the data copies and CPU load on the sending host: The Linux system call `sendfile()`. It is offered by the Linux kernel since version 2.1. In contrast to `read()`/`write()`, it allows the data to be sent directly from the temporary kernel buffer onto the network. Since a modern NIC is equipped with a DMA engine (like the Chelsio T3 adapter which is used later on in the experiments), the CPU copies are avoided altogether. Furthermore, the number of context switches and necessary system calls are reduced. The Apache HTTP web server can directly apply this mechanism to reduce the CPU load and improve scalability. Figure 2.4 shows the procedures involved in a `sendfile()` call, assuming the NIC has got a DMA engine (which is realistic for 10 Gbps NICs). First, the content of the file is copied via DMA from the hard disk into a kernel buffer. Descriptors pointing to the kernel buffers containing the data to be sent are appended to the socket buffer. Another DMA copy writes the data into the kernel buffer of the protocol engine which finally sends the data over the network. Although still two copies are involved none of them is executed by the CPU.

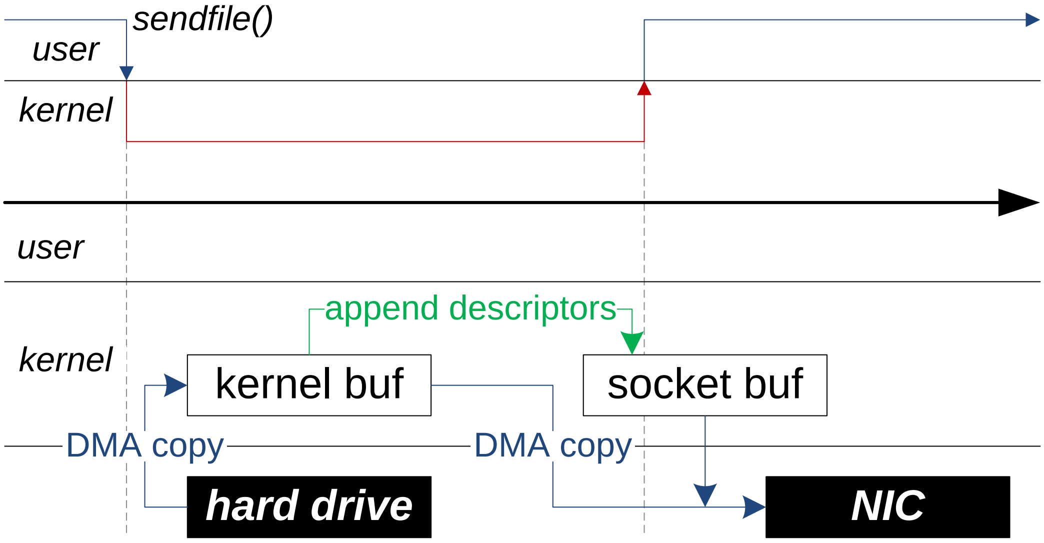

Figure 2.4: Sendfile zerocopy mechanism

2.3 Network Processing and NAPI

To understand better the results presented later, some information about network processing is helpful to interpret them correctly.

[25] describes the Linux interrupt handling mechanism as follows: When the network card receives data, an interrupt is generated. The kernel asynchronously executes an interrupt handler. In Linux kernel 2.4 the handler pulls the packet descriptors and enqueues them in a backlog queue. Afterwards, a soft interrupt (softirq) is raised and the execution of the remaining processing is done at the next available opportunity. The softirq handler dequeues the packets from the backlog and takes appropriate action.

With the Linux kernel 2.6, the one used in the experiments presented later, a new mechanism is introduced which aims at reducing the latency and packet/interrupt ratio. It eliminates the backlog queue. Instead, a reference to the device is put into a poll-list attached to the interrupted CPU, a softirq is scheduled and the interrupt reception disabled. The softirq handler polls all devices registered in the poll-list to get the packets until a quota is reached. If that happens and the device has still packets to offer, it is put at the end of the poll-list. If the device has no more packets it is taken off the poll-list and allowed to interrupt again. This leads to an interrupt driven system under low load, when the kernel has time to process the packets until the next arrive. Under heavy load devices are polled and interrupts disabled. As shown later with the experiments this can lead to an unintuitive development of the interrupt rate where fewer interrupts are measured although the network load is increasing.

2.4 Codecs

To be able to stream a media file, an appropriate codec is needed. Not all of them are equally suitable for HD video streaming. Since this work does not focus on codec specific details, it is not important to look at codec details. However an appropriate codec has to be chosen to obtain meaningful results from the experiments. In this work Quicktime H.264/AVC codec standard implementation in a *mov*-container will be used. As described in [42] H.264 is an advancement of the well known MPEG2 (which can also be used for SD and HD encoding) and H.263 video compression standards aiming at higher coding efficiency. H.264 codecs are widely used for streaming High Definition Media over the Internet and it is a mandatory codec for Blu-Ray Players [9]. Videos compressed with H.264 can easily be streamed with the prevalent media servers. Most of the streaming clients are able to decode it. With its different profiles it supports a wide variety of bit rates, which is important to analyze the impact of this factor on the streaming performance. It has a maximal data rate of 10 Mbps for 420p and ”Baseline Profile” with 30 fps (frames per second) and up to 50 Mbps for 1080p and ”High Profile” with 60 fps (explained in [42]). The average rate is usually about a factor of 2-3 smaller. Other codecs for HD media like On2’s VP8 [30] focus on higher compression resulting in lower image quality with a data rate of usually about 1-5 Mbps for a 1080p movie. BBC’s Dirac [7] and Microsoft’s VC-1 [21] video compression would also allow high bit rate streaming in HD resolution. Because the H.264 video compression standard is more established [32] it is considered to be most appropriate for the experiments with the predominant media streaming systems.

2.5 Streaming Protocols

Typically Video on Demand uses either the connectionless User Datagram Protocol (UDP) as transport protocol in combination with the Real Time Transport Protocol (RTP) or the widely used Hypertext Transport Protocol (HTTP) on top of the connection-oriented TCP transport.

2.5.1 RTSP/RTP on top of UDP

RTP defines a standardized packet format for delivering audio and video over the Internet. Most commonly UDP is used to transport the RTP payload. Although it is possible to have TCP as underlying protocol, it is rarely used. Since UDP is non-reliable, RTP is used in combination with the Real Time Control Protocol (RTCP). RTCP adds a feedback channel to the stream, which monitors the data transmission and carries out-of-band control information. For VoD the user has to be able to remotely control the media streaming through operations like seeking, pause/resume. This is taken care of by the Real Time Streaming Protocol (RTSP).

Since RTP was designed with the focus on streaming media and other real-time data, it seems like a good choice for VoD. With the use of UDP, unnecessary retransmissions of data are avoided. Missing parts can be skipped directly instead of having to stop the media stream and wait for the retransmissions. However it is important to notice that opposed to Live Streaming, with VoD it is often more desirable to wait a few seconds if data is missing and see the whole movie instead of skipping parts of the video in order to avoid waiting times.

2.5.2 HTTP on top of TCP

In VoD it can be desirable to have retransmissions with highly compressed media where the loss of a key frame can cause severe playback disruption. For this reason TCP may be a good choice as transport protocol. Streaming media over HTTP assures good interoperability and server efficiency (as later experiments will show). E.g. YouTube as the leading video platform in the Internet (according to a survey of comScore [10]) uses HTTP-based video dissemination. Even though HTTP was designed for document exchange which seems to be contrary to a continuous stream, it offers some convincing 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.

The play, pause and seek requests provided by RTSP to remotely control the media server can be simulated (as shown in figure 2.5) with HTTP by stopping to read data from the TCP buffer and thus delaying TCP window updates. This causes the sender to stall the stream. For short stops the stream can be cached locally. Seeking can be done by converting the time offset within the movie to a byte offset and requesting the according byte range. In the RTSP case the server will do this conversion to send the requested data to the client. In the HTTP case, the client has to do it.

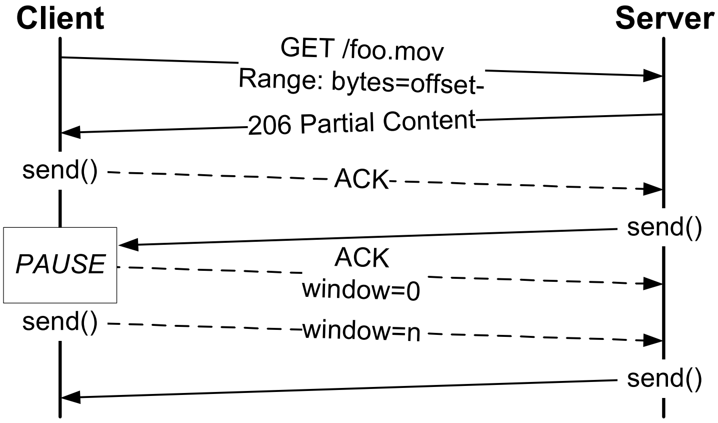

Figure 2.5: HTTP VoD

Server	Platform	RTSP/RTP/ UDP	RTSP/RTP/ TCP	HTTP
Quicktime Streaming Server	Mac	yes	yes	no[a]
Darwin Streaming Server	Linux	yes	yes	no[a]
VideoLAN	Linux/Windows/Mac	yes	no	yes[b]
Apache Web Server	Linux/Windows/Mac	no	no	yes
Adobe Flash Media Server	Linux/Windows	no[c]	no	yes
RealNetworks Helix Server	Linux	yes	yes	yes

Table 2.1: Video Servers - Supported Platforms and Protocols

[a] RTSP/RTP over TCP can be bound on port 80. Apple describes this with HTTP streaming.
[b] VLC does not support VoD using HTTP. Only one to one streaming is supported over the HTTP protocol
[c] RTMP/RTP/UDP is supported instead.

2.6 Video Streaming Servers

Today's prevalent media servers specialized in streaming media are Apples Quicktime Streaming Server for Mac OS X [6] / Darwin Streaming Server [5] for Linux (both sharing the same code base), VideoLANs VLC [1], RealNetworks Real Server / Helix Server [36]. An overview of the servers is given in 2.1

In this thesis experiments are performed on a Linux Operating system, which excludes the Quicktime Streaming Server from the tests. The Helix Server is not applicable due to its restriction to the Real Networks own video file formats (*.rm, .ra, .rv*). The Adobe Media server is restricted on its own *flv* video-format and thus not considered for the experiments. The Apache HTTP Server, which is the de facto standard, is used to test the HTTP streaming performance. The VLC server and DSS serve as media sources for RTSP/RTP video dissemination.

2.7 TCP Throughput Limitations

In the introduction it was mentioned that the 1 GHz per Gbps rule is still appropriate for 1 Gbps networks. Before we start looking at real video streaming systems, it is good to know what throughput is possible with TCP. For this reason, experiments measuring the TCP throughput between two BladeServers with 2.3 GHz quad core CPUs connected through 10 Gbit/s network are performed. More details on the test bed are provided in section 4.1.

In a first experiment 2 GB main memory are allocated on a BladeServer and continuously sent over the network to the second BladeServer. The sending host executes the `send()` operation in a loop until the whole 2 GB are transmitted. The receiving host counts the amount of data received and discards it afterwards. In the standard setting the TCP_NODELAY option is activated which means that the Nagle algorithm [27] is deactivated. If packets smaller than the TCP Maximum Segment Size (MSS) are being sent, the Nagle algorithm would send them only if all previously transmitted packets have been acknowledged. Otherwise it optimizes throughput by sending several smaller ones as one packet. The TCP_NODELAY option is deactivated to ensure valid results for throughput measurements with small packet sizes. The SO_SNDBUF is set on 28416 Bytes and the SO_RCVBUF on 87380 bytes per default. The size of the application buffers sent in the loop is varied during the experiments from 64 Bytes up to 8024 Bytes. With the default settings only a throughput of roughly 6 Gbps is possible.

In a second series of experiments the sender and receiver spawn 8 send / receive threads transmitting data in parallel to optimize for 4 cores. The experiments show the impact of changing the SO_SNDBUF and SO_RCVBUF to different values between 16 KB and 8 MB. The data is still sent from the 2 GB of allocated main memory. The impact of changing these parameters is shown in Figure 2.6. For each experiment the SO_SNDBUF and SO_RCVBUF are set to the same value (indicated in the legend). Although it is possible to send with a throughput of more then 9 Gbps if the SO_SNDBUF and SO_RCVBUF are set to 256 KB or more, the CPU is under heavy load as shown in Figure 2.7 (measured with 8192 KB SO_SNDBUF / SO_RCVBUF). The full bandwidth can only be used when 4 KB or more is sent as one application buffer.

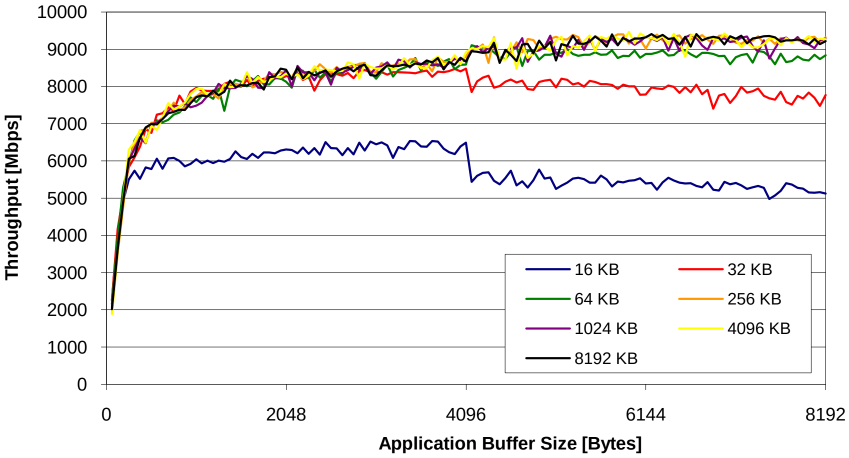

Figure 2.6: TCP Throughput - Communication Buffer Size

A further optimization of throughput can be achieved if all the data being sent is small enough

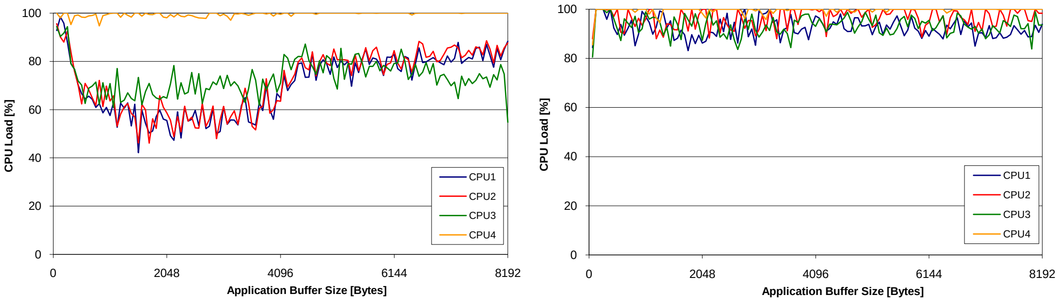

Figure 2.7: CPU Load (client left chart, server right chart)

to be cached within the L2 cache. For this reason the same experiment was executed using a 1 MB buffer which is sent repeatedly. The result is shown in the third curve of Figure 2.8. It shows the difference between sending the data from memory with standard settings (blue curve), sending it from memory with optimized settings (red curve) and sending it from L2 cache with optimized settings (green curve). Sending out of the L2 cache results again in significant performance improvements. The CPU load is reduced and the full bandwidth can already be used by sending application buffers of about 512 KB size. This measurements show clearly that TCP has a serious impact on the CPU load. Sending data which does not fit into the L2 cache uses all 4 cores to capacity on the sending side and on the receiving side 1 core is fully loaded and 3 cores are loaded 60% - 80%. Since video files are typically larger than the L2 cache, the last optimization which puts everything in the L2 cache is not realistic in a VoD scenario. Without optimizations the full bandwidth cannot be used. With optimizations the full bandwidth can be reached but only under full CPU load. Hence four times 2.3 GHz is needed to send with 10 Gbps, meaning that still about 1 Hz per bps is necessary. Concluding one can say that the 1 GHz per Gbps rule of thumb can still be used in a 10 Gbps network as a rough estimate, but it also depends heavily on the settings and the transmission pattern.

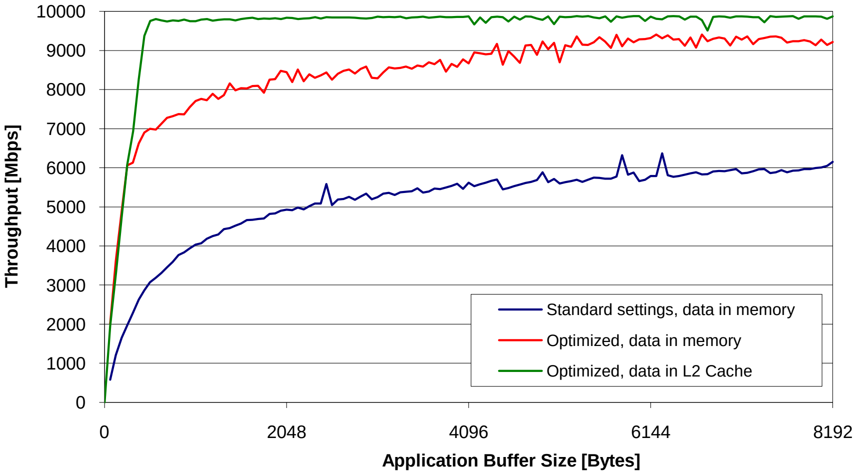

Figure 2.8: TCP Throughput - L2 Cache vs. Memory

3
Related Work

This work uses methods and discusses problems of several related research areas such as zero copy and copy avoidance mechanisms, data streaming systems (and in particular video streaming) and caching mechanisms.

3.1 Zero Copy and Transport Offloading

Already in the early 90s, Fall and Pasquale have seen the need to shortcut data paths within the operating system. In [13] they have proposed an extension to the UNIX system calls referred to as splice which moves data between two file descriptors without copying between kernel address space and user address space. Splice is able to avoid unnecessary data copies, provides asynchronous operation and supports concurrent I/O which all helps in reducing the CPU load as well as the number of context switches. Splice is closely related to the sendfile mechanism we applied. In [14], the same authors have shown the applicability of the splice system call to continuous-media playback over a UDP transport. Later, it will be shown that UDP used to be a valid choice for media transmissions at those low data rates. [19] presents a copy avoidance solution for media-on-demand servers as part of the INSTANCE project. A shared memory region is used between the hard disk 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 the purpose of their work is similar, solutions presented in it are fundamentally different. They suggest UDP as a suitable transport layer and restrict their applicability to non-live media content since the system assumes the data are not being altered on the stable storage. Furthermore the data is sent out only in fixed periodic intervals.
An acceleration of HTTP using RDMA has been proposed in [12]. An iWARP Apache module was presented which extends the HTTP protocol with RDMA support. While HTTP is push-based, this work proposes a completely new protocol which is pull-based and independent of HTTP. [26] explains why general TCP offload engines have had only little success. It describes its performance and deployment issues and discusses why transport offloading is justified with RDMA. Based on its rationale, it conjectures that RDMA might be useful for video on demand.

3.2 Streaming Systems

[24] discusses data streaming on a variety of levels not limited to network communication. The abstraction of I/O channels is used to describe the link between any pair of I/O devices within the operating system kernel. This work aims at minimizing data copies for an embeddable, real-time operating system with the ultimate goal of providing a zerocopy architecture due to its improved performance. For this purpose, a global buffer cache is suggested. [43] reviews media dissemination methods, video compression techniques and streaming protocols.

Much work has been done on optimizing data streaming for bandwidth efficiency. Among these schemes there is a mechanism called batching [15]. When the server receives multiple requests for the same video, it services them all by multicasting the video to the whole group only once. Another proposed mechanism is Staggered Broadcasting [4], which starts a new stream for a video after a fixed interval and all clients requesting the video have to wait until a new stream starts. They can cache already started streams in order to jump at another position within the movie. Pyramid broadcasting [2] is a variation of staggered broadcasting. The video is divided into different segments in multiplicatively increasing size. For each segment there is one stream which always plays it in a loop. The client must cache the streams in order to have the data from the next stream ready when the previous stream ended. The waiting time for the first stream to begin is reduced heavily with this scheme. [23] discusses the changes needed in TCP to use it for real-time network applications and describes a slightly changed TCP protocol, the TCP-RPM protocol, which provides special support for real time applications while preserving the benefits of TCP. [22] analyzes the TCP overhead. It divides the overheads into the two categories 'scale per byte' and 'scale per packet' and provides measurements for the time spent in different stages of the TCP stack processing.

3.3 Caching

The issue of loading video content from disks in a video-on-demand server into main memory is addressed in [18]. A per-device rather than a per-stream buffer allocation scheme is proposed to improve the scalability of the server in terms of number of clients served. Enough network and computing resources are assumed to be available since the proposed allocation scheme is slightly more demanding than the per-stream buffering. Our work on the other hand assumes that the data-rate at which the content can be read from disk is high enough (e.g. by using a RAID system) to saturate the network link even for frequently changing client requests.

An overview of the many aspects in which the operating system can influence the performance of multimedia applications is given in [34]. The considered aspects include resource management and quality of service guarantees such as a short response time or a minimum throughput, CPU scheduling algorithms and disk management. In the context of memory management, data-centered allocation (memory is allocated to data objects) versus user-centered allocation (memory is allocated to a process) is discussed. In the same context, data replacement and prefetching mechanisms are listed. Finally, I/O tuning is considered which is what our work focuses on in order to avoid major system bottlenecks. [35] evaluates resource sharing techniques in different VoD usage scenarios. It provides a statistical approach for cache management to reduce disk I/O and shows how design parameters can be tuned optimally.

4 Scalability of Current Systems

This chapter shows the scalability of a VoD system with the Darwin Streaming Server and VLC Server using RTSP/RTP and the Apache web server using HTTP. It describes the test setup and performance indicators. Experiments show the impact of different service parameters on the media servers.

4.1 Test Setup

The server performance is examined through a series of experiments on a 10 Gb Ethernet fabric. To avoid disk access overheads, the whole movies are preloaded into the main memory. The test bed consists of an IBM BladeCenter containing six HS21 Blade-Servers. 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 RNIC. The BladeServers are running a Fedora Linux 2.6.27 kernel with the OpenFabrics Enterprise Distribution v1.4 [3] for iWARP support. One BladeServer acts as the server (media source) and the other five are connecting to it as clients (media sink). As explained in more detail in section 2.4, the test videos are encoded with the predominant H.264 HD media codec. The following indicators are used to determine the performance of the media server: CPU load, interrupts per second, cycles per byte and quality of service experienced on the client side (frames/sec. arriving in time).
The main criterion of media dissemination efficiency is the maximum number of clients a single server is able to serve concurrently without service degradation (frame loss or late frames). There are many different scenarios possible in terms of access patterns and server/client settings. To find out which parameters have an impact on the server performance when changed, the influence of VoD service parameters such as offering different numbers of movies, movies of different length, movies with different bit rates, changing the client-side cache size, as well as jumping within the movie to another position is being investigated. The impact of a single parameter is analyzed by setting it to different values and leaving all other parameters at the default value. Table 4.1 lists the default value for each parameter, as well as the minimal and maximal tested value.

4.2 Evaluation Method

The tests are executed using the VideoLAN client version 0.9.5. To simulate hundreds of clients requesting movies from the BladeServer acting as media server, several VLC instances are executed on each of the other five BladeServers acting as clients. Since it is not

Service parameter	Default value	Min. Value	Max. Value
Number of movies	1	1	8
Movie length	56 min. (3.8 GB)	2:21 min. (160 MB)	56 min. (3.8 GB)
Bit rate	8.7 Mbps	1 Mbps	8.7 Mbps
Cache size	400 ms	100 ms	6400 ms
Jumps	no jumps	avg. after 30 sec.	avg. after 240 sec.

Table 4.1: Test parameters and values

feasible to fully decode all arriving data, it is only demuxed and the dummy decoder is activated. With this setting the incoming video stream goes through the demuxer which extracts the video and audio stream and passes them along to the dummy decoder module. The dummy decoder just performs basic sanity checks, but does not extract and reconstruct the real frame. For our tests the dummy decoder was slightly modified to count and log frames which arrive in time and frames which arrive late or not at all. The clients are started sequentially. After waiting one second in-between one more client is prompted to request a movie from the server. Because the CPU load indicated by /proc/stat is not reliable, it is measured as described in [28] with *OProfile*. Opcontrol is set to collect 1'000 samples per second of the CPU_CLK_UNHALTED events. Afterwards, the CPU load can be determined by dividing the number of samples where the CPU was busy by the CPU frequency. The CPU load is averaged over 20 seconds. The number of requests is held constant during that time, no new clients request a video and no clients stop playing their video. The interrupt rate is also measured with *OProfile*, collecting samples of the event HW_INT_RCV. *OProfile* is set to measure each 1'000th sample. Cycles per byte are calculated from the CPU load and the amount of data which all clients receive. The number reflects the cycles needed at the server to transmit a byte which the client actually receives.

4.3 Initial Performance Evaluation with Default Parameters

A first experiment tests the performance of the VLC and DSS using the RTP/RTSP protocols (Figure 4.1) and the Apache web server using the HTTP protocol (Figure 4.2). The performance indicators are measured against an increasing number of clients. In order to determine the effect of the zero-copy sendfile mechanism, the HTTP server is tested twice, with and without the sendfile option enabled. The following section explains the most important results from the measurements. The left two charts in Figure 4.1 depict the CPU load (upper chart) and interrupt rate (lower chart) measured on the server side. They both indicate that the VLC server starts having problems with more than 160 clients. At this point the CPU load hits the limit of nearly 400% (100% on each core). On the client side one can clearly correlate this limit with the sharply dropping frame rate after 160 clients as shown in the lower charts. Up to 160 clients the frame rate curve increases linearly with an increase of 24 frames/sec. per additional client, which indicates that on the client side no frames are missing or late.
The Darwin Streaming Server tests reveal a scalability of up to 110 clients. Up to this point the increase in the frame rate is also linear by 24 frames/sec. per new client. The CPU load only goes up to 250% - 300%. It seems as if there was more capacity free. We contacted the developers of DSS and tried to optimize to take advantage of the unused 100% CPU capacity.

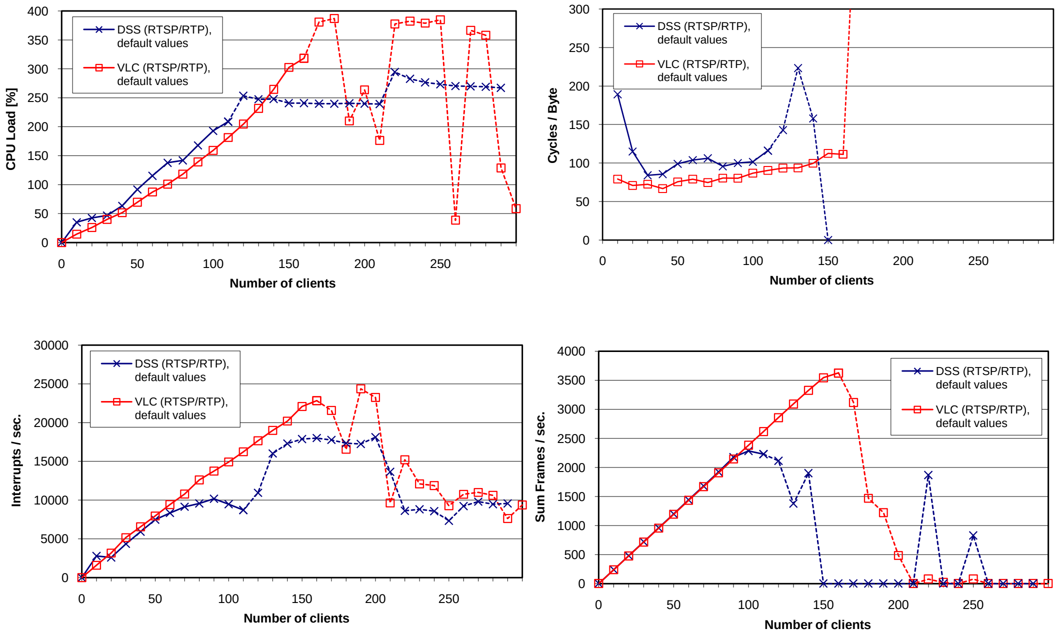

Figure 4.1: RTP results with default values

All efforts led to the result that it stopped between 250% and 300%.
The cycles per byte measurement (upper right chart) shows for the VLC and DSS a linearly increasing curve. The VLC is slightly more efficient using only 80 cycles per byte to serve 10 clients with an increase of about 2 cycles/byte per additional client whereas the DSS starts with 200 cycles per byte serving 10 clients and drops down to roughly 100 cycles per byte later on. The erratic shapes of the curves beyond the scalability limit (dashed parts of the plots) are caused by non deterministic, non reproducible behavior of the server due to the high CPU load. They indicate severe service degradation.

The HTTP results (see Figure 4.2) evidence a significantly better scalability than the RTSP/RTP results. Even without using the sendfile feature, the server scales up to 800 nodes. After this point it also hits the CPU limit and is not able to handle any more clients. Due to the behavior of the VLC clients which disconnect and try to reconnect when they do not receive the requested data in time, the HTTP server is not able to serve any client with enough frames allowing the VLC client to continue playing the video. With sendfile enabled the Apache server can support up to at least 1’000 clients with only about 70% CPU load. Because the bandwidth is getting saturated at this point, it is not possible to test beyond 1’000 nodes. As expected the sendfile option has no impact on the number of interrupts. Up to the point where they both are able to provide all clients with the full frame rate there is no significant difference between the curves of both HTTP tests. The interrupt rate stops to increase at about 15’000 interrupts/sec. One can see once more the effect of the Linux Kernel 2.6 switching under heavy load from the interrupt-based system more and more to a poll-based system. While the HTTP server with `sendfile()` enabled has a linearly increasing CPU load, resulting in a constant cycles/byte rate of about 4 cycles/byte, the HTTP server without `sendfile()` has an exponentially increasing CPU load and cycle/byte rate.

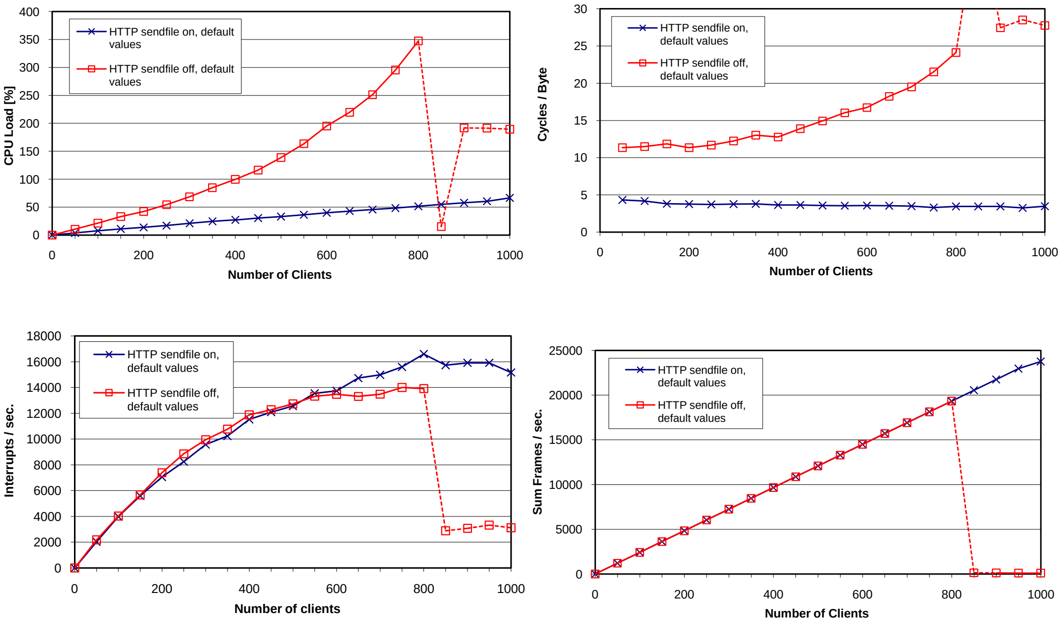

Figure 4.2: HTTP results with default values

The performance differences between the RTSP/RTP servers and the Apache HTTP server are remarkable, considering that TCP is known to have a higher overhead than UDP. Both RTSP/RTP induce exponentially increasing CPU load which leads to severe frame loss although only about 15% of the network bandwidth is used. The bigger overhead could be due to the larger protocol stack and the fact that RTSP/RTP usually uses RTCP. The main purpose of RTCP is to provide a feedback channel, which can force retransmitions of lost packets. This means that with RTSP/RTP there can be retransmissions as well. Furthermore, the HTTP server is well established and highly optimized.

4.4 Performance Critical Service Parameters

Having tested the performance using default values, this section will analyze the impact on the performance when changing different service parameters. First, the number of movies provided by the media server is varied between 1 and 8. Afterwards, it is measured what difference the movie length does make. The next section shows the difference between caching only 100ms of the video on client side and caching several seconds. Another service parameter is the bit rate of the videos offered by the server, which also indicates the difference between SD and HD movies. Finally, it is shown how a user jumping to different positions within the video can have an impact on the server performance.

4.4.1 Number of Movies

The impact of the number of offered movies is measured with all movies being requested equally often by clients. In this case the movie size is reduced to 7:03 min. (480 MB), to be able to load all movies into the main memory for the tests. Figure 4.3 shows the performance

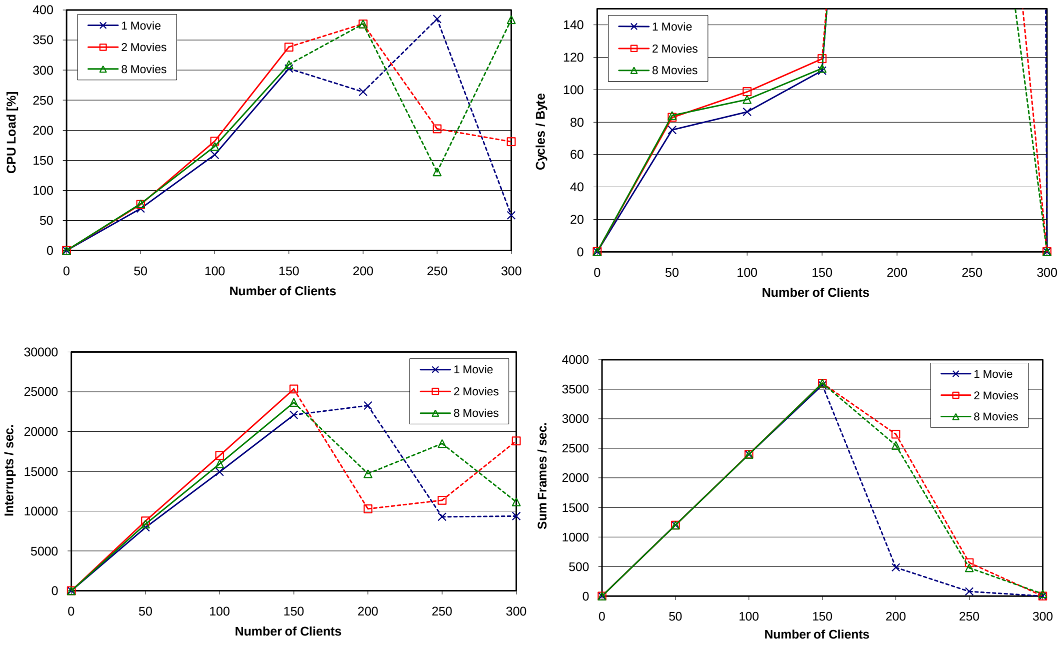

Figure 4.3: Impact of number of movies

of the VLC streaming server. There is no significant difference recognizable between any of these tests and the initial experiments with the default values. The DSS and HTTP tests reveal the same behavior as the VLC test. All performance indicators show for all movie servers no significant difference until the user limit is hit. The charts for the HTTP server and Darwin Streaming Server were omitted for simplicity.

4.4.2 Movie Length

The same behavior is observed with the movie length. Tests are executed with movie length of 2:21 min (160 MB), 7:03 min. (480 MB), 14 min. (960 MB), 28 min. (1.9 GB) and 56 min. (3.8 GB). All four performance indicators for all movie servers signify no difference in the performance as depicted in Figure 4.4 for the VLC server.

4.4.3 Size of Cached Data at Client

Also for the caching value no different server performance could be measured with different caching values. It is varied from 100ms up to 6400ms, doubling it with each measurement. Figure 4.5 illustrates the results of the experiment with the VLC server.

4.4.4 Bit Rate

For the bit rate test, the movie is encoded with the H.264 codec at 1 Mbit/s (SD), 2.5 Mbit (480p), 5.3 Mbit (720p) and 8.7 Mbit (1080p). As expected the servers scale much higher with lower bit rates. The VLC server as well as the DSS have no problem to stream the 1 Mbit and the 2.5 Mbit encoded movies to up to 300 clients. With a bit rate of 5.3 Mbit problems begin to start after 270 clients with VLC and 230 clients with the DSS. With 8.7 Mbps bit

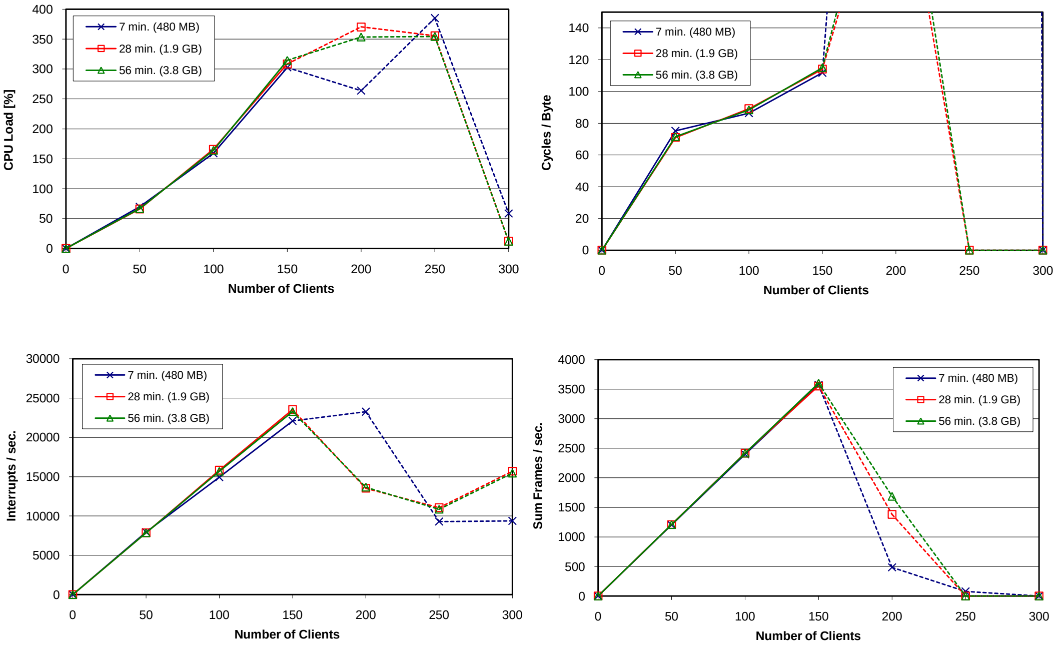

Figure 4.4: Impact of movie length

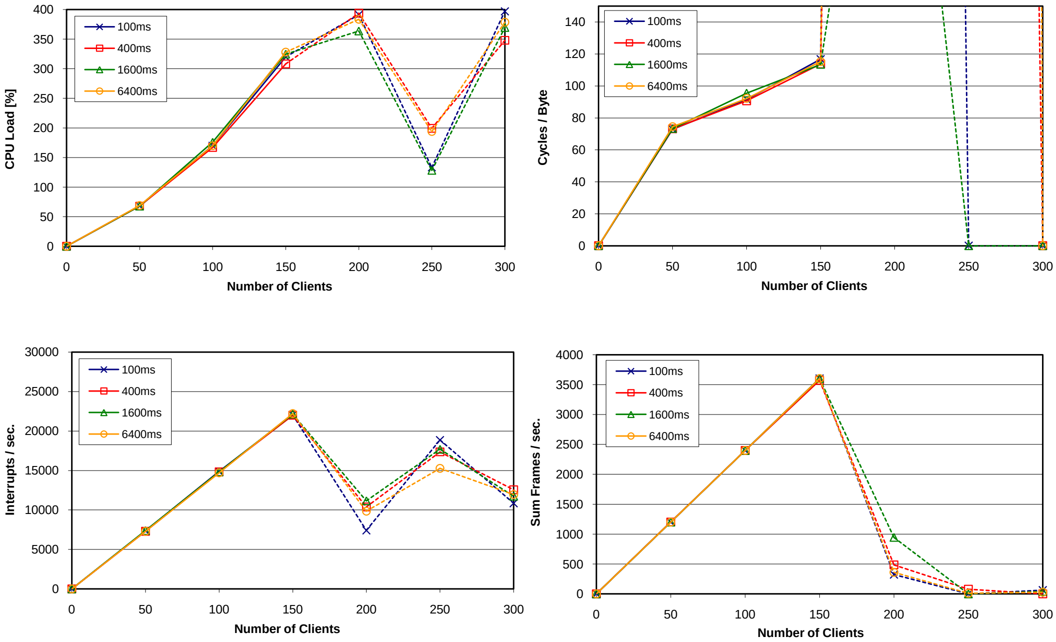

Figure 4.5: Impact of cache size

rate, the servers stop working properly already at 160 and 110 clients (Figure 4.7). In the tests with default values it was already noticed that the DSS does not make use of the full CPU capacity. The same is true for the 5.3 Mbps movie. CPU load stops increasing at about 250% load. In general the VLC server not only makes use of the full capacity, but also induces

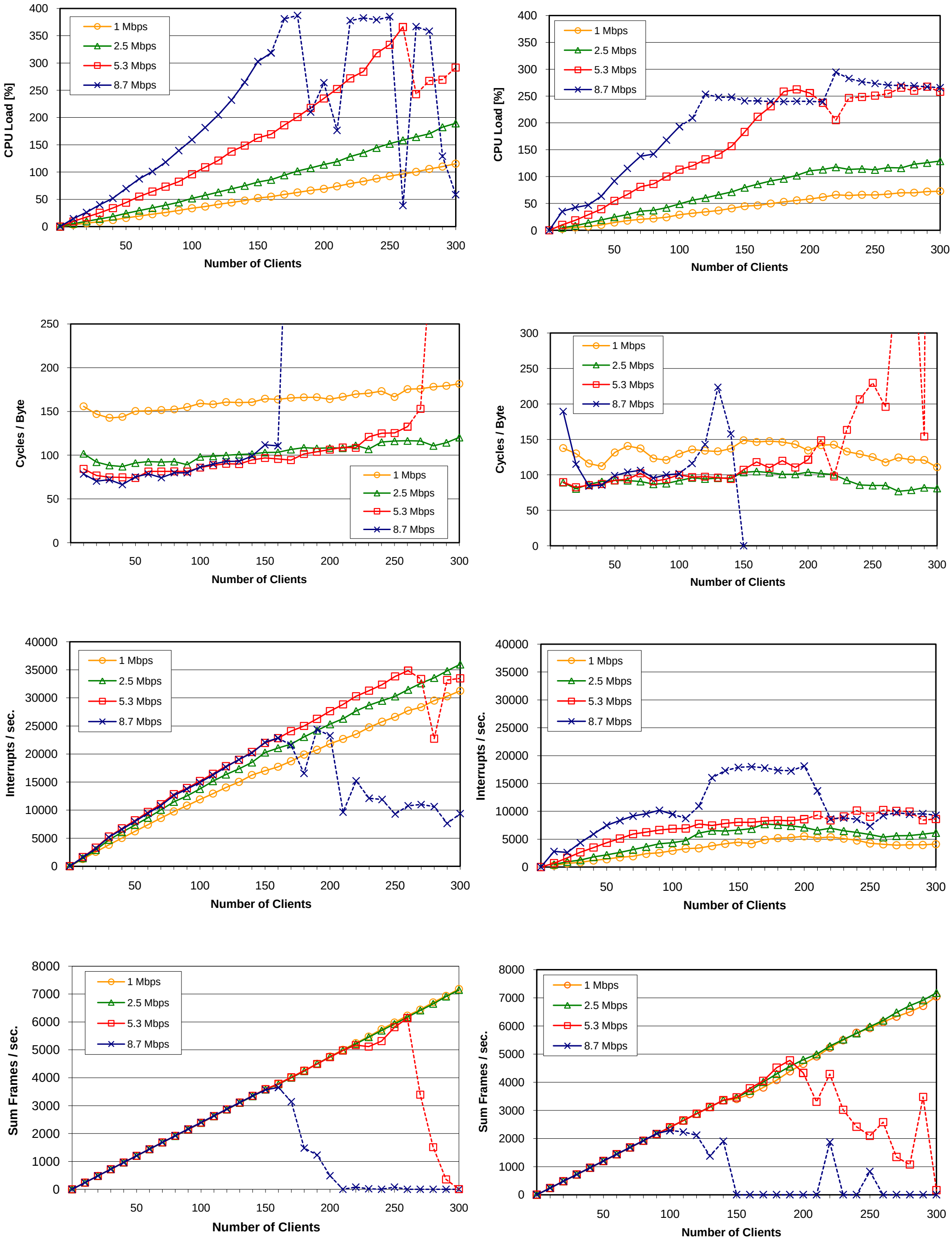

Figure 4.6: RTSP/RTP - Server Impact of bit rate (VLC left side, DSS right side)

less CPU load for the same amount of users as the DSS. The frames/sec. charts (fourth row) confirm this. They show a sharp decline in received frames on the client side in both cases where a CPU limit is hit (8.7 Mbps and 5.3 Mbps) for both servers.

Although the video with 8.7 Mbps bit rate scales worse in terms of the maximum number of

clients, the cycles/byte indicator (second row) shows a better throughput efficiency. The 8.7 Mbps and the 5.3 Mbps videos need about 80 cycles per byte, going up to 150 in case of the 1 Mbit video. This means that the worse scalability is not due to less efficient data throughput with higher bit rates, but due to the bigger data volume which has to be transferred.
The impact of different bit rates on the interrupts/sec. (third row) is not as big as on the CPU load. The savings using the 1 Mbit movie instead of the 8.7 Mbit movie are only about 25%. While the interrupt rate of the VLC server increases linearly up to 35'000 interrupts/sec., it increases only little with the DSS to about 10'000 interrupts/sec and remains constant at this rate.
In this respect the HTTP server reveals its strength in handling a high number of clients once again. Even without sendfile support it scales up to the full 1'000 clients with 5.3 Mbps and lower bit rates. Only in the case of 8.7 Mbps movie, clients begin to experience frame loss after 800 clients (fourth row). Comparing the cycles per byte curves from the HTTP server without sendfile enabled with each other, one can clearly recognize the effect of the exponential CPU load. At the beginning, the 8.7 Mbps movie is streamed most efficiently in terms of CPU cycles per byte. As more clients request a video it gets more and more inefficient. Since the other movie curves increases by a smaller factor, the 8.7 Mbps movie is the most inefficient from all bit rates with about 20 cycles / byte after 650 clients. The bit rate has most impact on the HTTP server with sendfile disabled. For the 1 Mbps movie the CPU load is constantly bellow 50% whereas the 8.7 Mbps movie is limited to 800 clients due to the CPU limitation.
Comparing the RTSP/RTP servers with the HTTP servers there is only the interrupt rate of the DSS which is on the same level as those of the HTTP servers. All other indicators show a significantly worse performance. Overall both RTSP/RTP servers have similar limits.

4.4.5 Seek Rate

The seek rate is the second factor which has an impact on the server performance for the HTTP server. In a first measurement YouTube like behavior is simulated by letting the clients jump to another position in the movie after an average waiting time of 30 seconds. The RTSP/RTP servers show no different behavior using different seek rates (Figure 4.8). This is a strength of the RTP servers compared to HTTP. Due to the waiting times between jumps where no frames are transmitted they scale slightly better. However the HTTP server can handle at least twice as much clients despite its performance losses. Looking at the HTTP server without the sendfile mechanism shows that the limit, where clients experience heavy frame loss, is reached after 400 clients (fourth row of charts), which is half of the limit without any jumps. Increasing the average waiting time between two jumps to 60 seconds, improves the server performance up to 700 clients.
The clients have to wait after a jump until they can continue to play the movie, which reduces the frame rate as depicted in the lower charts. This is not due to lost frames, but to a lower demand of frames from client side. Because of the resulting reduction of server load, the server scales up to 1'000 clients with an average waiting time between two jumps of 120 and 240 seconds. The charts in the first two rows indicating the CPU load and cycles/byte confirm this by showing double the CPU load with one jump every 30 seconds compared to one jump every 240 seconds. With the sendfile mechanism turned on and two jumps per minute the server is still able to support the full 1'000 clients, but the CPU load is a factor 3

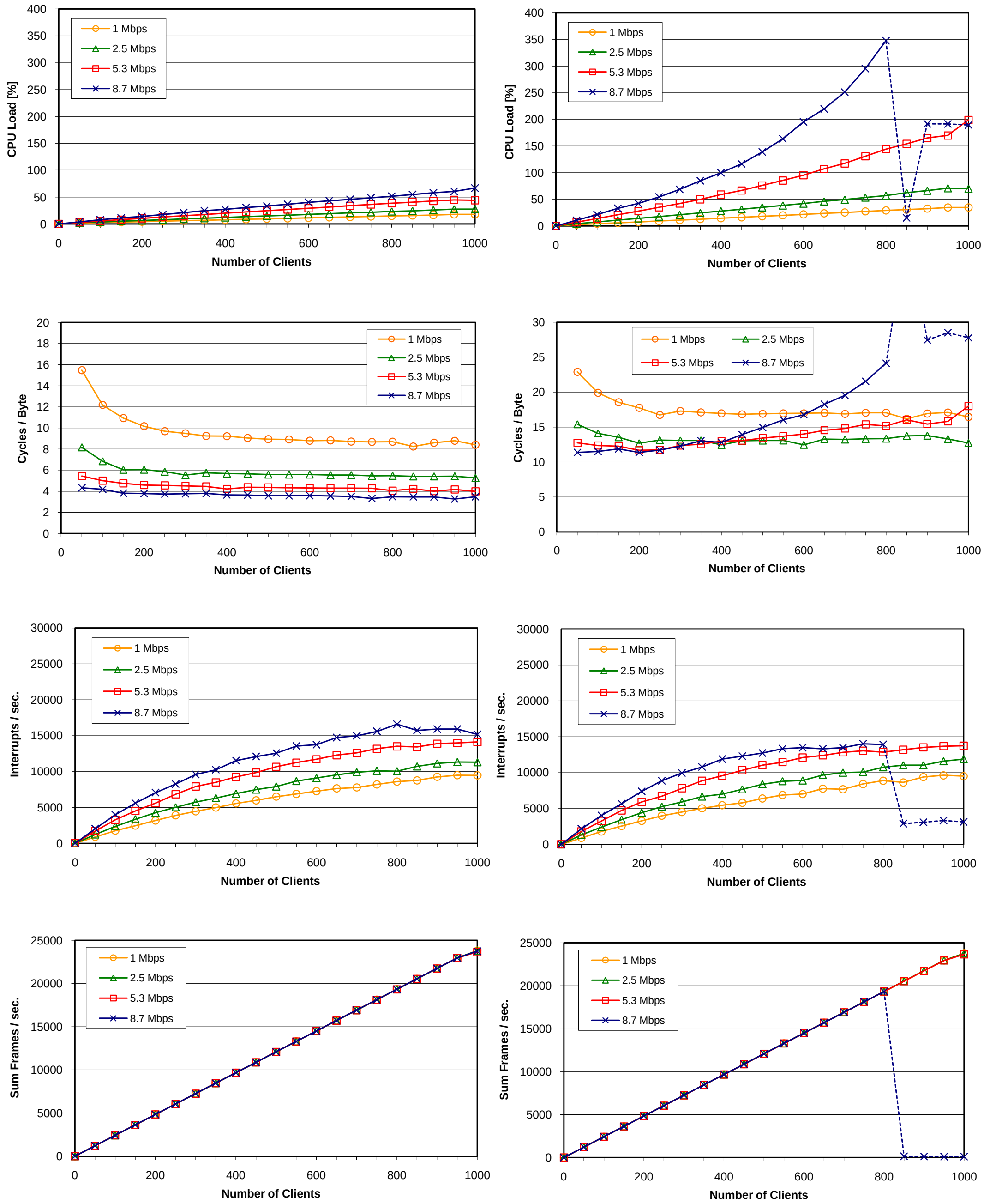

Figure 4.7: HTTP - Server Impact of bit rate (sendfile disabled left side, sendfile enabled right side)

higher compared to the initial measurements without any jumps. Decreasing the seek rate to 1 jump per minute reduces the CPU load to 140% which is still double the initial value. The differences to the initial measurements begin to fade out as the jump rate is lowered further. The lower chart indicate also no frame losses.

Section 2.3 describes the mechanism which can lead to a seemingly contradictory interrupt rate, where the number of interrupts is reduced although more clients connect to the server

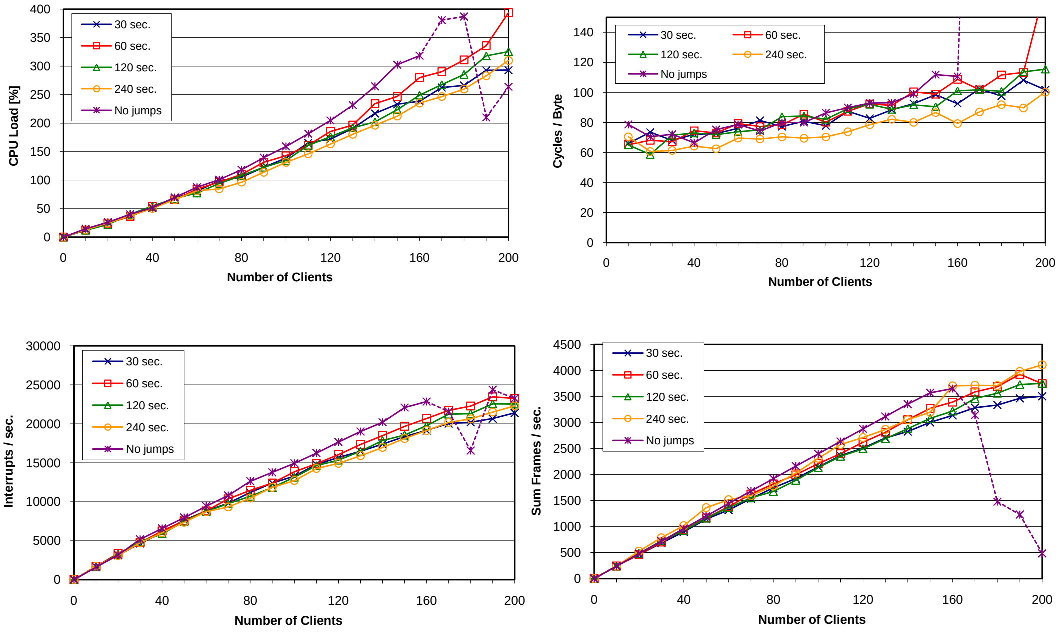

Figure 4.8: RTP - VLC Server Impact of seek rate

and the network load increases. This can be seen in the third row. In both cases with sendfile enabled and disabled the interrupt rate begins to drop after about half of the maximum supported users connected.

4.5 TCP Offload Engine

A TCP Offload Engine (TOE) is used to offload the processing of the TCP/IP stack from the CPU to the NIC. Especially NICs designed for high data rates (10 Gbit/s) come with an integrated TOE. In terms of performance savings it differs from a RNIC in respect of the copy operations and the context switches which are still necessary and occupy the CPU. In section 2.1 a chart indicates an estimate of the overhead introduced by TCP/IP processing and other factors. An estimate of the effect of the TOE is depicted in the middle column of Figure 2.1. The following experiments show the exact impact on the server performance caused by the processing of the TCP/IP stack. It evaluates the HTTP server streaming a video using default values. The NIC was used with TOE activated. In order to activate TOE another driver had to be installed. The charts on the left show the performance evaluation using the HTTP server without the sendfile feature enabled. Although, as expected, the top chart shows reduction of the CPU load, its degree is remarkable. Even without the sendfile feature enabled the server is able to handle up to at least 1'000 clients with only 150% CPU load without any frame losses (which can be seen in the lower charts). The CPU load is still exponentially increasing which suggests that once the bandwidth grows further it could pose a problem again. Although today, it seems sufficient as long as the data is sent using a TOE. Section 2.7 has shown that without TOE or the `sendfile()` mechanism, TCP alone would load all 4 cores to capacity. The interrupt rate increases first slower than without TOE enabled. But it in-

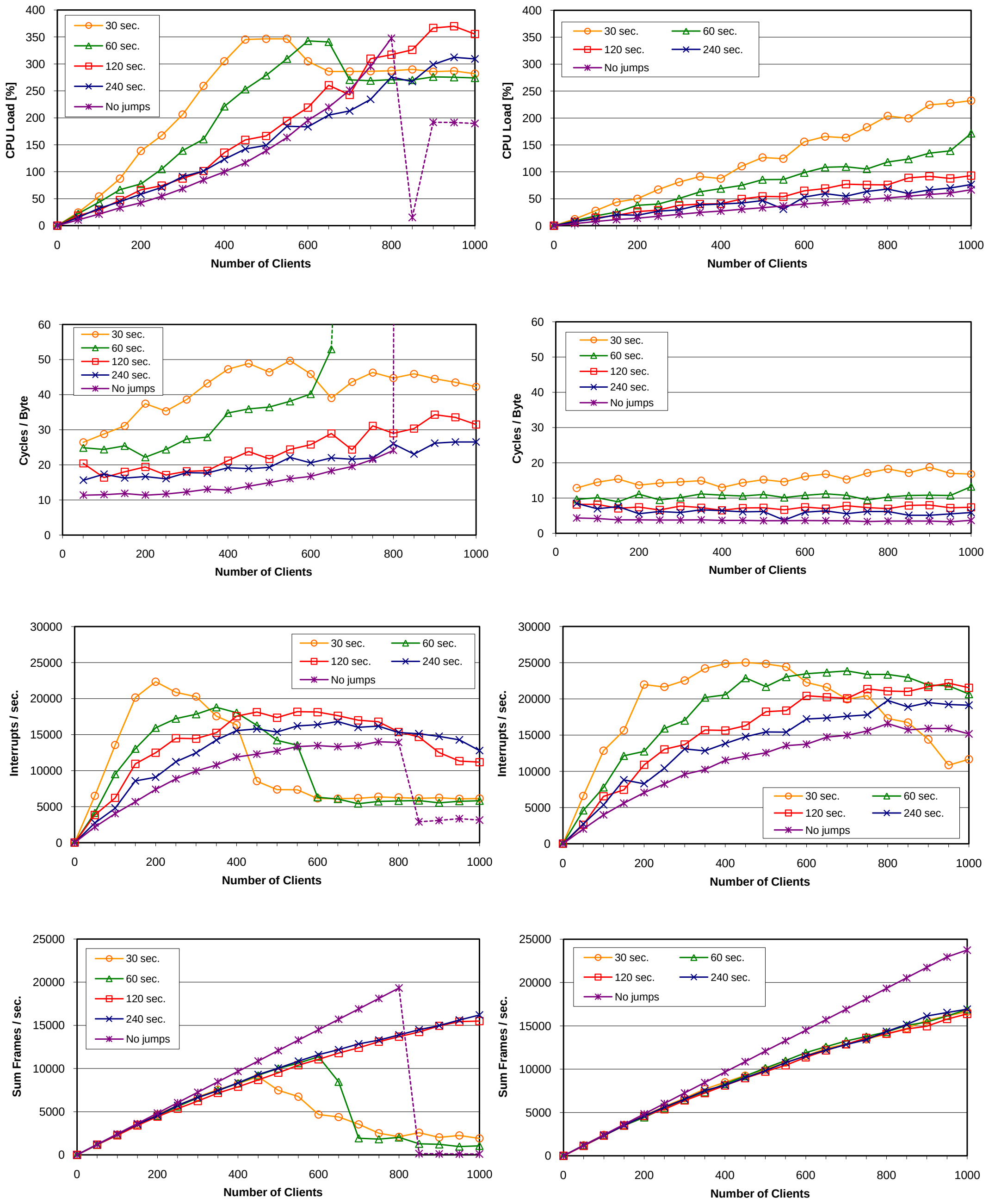

Figure 4.9: HTTP - Server Impact of seek rate (Sendfile off left side, sendfile on right side)

creases up to more than 20'000 interrupts/sec. whereas the test without TOE enabled stops at about 14'000 interrupts/sec. This could be due to the different drivers installed. Since the Linux kernel 2.6 switches from the interrupt-based system to the poll-based system when interrupts are not handled before the next ones arrive, the TOE driver could handle them more efficiently than the driver with TOE disabled. This would allow the system to remain longer in the interrupt-based state, leading to a higher interrupt rate. Looking at the charts on the right side showing the experiments with sendfile enabled, one can see a reduction of CPU

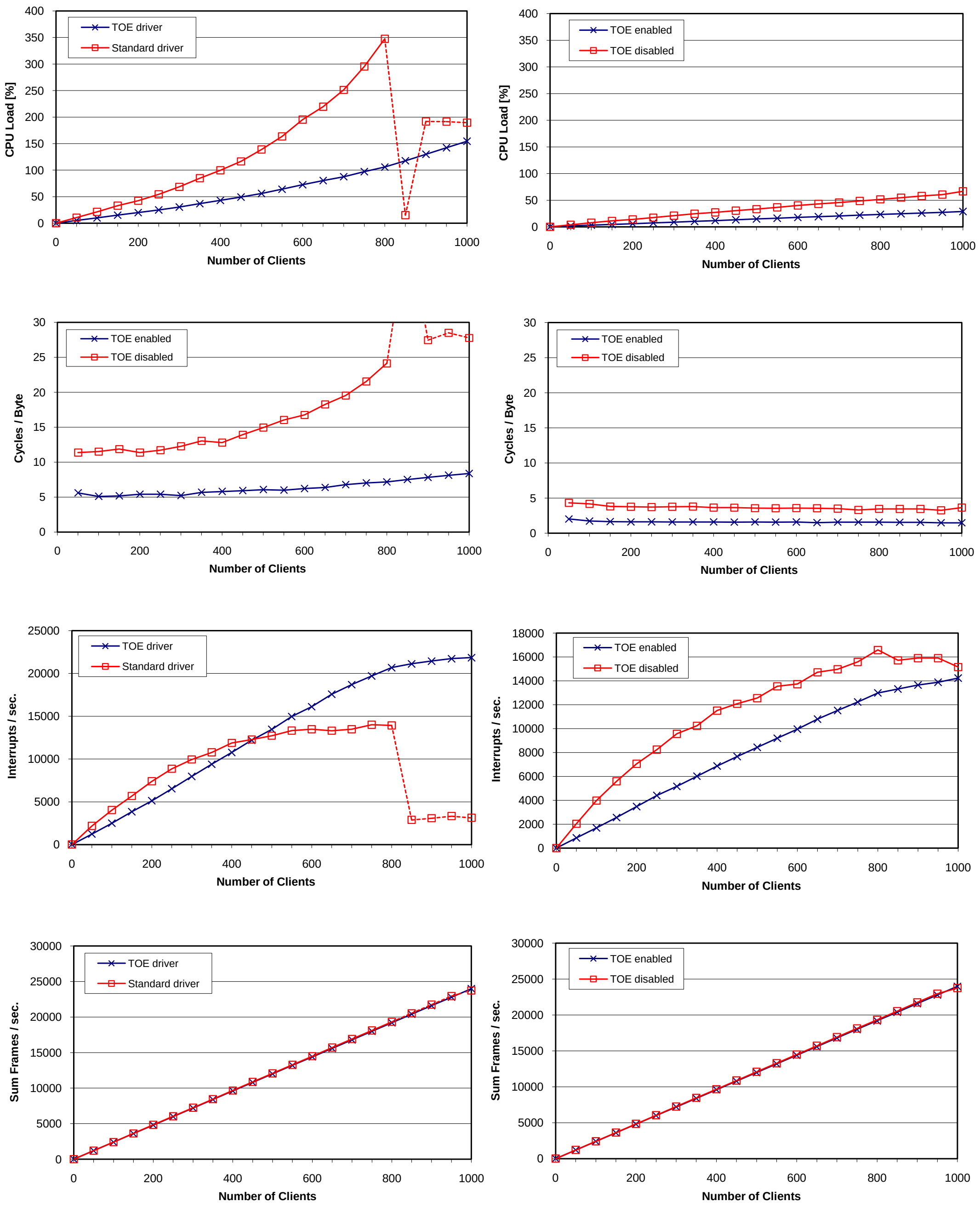

Figure 4.10: HTTP with TOE driver (left side without sendfile enabled, right side with sendfile enabled)

load by a factor of two. It is again possible to support at least 1'000 clients as depicted in the lower chart.

Overall the TOE improves performance most for HTTP without `sendfile()` enabled. Because of the exponential increase in CPU load this can result in a reduction of up to a factor three. Nevertheless it is also a significant improvement if the `sendfile()` feature is enabled.

5
Reducing the Local Data Copy Overhead

The results seen before indicated clear performance improvements using the `sendfile()` mechanism. However an iWARP-based approach potentially eliminates all server CPU involvement during video data dissemination. This chapter discusses the benefits for VoD offered by the new RDMA semantics, proposes a protocol based on iWARP and shows the performance measurements of an actual implementation of the protocol.

5.1 iWARP-based VoD Protocol

The purpose of an iWARP-based 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. One has basically the choice between two new and fundamentally different operations for streaming data: RDMA Write and RDMA Read.

5.1.1 RDMA Write-based Protocol

Using RDMA Write, the server pushes the data to the client. The clients do not know when new data was written into their memory because a completed RDMA write does only trigger an event on the data source, but not on the host where the data was written to. This means the server has to inform the client about the new data in its memory. Also the client would have to tell the server where it wants the data to be placed. In general, RDMA Write is more appropriate when the data source should be in control of the stream. With VoD you have the exact opposite. The data sink is controlling the stream. This leads to the conclusion that you could more easily use the RDMA Send/Receive, where the client can specify with its receive WR where the data should be written to (without having to inform the server about memory addresses) and the client would get a notification when something was written. In this respect the write semantic does not give an advantage, but Send/Receive does.

5.1.2 RDMA Read-based Protocol

On the other hand using RDMA Read gives advantages compared to the traditional `send()` / `receive()` operations. The client can control where the data is written to in its memory the same way, as it can do it with Send/Receive. But additionally, the client can trigger fetching data completely without any direct interaction with the server application. Whether it just reads sequentially the movie data or it jumps to another position within the movie, the server application does not need to take any direct action, because the Reads are handled solely by the RNIC. This brings the advantage that a server does not need to maintain the state of the clients. The only part where the server has to interact with the client is during the connection setup where it has to send information about mappings of the movie data and memory addresses. Hence the server implementation is extremely short.

The disadvantage is that the whole movie has to be in the main memory of the server. There are several upcoming solutions to this problem discussed in chapter 6.
For this reason the RDMA Read operations which allow a completely client-driven protocol, offering a VCR-like media control with at minimum server load will be used in our streaming system.
The communication protocol depicted in Figure 5.1 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 the connection request as well as to 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 consists of a triplet containing the starting address of the buffer holding the movie, the length in bytes and an RDMA steering tag (STag). The client has now all the information at hand to fetch the movie using RDMA Read operations. Since the whole 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.

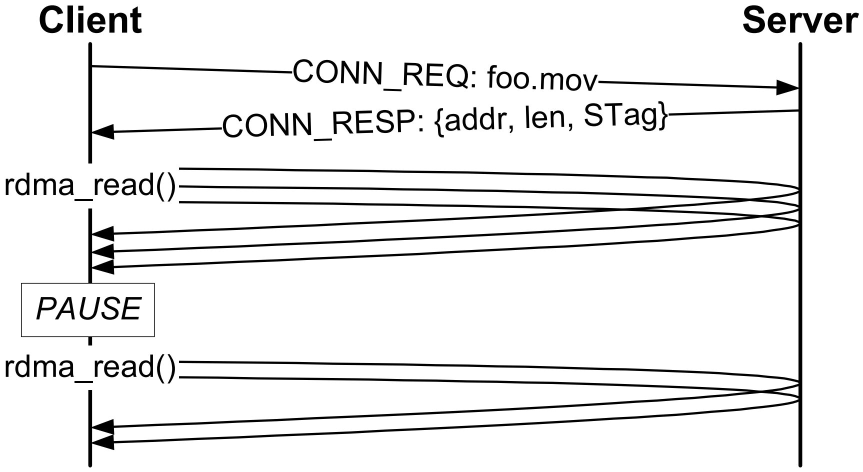

Figure 5.1: iWARP-based protocol

In the following sections we will first discuss how this protocol can be used together with existing streaming clients and then compare a concrete iWARP-based implementation with the previously analyzed systems.

5.2 Integration of iWARP-based Protocols into Existing Streaming Systems

This section will discuss two ways how iWARP can be combined with existing media streaming clients: Via a local proxy which translates another protocol to RDMA and a direct implementation into a streaming client.

5.2.1 iWARP Proxy

Figure 5.2 shows the schematic of the proxy. The iWARP proxy waits on the local port 80 for incoming HTTP requests. An arbitrary media player which is able to stream via the HTTP protocol, sends a HTTP *GET* request to the proxy (e.g. if the user wants to stream the video foo.mov it sends the request GET /foo.mov). Upon arrival of this request the proxy sets up a connection to the RDMA server and sends along with the connection request the request for the video. After the connection setup it can, as explained before, stream the video continuously via RDMA Reads. Once an RDMA Read completed on the proxy, it can deliver the video data with a HTTP *206 HTTP Partial Content* response to the media player. The proxy can also handle seeks within the video as described in section 5.1. It would also be possible to use RTSP/RTP as protocol between the proxy and the streaming client. But HTTP has the advantage, that the player already translates the positions within the video to a byte offset. RTSP/RTP provides positions within the video with a time offset and assumes that the server will translate it into a byte offset. Additionally, HTTP is simpler because there are fewer commands which have to be translated and there is only one communication channel between the client and the server.

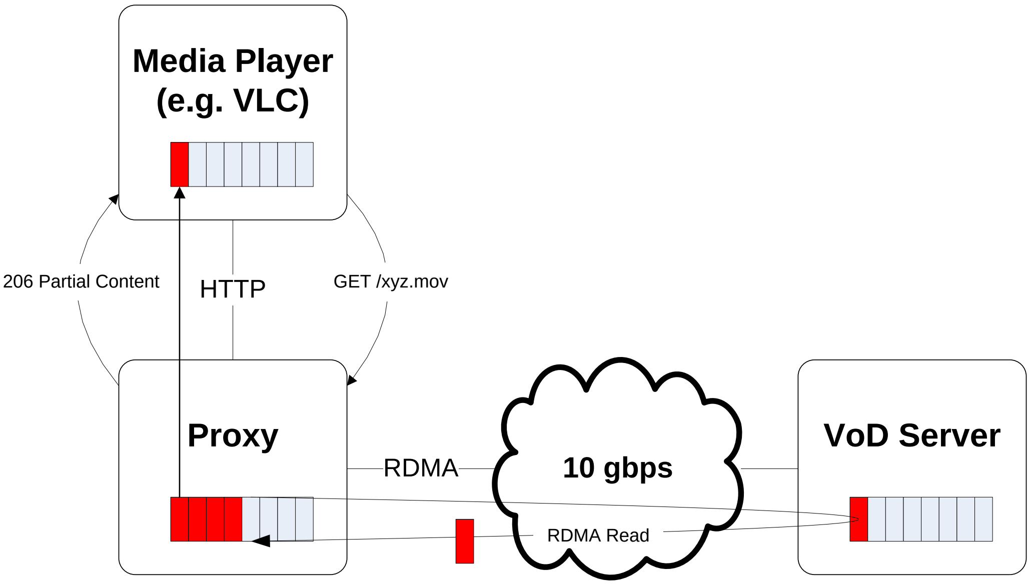

Figure 5.2: iWARP Proxy

This proxy solution has the advantage of being usable with all streaming clients that support HTTP. The server can profit from the full performance savings of an iWARP-based streaming system. However this approach has the disadvantage on the client side. It still has the overhead from the HTTP protocol and the data is not placed directly into the main memory of the application, as it would be possible with a direct implementation into the media player.

5.2.2 VLC Extension with an iWARP Module

Another way for an iWARP-based solution is to implement it directly into the streaming client. This allows taking full control of the used protocol. Once again the VideoLAN client as one of the leading open source media players was used to implement the iWARP extension. This implementation places the data directly to the position in the memory where the decoder modules of VLC need it to be later on and is thus as efficient as possible also on the client side. In addition to data copy avoidance, using an iWARP-based protocol with an RNIC avoids interrupts from the NIC since the CPU is not involved in protocol processing (see section 5.3). This is highly desirable as it reduces the context switching rate significantly and therefore leads to less cache pollution. The core of the iWARP module are the functions `Open()` / `Close()`, `Block()` and `Seek()`. The `Open()` and `Close()` functions take care of the connection setup, initial video information exchange and tear down as described in section 5.1.2. The `Block()` function delivers chunks of video data. The function always caches blocks in advance according to the specified caching value. The VLC Client calls the function when the Demuxer is ready to take new data. Upon a Block request the module checks whether new blocks from previously posted RDMA Read requests already arrived. It then delivers them and places new RDMA Read requests on the Send Queue. The `Seek()` function is called when the client jumps to a new position within the video. Due to iWARP restrictions all the previously sent RDMA Read requests have first to be polled or the connection must be reset. Since there are always only few requests pending, this function waits until all previous RDMA Reads finished and discards the data. It then fills up the Send Queue with RDMA Read Requests with a total size of the specified caching value.

5.2.3 RDMA Read Size

An open question in both previously described systems is the size of the RDMA Reads. As described in 2.1.1, iWARP does not transmit data in order which means that one can only be certain that the first bit of a RDMA Read has arrived after the whole RDMA Read is completed. This means that one can not simulate a continuous stream by issuing RDMA Reads of a big size. RDMA is designed for large amounts of data. Issuing many small RDMA Reads could degrade the throughput. This is why it is important to measure the performance of RDMA Read with different sizes.

For this reason, we set up a simple server on one of the BladeServers which allocates a memory region, fills it with random numbers and accepts one client. Another BladeServer acts as client which reads the whole memory region sequentially by issuing continuously RDMA Reads. The WRs are posted in parallel, meaning that the client does not wait for one RDMA Read to complete until the next one is posted. First, there are 64 RDMA Read requests posted at once. After having posted a series of request the client polls until at least one RDMA Read has completed and posts immediately afterwards again the same amount of requests to the send queue. During the experiment only few Read requests complete at once, meaning that there are always at least 50 RDMA Read Requests outstanding. The Read size is varied from 1 Byte to 2 GB. Figure 5.3 shows the results of the experiment. It indicates that reading a page size or more at once (4 KB) is enough to use the whole bandwidth. Before that point the overhead has a significant impact on the throughput. Since usually media players cache several hundred milliseconds of the video which is even with an SD bit rate (1 Mbit/s) at least 12 KB for 100ms, it is possible to simulate a continuous video stream with RDMA. As shown in section 2.7 sending data packets smaller than 4 KB leads also to performance problems with TCP.

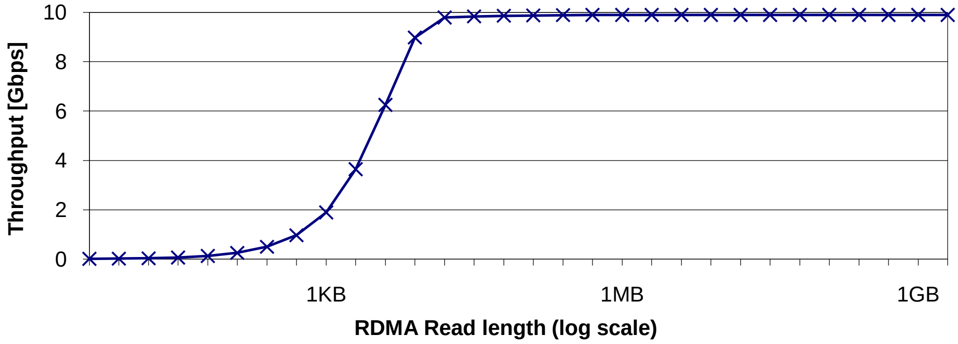

Figure 5.3: RDMA Read throughput with different read sizes

5.3 Performance Evaluation

This section gives a short performance evaluation of the iWARP implementation and compares it directly with the other protocols.

5.3.1 iWARP evaluation

The results of the performance evaluation compared to the HTTP server can be seen in Figure 5.4. The chart on the top reflects the CPU load of the iWARP-based solution compared to the HTTP server. The iWARP implementation can saturate the link by serving the required data to all clients, whereas the HTTP server without sendfile support has an exponentially increasing CPU load. Although the HTTP server with `sendfile()` induces up to 70% CPU load, this is not much load, the savings from the RDMA solution can be important. If

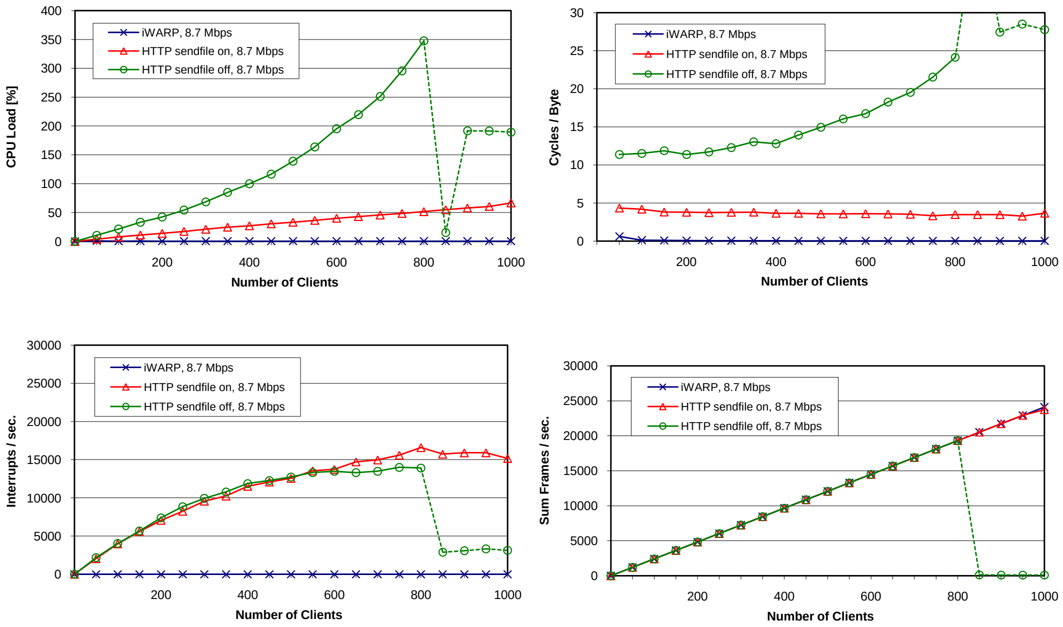

Figure 5.4: HTTP compared to iWARP performance

a Video Server simultaneously has other work to do, like encoding a video stream from an external source (e.g. a video camera) this extra CPU power is necessary.
Not very surprisingly the tests with the lower bit rates indicate no late or lost frames and a negligible amount of interrupts and CPU load. Comparing the iWARP-based system to the RTSP/RTP systems shows an even bigger performance difference. Contrary to the HTTP server which also induced a high number of interrupts, iWARP is by all indicators by far superior.
More interesting is whether the RNIC can handle high numbers of jumps within the movie. Figure 5.5 illustrates the measured indicators with in average one jump every 30 seconds compared to the HTTP results. As usual iWARP induces negligible CPU load and interrupts on the server side. But the last chart depicts problems with iWARP on the client side. The result for iWARP indicates that it can only handle up to 400 clients. A manual check was performed to verify the result. First, 800 clients were started on 4 BladeServers and then one single client which actually decodes the video and displays it was started on the fifth client BladeServer. The client was able to stream the video and decode it without problems. It could also handle jumps within the video at a rate of one jump every 30 seconds and more. The client BladeServers seem not to be able to handle 200 RDMA enabled VLC clients which perform a jump every 30 seconds. It was not possible to perform a reliable test with the available infrastructure.

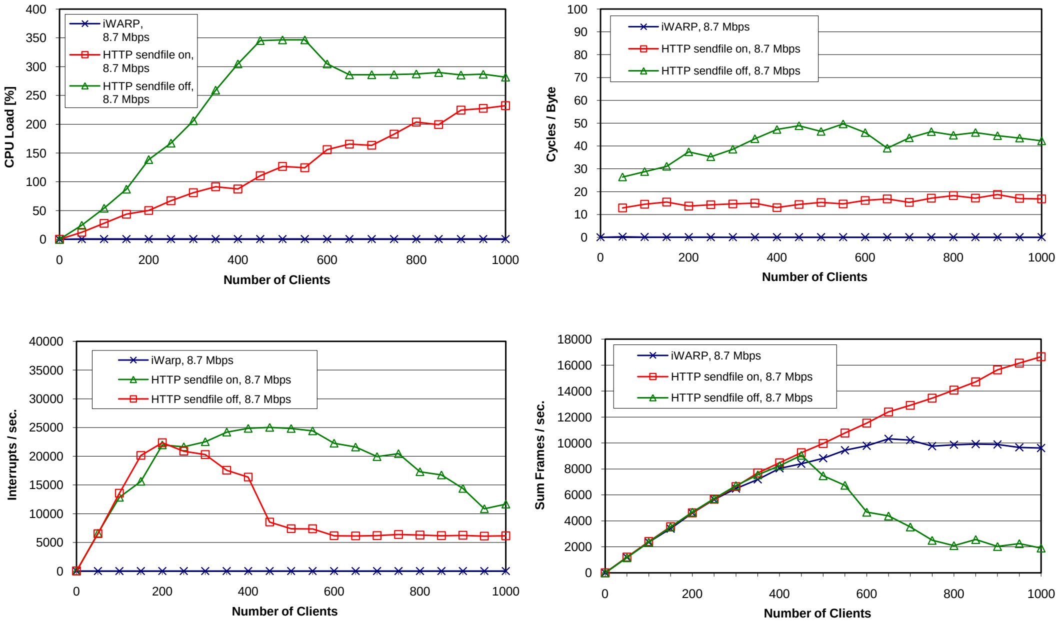

Figure 5.5: Jumps every 30 seconds - HTTP compared to RDMA performance

5.3.2 Direct Protocol Comparison

A VoD server offering HD media based on RTP suffers from a high interrupt- and context switching rate as well as a CPU overhead exponential to the number of served clients. By using HTTP instead, the overhead can be reduced by about an order of magnitude (see Figure 5.6; logarithmic scale; quad core test server). Applying the `sendfile()` mechanism

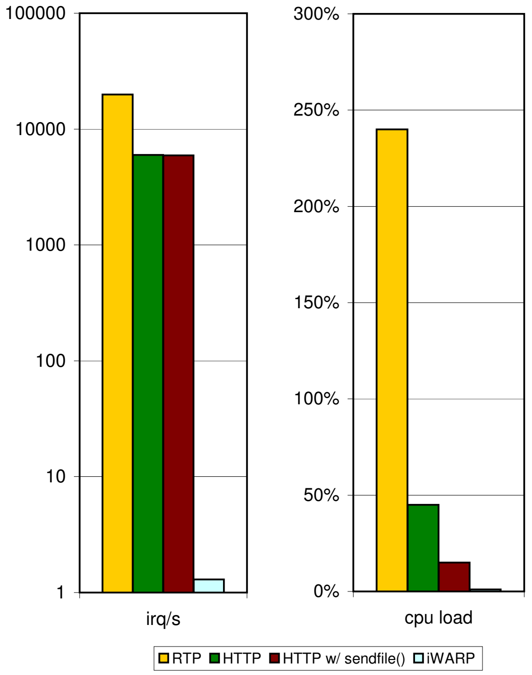

Figure 5.6: Direct Protocol Comparison

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 necessitate 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 while still being able to do zero-copy data transmission on the server side.

In order to allow for 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. When using RDMA however, 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 to HTTP and bring the CPU load down to a small constant.

6
Outlook and Future Work

In the previous chapters it was shown how existing media servers perform in a VoD scenario. By proposing an iWARP-based application protocol, we have seen how an efficient VRC-like media control can be implemented. 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 data into local memory. The following section proposes an iWARP-based Live Streaming system. The implementation and evaluation of this system is part of future work. Based on this system it will be shown how one can reduce the disadvantage of having to load the video content into the main memory.

6.1 Live Streaming

The `sendfile()` approach can not directly be applied for 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 our extension, we argue that live streaming can be seen as a special case of VoD where the user does not interact with the stream. 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.

The server maintains a ring buffer where the encoded data chunks from the source are kept. The two key parameters are the size of the ring (number of chunks) as well as the size of the individual chunk. The larger the ring buffer is, the longer a chunk is available before it is overwritten again. The size of the individual chunks on the other hand have a direct impact on the delay between the live stream and the client playback time: larger chunks increase the delay because it takes longer until they are filled. We assume without loss of generality that a client always reads a whole chunk.

The ring buffer which is continuously updated necessitates an extension of the application protocol presented before: The server needs to inform the client whenever a new chunk has become available. The client will then fetch the data as needed. Our extension for live streaming using the iWARP-based protocol is to add a small notification message sent by the server to each client via a Send/Receive operation. By experiment we have found that sending the notification messages has very little impact on the server performance (less than 5% CPU load for sending notifications to the 1'000 clients). These notifications could even be omitted if the stream was encoded with a fixed bit rate — the clients could then simply fetch the data at the appropriate rate.

6.2 VCR-Like Media Control using Live Streaming

This section presents a way to stream media in a VoD scenario, but in contrast to the previously presented VoD protocol, it needs only little amount of data in main memory. The data source is a completely encoded video on the hard disk. As described in the related work section there has already been done extensive research on how to use several broadcast live streams of the same video to allow for VCR-like media control while saving bandwidth. These schemes can also be used to save memory in the iWARP-based VoD scenario. For example one can use the pyramid broadcast mechanism as shortly described in chapter 3. Instead of using broadcast to send it over the network the broadcast ends in the memory of the server. The server divides the video into segments as shown in Figure 6.1. The first segment should only consist of a few seconds of video data. Each following segment should be twice as large as the previous segment. Depending on the video size a logarithmic number of segments is needed. The server plays all these segments in parallel by streaming them into the memory and allowing the clients to tune into the stream as described previously in 6.1. A client who wants to watch the video has only to wait a short amount of time until the first stream wraps around and begins again streaming the first segment from the beginning. The client has to wait until that happens. Then it can tune into this stream and watch the video. In parallel it has to stream all the other segments and cache them locally. When the first stream finished, the second stream will either be at the end of the second segment or in the middle. When it is at the end it wraps around and begins to play the second segment again and the client can just play that stream. When it is in the middle of the second segment the client has already cached the first half of the second stream (since the first stream takes half as long as the second stream). This guarantees short waiting times until the client can begin to stream the video. Furthermore the client can jump forward to one of the other streams or to a part which it has already cached of these streams. This way, the client would have some advantages of VoD, although it is not true VoD. The server has eliminated its memory problem, since there is only a logarithmic number of streams necessary per video. The disadvantage is that the client must have enough bandwidth to read all streams in parallel. For the server the aggregate throughput is the same, since all clients request all data only once.

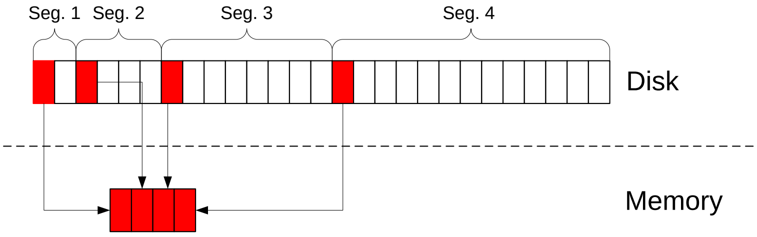

Figure 6.1: Pyramid Broadcast as Caching Mechanism

7 Conclusions

Through experiments it was shown that RTSP/RTP Video on Demand servers do not scale with a high aggregate throughput. With current trends of broadband connections the bottleneck is shifting from the bandwidth to the server infrastructure itself. Only 160 clients requesting a 1080p HD movie can be supported and only about 15% of the full bandwidth can be used. The limiting factor is the CPU which is fully loaded at that point. Lower bit rates allow for more users, but estimates predict an increasing demand in high resolution videos. HTTP performs significantly better than RTSP/RTP. Even without sendfile support enabled it can support under full CPU load up to 800 clients requesting an 8.7 Mbps movie. Enabling sendfile support increases the number of clients to 1'000 which is the maximum allowed by the bandwidth and decreases CPU load to 70%. However the interrupt rate and context switches are still high.

In contrast our iWARP-based VoD solution using a 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. several Gbps per client). The drawback of the RDMA solution is that the server must be equipped with a RNIC. In addition to that, the clients must have RDMA stack support as well — but here a software RDMA stack may be completely sufficient (e.g. IBM SoftRDMA driver [29]). Since 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) or by attaching the server to a RamSan system [41] 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 reestablishment 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 reopen 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.

8 Acknowledgment

I want to express my gratitude to my thesis advisor Philip Frey. During the course of this master thesis he helped me with invaluable assistance, advice and guidance. I want to thank Prof. Dr. Alonso for supervising the thesis, discussing it in regular meetings and support me with many ideas. My thanks also go out to Dr. Bernard Metzler and Dr. Fredy Neeser for supporting me and giving me many insights on RDMA and related issues. Special thanks go to Dr. Patrick Droz who always had time to help me during the last six month.

Bibliography

[1] VLC Media Player. `http://www.videolan.org`.

[2] A Permutation-Based Pyramid Broadcasting Scheme for Video-on-Demand Systems. In *ICMCS '96: Proceedings of the 1996 International Conference on Multimedia Computing and Systems (ICMCS '96)*, page 0118, Washington, DC, USA, 1996. IEEE Computer Society.

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

[4] Kevin C. Almeroth and Mostafa H. Ammar. The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal on Selected Areas in Communications*, 14:1110–1122, 1996.

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

[6] Apple Inc. Quicktime streaming server. `http://www.apple.com/quicktime/streamingserver/`.

[7] BBC Research department. Dirac video compression. `http://diracvideo.org`.

[8] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), 1996.

[9] Blu-ray Disc Association. Application Definition Blu-ray Disc Format. `http://www.blu-raydisc.com/Assets/Downloadablefile/bdj_gem_application_definition-15496.pdf`, 2005.

[10] comScore Inc. YouTube Attracts 100 Million U.S. Online Video Viewers in October 2008. `http://www.comscore.com/press/release.asp?press=2616`.

[11] P. Culley, U. Elzur, R. Recio, S. Bailey, and J. Carrier. Marker PDU Aligned Framing for TCP Specification. RFC 5044 (Proposed Standard), October 2007.

[12] Dennis Dalessandro and Pete Wyckoff. Accelerating Web Protocols Using RDMA. In *The 6th IEEE International Symposium on Network Computing and Applications*.

[13] Kevin Fall and Joseph 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.

[14] Kevin Fall and Joseph 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.

[15] Annie P. Foong, Thomas R. Huff, Herbert H. Hum, Jaidev P. Patwardhan, and Greg J. Regnier. On optimal batching policies for video-on-demand storage servers. In *Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems*, pages 253–258, 1996.

[16] Annie P. Foong, Thomas R. Huff, Herbert H. Hum, Jaidev P. Patwardhan, and Greg J. Regnier. TCP performance re-visited. In *In IEEE International Symposium on Performance of Systems and Software*, pages 70–79, 2003.

[17] Philip Frey and Gustavo Alonso. Minimizing the Hidden Cost of RDMA. *IBM Journal of Research and Development. Special Issue on Network-Optimized Computing*, (2009, in press).

[18] Alberto Garcia-Martinez, Jesus F. Conde, and Angel Vina. Efficient Memory Management in VOD Disk Array Servers Using Per-Storage-Device Buffering. In *Euromicro Conference, 1998. Proceedings. 24th*, pages 551–558, 1998.

[19] Pal Halvorsen, Espen Jorde, Karl andr Skevik, Vera Goebel, and Thomas Plagemann. Performance Tradeoffs for Static Allocation of Zero-Copy Buffers. In *Proceedings of the 28th Euromicro Conference*, 2002.

[20] Jeff Hilland, Paul Culley, Jim Pinkerton, and Renato Recio. RDMA Protocol Verbs Specification. `http://www.rdmaconsortium.org/home/draft-hilland-iwarp-verbs-v1.0-RDMAC.pdf`.

[21] Jay Loomis and Mike Wasson. VC-1 Technical Overview.

[22] Jonathan Kay and Joseph Pasquale. Profiling and reducing processing overheads in TCP/IP. *IEEE/ACM Trans. Netw.*, 4(6):817–828, 1996.

[23] Sam Liang and David Cheriton. TCP-RTM: Using TCP for real time multimedia applications. In *in International Conference on Network Protocols*, 2002.

[24] Frank W. Miller, Pete Keleher, and Satish K. Tripathi. General Data Streaming. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 232–241, 1998.

[25] Jeffrey Mogul, Dec Western, Jeffrey C. Mogul, and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, 15:217–252, 1997.

[26] Jeffrey C. Mogul. TCP offload is a dumb idea whose time has come. In *HOTOS'03: Proceedings of the 9th conference on Hot Topics in Operating Systems*, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.

[27] John Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, 1984.

[28] Fredy Neeser, Bernard Metzler, and Philip Frey. SoftRDMA: Implementing iWARP over TCP Kernel Sockets. *IBM Journal of Research and Development. Special Issue on Network-Optimized Computing*, (2009, in press).

[29] Fredy Neeser, Bernard Metzler, and Philip W. Frey. SoftRDMA. `http://www.zurich.ibm.com/sys/servers/rdma_soft.html`.

[30] On2 Technologies Inc. On2 VP8. `http://www.on2.com/`.

[31] L. Ong and J. Yoakum. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational), 2002.

[32] K. Onthriar, K.K. Loo, and Z. Xue. Performance Comparison of Emerging Dirac Video Codec with H.264/AV. *Digital Telecommunications, International Conference on*, 0:22, 2006.

[33] J. Pinkerton and E. Deleganes. Direct Data Placement Protocol (DDP) / Remote Direct Memory Access Protocol (RDMAP) Security. RFC 5042 (Proposed Standard), 2007.

[34] Thomas Plagemann, Vera Goebel, Pal Halvorsen, and Otto Anshus. Operating System Support for Multimedia Systems. *The Computer Communications Journal*, 23:267–289, 2000.

[35] Bashar Qudah and Nabil J. Sarhan. Analysis of Resource Sharing and Cache Management in Scalable Video-on-Demand. In *MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pages 327–334, Washington, DC, USA, 2006. IEEE Computer Society.

[36] RealNetworks Inc. The Helix DNA Server. `https://helix-server.helixcommunity.org/`.

[37] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia. A Remote Direct Memory Access Protocol Specification. RFC 5040 (Proposed Standard), 2007.

[38] A. Romanow and S. Bailey. An Overview of RDMA over IP. In *Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks*, 2003.

[39] Vik Saxena. Bandwidth drivers for 100 G Ethernet. `http://www.ieee802.org/3/hssg/public/jan07/Saxena_01_0107.pdf`, 2007.

[40] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), 2003.

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

[42] ThomasWiegand, Gary J. Sullivan, Senior Member, Gisle Bjntegaard, and Ajay Luthr. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Cirquits and Systems for Video Technology*, 13:560–576, 2003.

[43] Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya qin Zhang, and Jon M. Peha. Streaming Video over the Internet: Approaches and Directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11:282–300, 2001.

List of Figures

List of Tables

A
Appendix

The core of this master thesis is summarized in the following paper. It is currently under review by the NOSSDAV 09 program committee.

Server-Efficient High-Definition Media Dissemination

Philip W. Frey, Andreas Hasler, Bernard Metzler
Systems Department
IBM Research GmbH, Rueschlikon

Gustavo Alonso
Systems Group
Department of Computer Science, ETH Zurich

ABSTRACT

Internet usage has changed dramatically in the last few years. Content is no longer dominated by static websites but comprises an increasing amount 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 ISPs 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 is 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 operations, context switching, and interrupt processing overhead. Such an overhead causes high CPU loads and turns the server's memory bus into a bottleneck. In this paper we illustrate and discuss the 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 using dedicated RDMA hardware. Our performance experiments indicate that these optimizations allow servers to scale better and remove most of the overheads caused by current approaches.

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 rapidly increasing [1]. In addition, the web content is no longer static but has become dynamic and far richer. According to Comcast [2], 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 [3], *high definition video-on-demand* (HD VoD) poses two orthogonal service requirements:

1. data throughput large enough to deliver the high-definition content, and

2. low latency as well as a high responsiveness to support convenient media control.

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

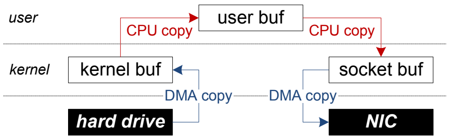

Figure 1: Copy Overhead for sending File Content

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.

Given these limitations, scalability in existing systems is achieved by adding more servers. While the 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. HD VoD will make the cost of scaling significantly higher if the same approach is used.

The problem we address in this paper is how to improve the scalability of a single video server (its ability to server a

NOSSDAV '09 Williamsburg, Virginia

larger number of clients) so that the demands of HD VoD on the overall server infrastructure are minimized. Such an improvement involves removing the overheads encountered in current systems. For this purpose, we propose a novel protocol based on *Remote Direct Memory Access* (RDMA [4]).

The contributions of this paper are three-fold. First, we analyze the performance of existing solutions and identify the 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 video-on-demand services. Section 4 presents a novel iWARP/RDMA-based protocol for video-on-demand as well as streaming services. Section 5 evaluates the proposed solutions and highlights their tradeoffs. Section 6 concludes our work.

2. BACKGROUND

2.1 iWARP, Remote Direct Memory Access over Ethernet

Remote Direct Memory Access (RDMA) is a mechanism whereby, without CPU involvement, network data is directly fetched from local application memory and placed into application memory of a remote computer. This placement is typically performed by a specialized RDMA-enabled NIC (*RNIC*). In bypassing the operating system and eliminating intermediate copying across application and operating system 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 high-definition media across the network. RDMA technology was initially restricted to specific, often proprietary technology such as InfiniBand [5]. With the advent of 10 Gb Ethernet and the TCP-based Internet Wide Area RDMA Protocol (iWARP), a special infrastructure is no longer required. RDMA can now be applied to applications communicating over the Internet.

A detailed description of the RDMA semantic can be found in [6]. 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 for 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 gets informed. To allow the RNIC to fetch and to place data without operating system involvement, concerned 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 RNIC and peer applications 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. In [7] they proposed a new UNIX system call: `splice()` 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. In [8], the same authors have shown the applicability of the splice system call to continuous-media playback over a UDP transport. 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.

[9] presents another copy avoidance solution for media-on-demand servers. 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.

An acceleration of HTTP using RDMA has been proposed in [10]. An iWARP Apache module was presented which extends the HTTP protocol with RDMA support. While HTTP is push-based, we propose a completely new protocol which 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 [11, 12]. Detailed discussions of the different challenges posed by designing mechanisms and protocols for Internet streaming services can be found in [13, 3]. In our work we assume the bandwidth to be sufficient and investigate server-side limitations.

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 criteria 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 investigated the influence of other service parameters such as offering different numbers of movies, different sized movies, movie access through various patterns as well as changing the client-side cache size. Compared to the actual bit rate, none of the other parameters showed a significant impact on scalability. Therefore, they are not further discussed.

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. Since 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 as well as 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 video-on-demand services using RTP: The open source VideoLAN Server [15] and the Darwin Streaming Server [16].

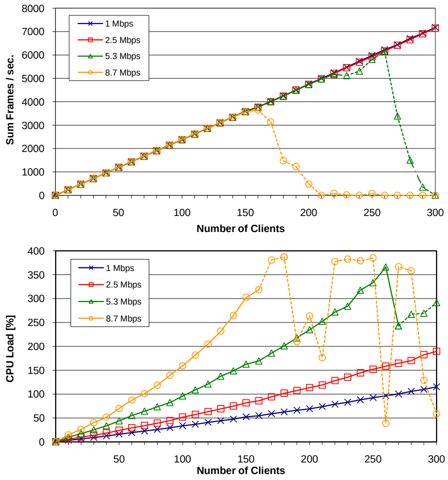

Figure 2: RTP Scalability for various Bit Rates

The upper chart of Figure 2 shows the cumulative rate of frames all clients have successfully received. 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 Mbps and at 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 at about 1100, a factor 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* [17], 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 non deterministic, non reproducible 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 for a 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.

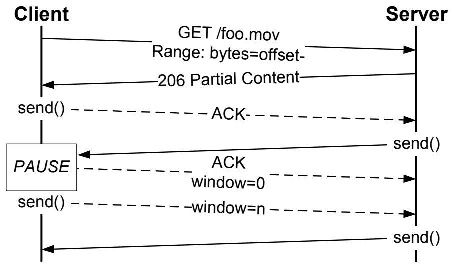

Figure 3: HTTP-based Stream Control

In the last section we have shown that RTP/UDP does not manage to fully utilize the 10 GbE link due to the resultant high CPU load when streaming at high bit rates. We now repeat the above experiment with the 8.7 Mbps data rate in order 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 get a fair comparison with RTP, for the current experiment, the `sendfile()` mechanism was disabled (see next section for `sendfile()` tests).

As can be seen in Figure 4 the CPU limit (where all 4 CPUs are completely busy) is reached much later with HTTP compared to RTP due to the reduced packetizing overhead and simpler connection management. However, the link can still not 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()`

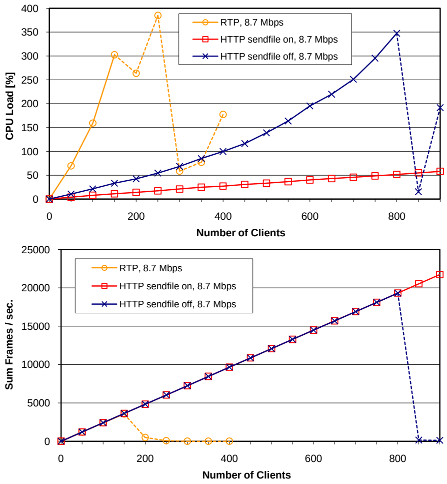

Figure 4: HTTP vs. RTP

system call. In contrast to read()/write(), it allows the data to be sent directly from the temporary kernel buffer onto the network. Since a modern NIC is equipped with a DMA engine (like the Chelsio T3 adapter we have used for our experiments), the CPU copies are avoided altogether. Furthermore, the number of context switches and 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 allow for a completely client-driven protocol offering efficient VCR-like media control at minimum server load.

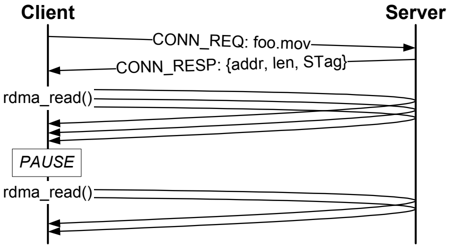

Figure 5: Client-driven RDMA Protocol

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 the connection request as well as to 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 consists of a triplet containing the starting address of the buffer holding the movie, the length in bytes and an RDMA steering tag (STag). The client has now all the information at hand to fetch the movie using RDMA Read operations. Since the whole 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

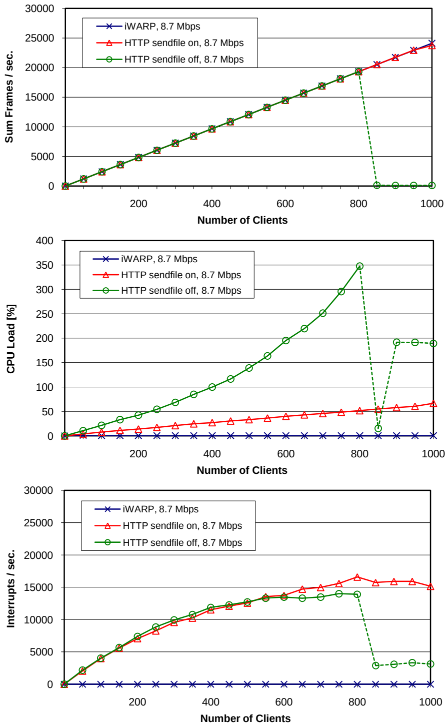

Figure 6: iWARP vs. HTTP

Figure 6 shows the performance results of our protocol compared to plain HTTP and HTTP with sendfile() support. We have limited the experiment to 1000 clients as this is enough to fill our 10 GbE link. The chart on the top 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 servers 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 local CPU performance, the RNIC itself is able to saturate the link by processing the RDMA Read requests in hardware. In addition to data copy avoidance, 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 [18].

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 effort to control the connections is done at clients side, thus freeing the server from almost all application protocol processing. The advantages of the simple protocol are three-fold:

First, each client is free to read any amount from any position within the advertised buffer whenever new data for playback is needed. In case there are several movies 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

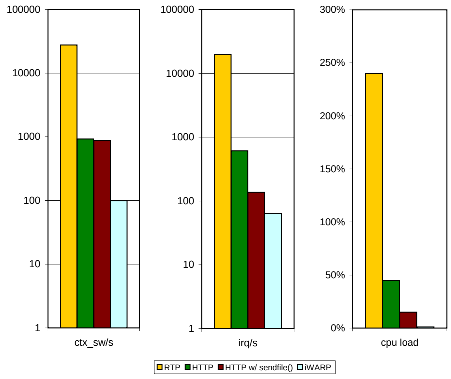

Figure 7: Direct Protocol Comparison

A VoD server offering high-definition media based on RTP suffers from a high interrupt- and context switching rate as well as a CPU overhead exponential to the number of served clients. By using HTTP instead, the overhead can be reduced by about an order of magnitude (see Figure 7; logarithmic scale; quad core test server). 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 necessitate 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 while still being able to do zero-copy data transmission on the server side.

In order to allow for 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. When using RDMA however, 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 to 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. several Gbps per client). The drawback of the RDMA solution is that the server must be equipped with an RNIC. In addition to that, the clients must have RDMA stack support as well — but here a software RDMA stack may be completely sufficient (e.g. IBM SoftRDMA driver [19]). Since 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 [21] 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 can not 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 servers 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 are 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. CONCLUSIONS AND OUTLOOK

We have analyzed and shown by experiment why server-side copy avoidance is key to achieve 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 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 in contrast to the `sendfile()` zero-copy mechanism offered by the Linux kernel.

7. REFERENCES

[1] "World internet stats." `http://www.internetworldstats.com/stats.htm`.

[2] V. Saxena, "Bandwidth drivers for 100 g ethernet." `http://www.ieee802.org/3/hssg/public/jan07/Saxena_01_0107.pdf`, 2007.

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

[4] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia, "A Remote Direct Memory Access Protocol Specification." RFC 5040 (Proposed Standard), 2007.

[5] "InfiniBand Trade Association." `http://www.infinibandta.org`.

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

[7] 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*, pp. 327–333, 1993.

[8] 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)*, pp. 100–109, 1994.

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

[10] D. Dalessandro and P. Wyckoff, "Accelerating web protocols using rdma," in *The 6th IEEE International Symposium on Network Computing and Applications*.

[11] T. P. Nguyen and A. Zakhor, "Distributed video streaming over internet," in *Multimedia Computing and Networking (MMCN02)*, pp. 186–195, 2002.

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

[13] 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*, vol. 11, pp. 282–300, 2001.

[14] O. Alliance, "Openfabrics enterprise distribution." `http://www.openfabrics.org/`.

[15] "VLC media player." `http://www.videolan.org`.

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

[17] "OProfile - A System Profiler for Linux." `http://oprofile.sourceforge.net`.

[18] 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*, pp. 75–84, 1991.

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

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

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