A Study between Networks and General Purpose Systems for High Bandwidth Applications

John Bresnahan, Ioan Raicu, Gohar Margaryan

CMSC322: Computer Architecture – Spring Quarter 2004
Department of Computer Science
University of Chicago
bresnaha@mcs.anl.gov, iraicu@cs.uchicago.edu, gohar@cs.uchicago.edu

ABSTRACT

The goal of this study is to investigate system bottlenecks for high bandwidth applications and how they shift from one component to another over time. The model we built emphasizes the flow of bytes in each protocol; the model imitates how typical device drivers move bytes from the network card to memory, and how typical protocol implementations move bytes around memory. Network speeds have been increasing at a very fast rate over the last decade. 100Mb/s sec network interfaces are virtually ubiquitous, 1 Gb/s interfaces are becoming standard and inexpensive, and 10 Gb/s interfaces are already commercially available. As higher speed network data transfers have become possible high bandwidth applications are likely to become a driving factor in computer system design. Network interfaces are not the only piece of hardware involved in high bandwidth applications; processors, disks, and memory architecture all play equally important roles. It is important to consider if these components can keep up with the increasing network speeds. In order to gain insight in the performance of computer systems as individual components increase in performance at different rates, we evaluated some existing systems and built a simulator, called sysSIM that models the flow of bytes through the components of a system. SysSIM is a discrete event driven simulator that allows us to investigate the roles and interactions of all of these components (CPU, memory, and network interfaces) to identify bottle necks. We conclude our study with a 40 year view of how the bottleneck has shifted from the network (prior to 1995) to the processor (between 1995 and 2007), and to the memory (beyond 2007).

TABLE OF CONTENTS

ABSTRACT **1**

TABLE OF CONTENTS **2**

1 INTRODUCTION **3**

1.1 SysSIM Overview 4
1.2 Motivation: Historical Trends 5
1.3 Timeline 7

2 RELATED WORK **8**

2.1 Simulators 8
2.2 Empirical Studies 9
2.3 Background Information 9
2.3.1 MPEG Overview *9*
2.3.2 Network Protocols Overview *11*

3 SYSSIM SIMULATOR IMPLEMENTATION **13**

3.1 Assumptions and Analytical Models 14
3.1.1 Component Modeling *14*
3.1.2 Benchmark Modeling: Network Protocols and MPEG-1 ..... *14*
3.2 Components 16
3.2.1 Memory *16*
3.2.2 Processor *16*
3.2.3 Network Interface *17*
3.2.4 User Job *17*
3.3 Input Data to Simulator 17
3.4 Output Data from Simulator 18

4 EXPERIMENTAL RESULTS **19**

4.1 Empirical Studies 19
4.1.1 Testbed Details *19*
4.1.2 Memory Subsystem Performance *20*
4.1.3 Network Protocols Performance: TCP and UDP *23*
4.1.4 MPEG Performance *27*
4.2 Benchmarks 29
4.3 Simulator Validation 30

5 CONCLUSION AND FUTURE WORK **32**

6 REFERENCES **34**

7 LIST OF FIGURES **36**

8 LIST OF TABLES **37**

1 INTRODUCTION

The goal of this study is to investigate system bottlenecks to satisfy high bandwidth applications and how they shift from one component to another over time. The model we built emphasizes the flow of bytes in each protocol; the model imitates how typical device drivers move bytes from the network card to memory, and how typical protocol implementations move bytes around memory.

Network speeds have been increasing at a rate of over 80% a year over the last decade. 100Mb/s sec network interfaces are virtually ubiquitous, 1 Gb/s interfaces are becoming standard and inexpensive, and 10 Gb/s interfaces are already commercially available. As higher speed network data transfers have become possible high bandwidth applications are likely to become a driving factor in computer system design.

Network interfaces are not the only piece of hardware involved in high bandwidth applications; processors, disks, and memory architecture all play equally important roles. It is important to determine if these components can keep up with the increasing network speeds. Unfortunately, investigation of network-oriented system design issues is hampered by a lack of suitable simulation tools, and real world empirical evaluations can only be performed on existing systems. In order to gain insight in the performance of computer systems as individual components increase in performance at different rates, we evaluated some existing systems and built a simulator, called sysSIM that models the flow of bytes through the components of a system.

SysSIM is a discrete event driven simulator that allows us to investigate the roles and interactions of all of these components (CPU, memory, and network interfaces) in hopes of achieving insight into bottle necks.

We modeled two network protocols, UDP [12] and TCP [13] with unidirectional traffic; we modeled the behavior of TCP and UDP in the Linux kernel in terms of their interaction with the network and main memory. In the case of TCP, when the network traffic stalls, it means that the kernel buffer is full, while for UDP, network traffic will be dropped. We defined a set of benchmarks that give us different network traffic characteristics. The benchmarks are based on video streaming with various levels of compression and different functionality. To determine a reasonable set of characteristic with regard to dropped packets, out of order packets, and network overhead, we used the Iperf [10] program. To keep the analysis simple, we modeled only local area network traffic where packet loss and out-of-order delivery was virtually zero based on our Iperf results.

The rest of this report is organized as follows. The remainder of section 1 covers the sysSIM overview, the motivation behind our work, and the timeline of the various parts of this project. Section 2 deals with related work, covering both simulators and empirical studies, along with an overview of MPEG and network protocols. Section 3 covers the sysSIM simulator implementation, assumptions, analytical models, component models, benchmarks, and input & output of the simulator. Section 4 covers the experimental results from both our empirical study and the validation of sysSIM. We conclude with section 5 covers our conclusions and future work. Section 6 covers our references, section 7 our list of figures, and section 8 our list of tables.

1.1 SysSIM Overview

Figure 1 below depicts the conceptual overview of sysSIM and its various components. The five components of sysSIM are: CPU, BUS, NIC buffer, kernel buffer, and user buffer. The interaction between the various components and the performance of the various components is derived from empirical results, theoretical results, and historical trends.

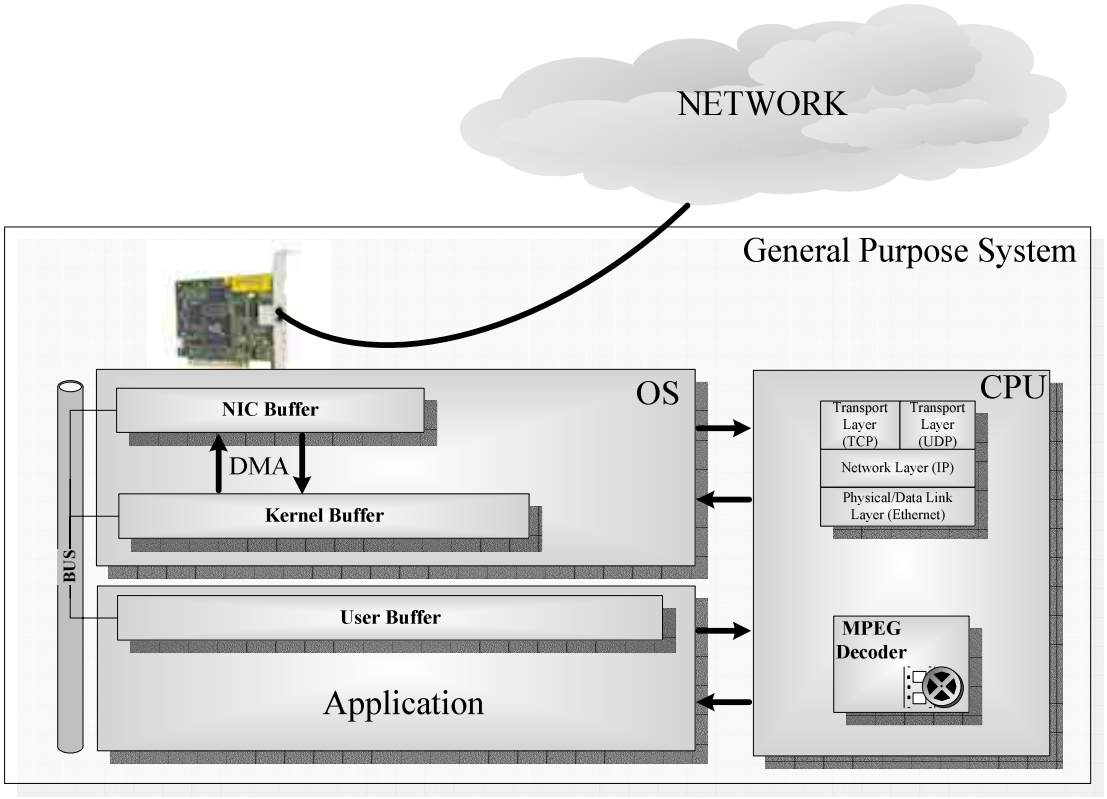

Figure 1: SysSIM simulator overview

A screen shot of sysSIM is depicted in Figure 2.

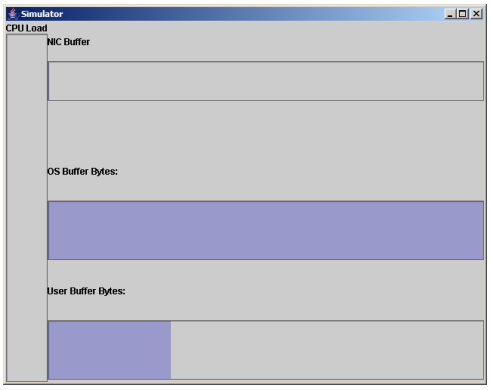

Figure 2: A screen shot of sysSIM

1.2 Motivation: Historical Trends

We collected historical data for the past 20 years regarding the performance gap between the various components. The components we investigated were: memory (latency and bandwidth), network cards (bandwidth), processors (speed and instruction latency), and communication busses (bandwidth). We extrapolate this data to predict the performance of future systems. We also use it to motivate our work.

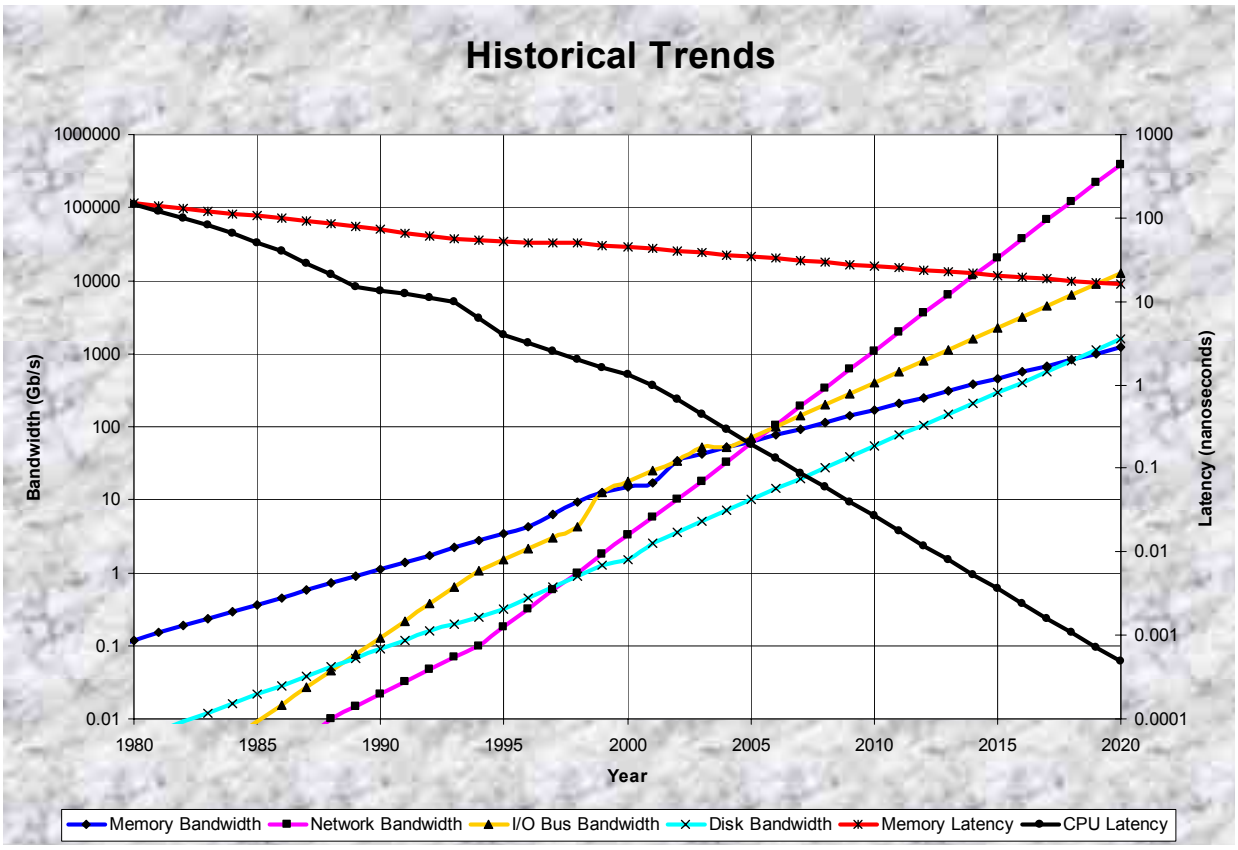

Figure 3: Historical Trends and Future Prediction

Computer technology advances at a very dramatic rate. At first sight no harm can come from that. However, not all implementation technologies change at the same pace. For many years network technology improved slowly, but the increasing importance of networking has led to a faster progress. CPU performance improves about 35% per year while memory latency improves at only 5% per year and memory bandwidth about 20% per year. Figure 3 shows the raw performance numbers for memory bandwidth, network bandwidth, I/O bus bandwidth, disk bandwidth, memory latency, and CPU latency.

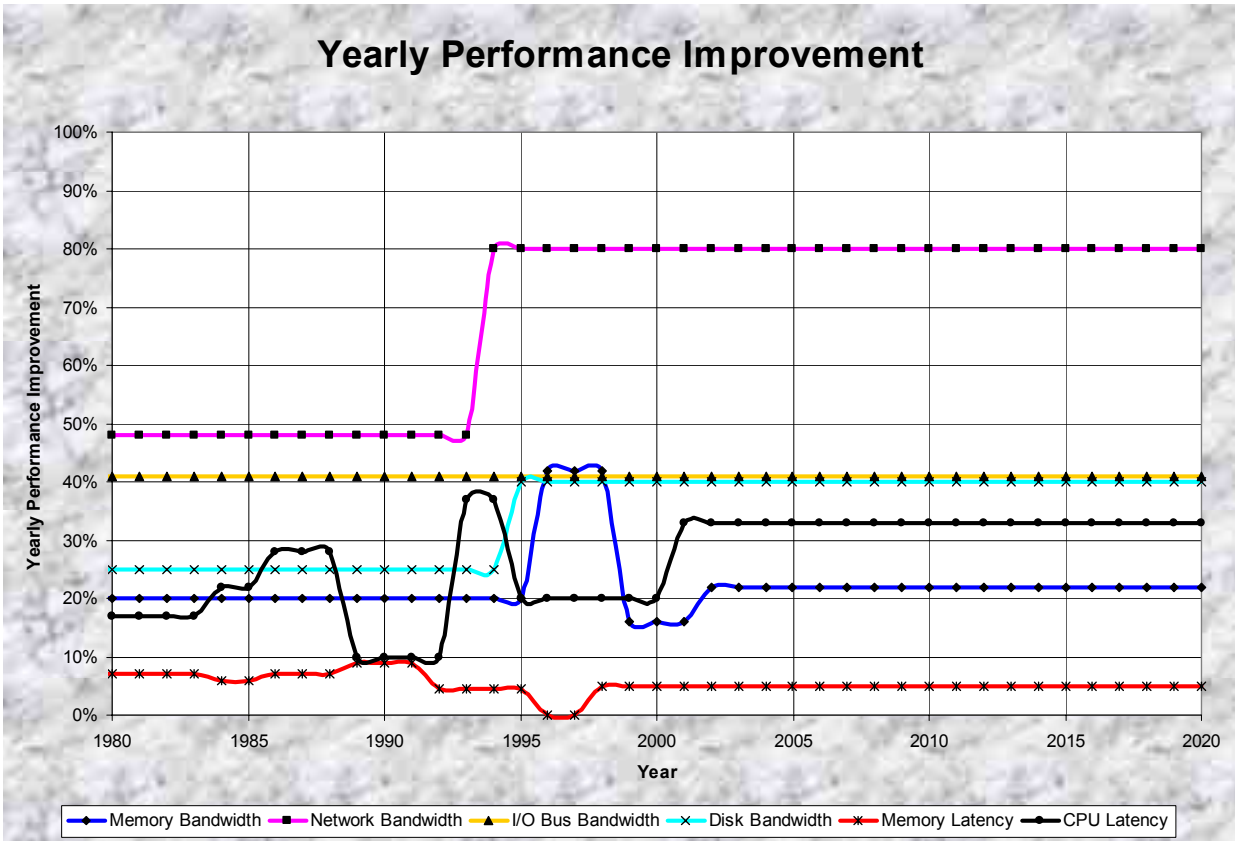

Figure 4: Historical Trends and Future Prediction: Yearly Performance Improvement

The trends show that network performance will soon exceed the performance of CPU and memory. Since the performance of the system is determined by the performance of the slowest component, there would be no reason to make faster network cards because the general purpose systems would not keep up with the network speeds. Figure 4 shows the improvement of the performance of the various components as an early percentage for a period of 40 years, 1980 to 2020. Note that up to 2004, we have actual numbers, and due to the consistent nature of the increase in performance per component over the last 20 years, we believe our extrapolation of the performance of these

components to be fairly accurate. As we can see network bandwidth have not yet exceeded memory and I/O bus bandwidth, however, as network speeds grow faster, it becomes important to predict the bottlenecks that will arise.

1.3 Timeline

The timeline of the project progress is depicted in Figure 5. We decided to include this outline in order to emphasize the various parts of the project and the amount of work that went into this project!

ID	Task Name	Start	Finish	Duration
1	Brainstorming	3/30/2004	4/19/2004	21d
2	Project Proposal	4/20/2004	4/22/2004	3d
3	**Define Benchmarks**	**4/19/2004**	**5/26/2004**	**38d**
4	High Bandwidth tests	4/19/2004	4/22/2004	4d
5	TCP and UDP network protocols	5/10/2004	5/13/2004	4d
6	MPEG-1 video streaming	5/25/2004	5/26/2004	2d
7	Run Iperf tests over LAN	4/26/2004	4/30/2004	5d
8	Test Memory Subsystem Performance	4/30/2004	5/5/2004	6d
9	Define Internal Data Structures	5/1/2004	5/10/2004	10d
10	Implement Simulator	5/13/2004	5/26/2004	14d
11	Model TCP and UDP Protocols	5/7/2004	5/13/2004	7d
12	Gather Historical Data (CPUs, Memory, Networks)	5/9/2004	5/13/2004	5d
13	Project Progress Report	5/11/2004	5/13/2004	3d
14	Test TCP and UDP protocol stack performance over loopback	5/13/2004	5/19/2004	7d
15	Test MPEG-1 Performance	5/25/2004	5/28/2004	4d
16	Gather Historical Data (Memory and System Busses)	5/24/2004	5/29/2004	6d
17	Validate Simulator	5/26/2004	5/27/2004	2d
18	Run Benchmarks	5/26/2004	5/28/2004	3d
19	Evaluate Results	5/28/2004	6/1/2004	5d
20	Presentation	5/28/2004	6/1/2004	5d
21	Final Report	5/28/2004	6/4/2004	8d

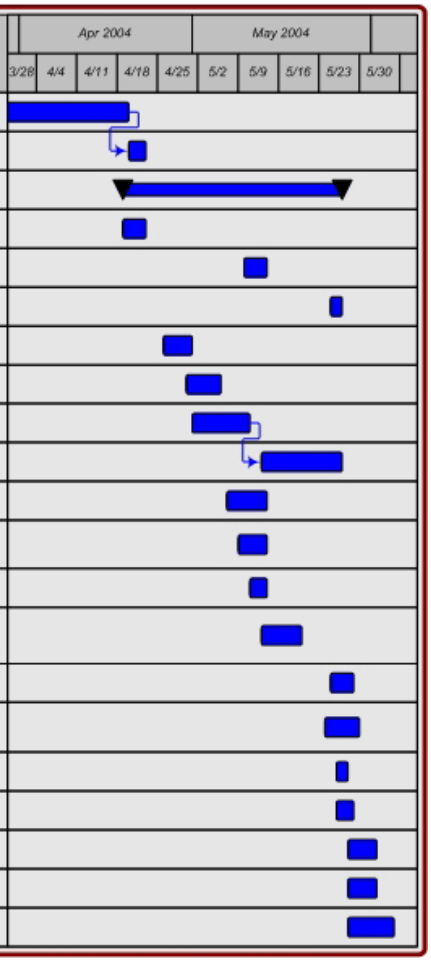

Figure 5: Timeline of project progress and participation

2 RELATED WORK

We have identified three different simulators, SimOS [2], M5 [4], and CSIM [3], that attempt to study the interaction between networks and computer system components. We found many more simulators [7, 8] that would allow us to study system component performance (not including network components) and we also found other simulators [1] that allowed us to study network performance concentrating mainly on protocol performance. We briefly describe SimOS, M5, and CSIM to better understand the extent to which the relationship between networks and computer systems has been studied.

2.1 Simulators

SimOS is a complete machine simulation environment designed for the efficient and accurate study of both uniprocessor and multiprocessor computer systems. SimOS simulates computer hardware in enough detail to boot and run commercial operating systems. SimOS currently provides CPU models of the MIPS R4000, MIPS R10000 and Digital Alpha processor families. In addition to the CPU, SimOS simulate caches, multiprocessor memory busses, disk drives, network interfaces (Ethernet), consoles, and other devices commonly found on general purpose computer systems.

M5 is a simulation system targeting network intensive workloads; it is capable of simulating multi-system networks within a single process. Within each simulated system, M5 provides a detailed performance model of the I/O subsystem, including the bus timing and coherence effects of network DMA transfers. M5 models system hardware well enough to run an unmodified commercial OS kernel.

CSIM is a parallel process and diagrams simulator. The CSIM environment consists of a set of tools for describing parallel systems, for running simulations, and for viewing simulation results. CSIM is a discrete event simulator for describing parallel processor architectures and software mappings. CSIM can describe the behavior of each type of device in a multi-device system in terms of time delays, functions, and interactions with other devices through designated ports. It can also interconnect the devices according to arbitrarily described topologies and running discrete event simulations of the described system. Of the simulators we examined, CSIM seems to match the goals of sysSIM best.

The primary disadvantage of these simulators is that they concentrate on simulating entire OSes which looses sight of the end goal to find bottlenecks due to its complexity. We have not found any work that attempted to analyze the performance of the various components of a computer system and how they perform in relation to each other with the scope of identifying bottlenecks. As we will see in the next section which discusses the empirical studies done, none of the studies examined all the components (CPU, memory and network) simultaneously.

2.2 Empirical Studies

We also found several papers which made real empirical performance measurements of the interaction between networks and system components, with Arpaci-Dusseau et al [5] being the most notable; the drawback of this work was that it attempted to examine the difference between cost and performance between uniprocessor systems, multiprocessor systems, and clusters of machines. Another paper from Mellanox Technologies does a comparative I/O analysis between InfiniBand compared with PCI-X, Fiber Channel, Gigabit Ethernet, storage over IP, HyperTransport, and RapidIO [6]; the drawback of this paper was that it only addressed the I/O bus performance characteristics. Finally, we found a useful presentation given by John R. Mashey entitled “Big Data ... and the Next Wave of InfraStress” which targets exactly the performance of computer systems as individual components increase in performance at different rates [9]. This presentation has no drawbacks other than it was done back in 1998, and it does not have many references to where the data was extracted from.

2.3 Background Information

This section covers the basic of MPEG video encoding and decoding and the basics of network protocols. The network protocols overview is essential since our study involves not just networking hardware, but also software that makes the network complete, namely the protocol stacks of TCP/IP and UDP/IP. The MPEG overview is also important to understand some of the benchmarks, namely the variable bit rate benchmarks.

2.3.1 MPEG Overview

MPEG [13] uses two types of compression methods to encode video data: inter-frame and intra-frame encoding. Inter-frame encoding is based upon both predictive coding and interpolative coding techniques. Video streams running at 30 frames per second will typically have much similarity between adjacent frames. If a motion compression method is aware of this "temporal redundancy," then only the differences in information between the

frames is encoded. This approach results in greater compression ratios, with far less data needing to be encoded. This type of inter-frame encoding is called predictive encoding.

A further reduction in data size may be achieved by the use of bi-directional prediction. Differential predictive encoding encodes only the differences between the current frame and the previous frame. Bi-directional prediction encodes the current frame based on the differences between the current, previous, and next frame of the video data. This type of inter-frame encoding is called motion-compensated interpolative encoding.

To support both inter-frame and intra-frame encoding, an MPEG data stream contains three types of coded frames:

- I-frames (intra-frame encoded)
- P-frames (predictive encoded)
- B-frames (bi-directional encoded)

Figure 6 depicts a pictorial representation of the dependence of I/P/B frames on each other. An I-frame contains a single frame of video data that does not rely on the information in any other frame to be encoded or decoded. A P-frame is constructed by predicting the difference between the current frame and closest preceding I- or P-frame. A B-frame is constructed from the two closest I- or P-frames.

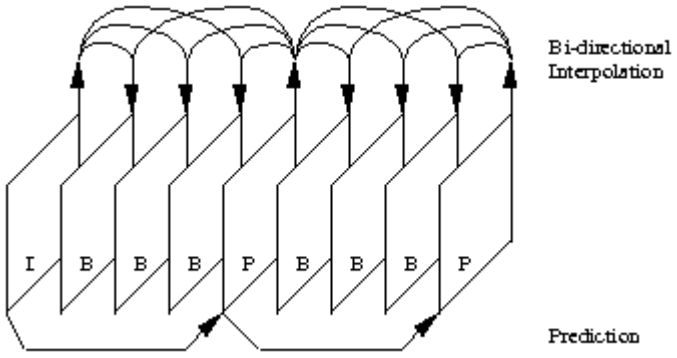

Figure 6: A pictorial depiction of the dependence of I/P/B frames on each other

A typical sequence of frames in an MPEG stream might look like IBBPBBPBBPBBPBBIBBPBBPBBPBBPBBI… where each I, B, P represents the corresponding type of frame.

2.3.2 Network Protocols Overview

To explain the structure of the TCP/IP and UDP/IP protocol stack, we present the Open System Interconnection (OSI) model and its seven layers in Figure 7.

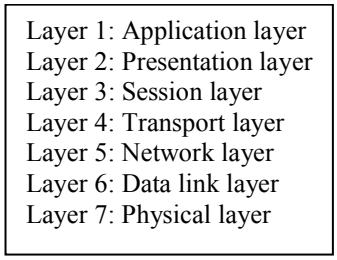

Figure 7: OSI Reference Model

The application layer usually consists of the end-user application such as a web browser, a video stream application, etc. The presentation and session layers are not used frequently in modern protocol stack implementations.

The transport layer provides reliable, transparent data transfers between senders and receivers. It provides an error recovery mechanism and flow control to throttle the sending rates. It also fragments data into smaller pieces, and passes them down to the network layer. Both TCP and UDP are found in the transport layer.

The network layer in the OSI model allows heterogeneous networks to be connected. It provides congestion control, it establishes, maintains, and tears down connections, and most important of all, it determines the route of packets transmitted. The Internet Protocol (IP) is found in the network layer. Finally, the data link and physical layer deals with the actual hardware of the network card.

The TCP/IP and UDP/IP protocol stacks can be decomposed into its three layers, the transport layer, the network layer, and the physical/data link layer. These layers and their corresponding header and payload sizes can be depicted in Figure 8 and Table 1. These numbers are important for our processing overhead computation.

Table 1: Packet breakdown and overhead incurred by header information for TCP/IP and UDP/IP

	TCP	UDP
Transport Layer (TCP/UDP) Payload	1460 bytes	1472 bytes
Transport Layer (TCP/UDP) Header	20 bytes	8 bytes
Network Layer (IP) Payload	1480 bytes	1480 bytes
Network Layer (IP) Header	20 bytes	20 bytes
Physical/Data Link (Ethernet) Layer Header	14 bytes	14 bytes
Total Ethernet MTU	1514 bytes	1514 bytes
Overhead %	3.70%	2.85%

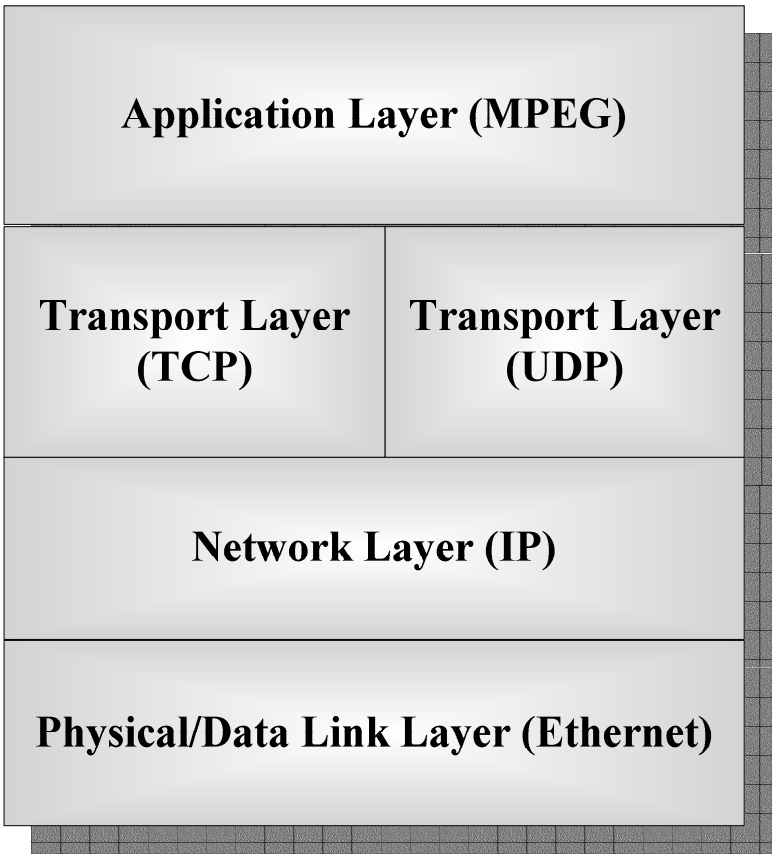

Figure 8: TCP/IP and UDP/IP protocol stack

We use the loopback address in our empirical study quite extensively, and therefore it is appropriate to define what exactly the loopback address is. The loopback address is a special IP number (127.0.0.1) that is designated for the software loopback interface of a machine. The loopback interface has no hardware associated with it, and it is not physically connected to a network. Tests over the loopback address normally will go through the corresponding protocol stack twice, once from the application layer down to the data link layer, and then back up from the data link layer to the application layer. The performance obtained over the loopback interface is primarily only bounded by the memory bandwidth and the CPU speed.

3 SYSSIM SIMULATOR IMPLEMENTATION

SysSIM was implemented in JAVA and currently has about 1500 lines of code. The simulator is light weight enough that we can run simulations in real time, but the rate of the simulation varies greatly depending on the packet size and the rate of the network speed.

The SysSIM simulator has five main components: network interface, CPU, kernel buffer memory, user buffer memory, and a user job. Each component is a software data structure that models the behavior of the physical device it represents. The components have configurable options, such as, size of data path width, clock cycle time, etc. As the simulator runs, the state of each component changes as the bytes flow from component to component. The simulator monitors the state of each device and reports both the overall performance and the performance per device.

The user code that describes the user's job is written in java and runs in its own thread. It makes simulated system calls to sysSIM when it requires network data or CPU processing. For example, the packet sizes along with the frequency of the generated packets are all configurable through java code. Since this project examines high bandwidth applications we are interested in network reads and writes. Currently, the simulator only supports network reads ; as soon as the job starts, data starts moving from the network to the network interface and then to the kernel buffer. The job determines when bytes are copied from the kernel buffer to the user buffer and when those bytes are to be freed, as well as how much CPU load will be required to process the data; furthermore, the user job can sleep if it finishes its job early. We found this feature to be necessary to properly model our benchmarks. Video streams maintain a maximum rate and if the components are moving too fast the user process must stall.

There are two modes in which the simulator can be run: a GUI and a command line interface. In the GUI interface mode, each component is displayed as a progress bar. For the memory buffers, the progress bars grows as the buffers fills. Similarly the internal network buffer displays the rate at which its buffer fills with a progress bar. The CPU workload is displayed as a vertical progress bar displaying the current workload of the processor as a percentage of its capacity. This mode is very useful as a pseudo-real time visualization tool. Among other things, it shows interesting oscillations that are easier to comprehend in real time then in a post processed graph. The command line tool runs the simulator and logs dropped packets and component stalls. Based on the logs generated,

the output is then post-processed to obtain overall throughput, video stream quality, and other important application performance characteristics. The rest of this section covers the assumptions and analytical models, component modeling, benchmark modeling, the various components of the simulator, the job profile, input to the simulator and output of the simulator.

3.1 Assumptions and Analytical Models

This sub-section describes the simplifying assumptions made in designing the simulator. The modeled characteristics of the various components and the benchmarks are discussed here as well.

3.1.1 Component Modeling

We have tested four different NICs, namely 10 Mb/s, 100 Mb/s, 1 Gb/s, and 10 Gb/s Ethernet. We assume the NIC can sustain its full bandwidth capacity and that the processor overhead will be 0% for reading/writing the raw data from the network or writing it to the network, even though there is processing overhead depending on the network protocol and packet sizes. Using Iperf [10] and the local loopback address, we computed these overheads with an empirical study.

To test the memory, we used Cache Burst 32 [12] to measure the read, write, copy and latency of the entire memory subsystem, including L1 cache, L2 cache and main memory. The results of our empirical study pertaining to the memory subsystem performance can be found in section 4.1.2.

The processor is modeled according to the other components observations in terms of processing power needed to process a certain number of bytes. For example, we have computed the byte per hertz of processing capacity for both TCP and UDP. For MPEG-1 decoding, we have also computed the byte per hertz of processing capacity for MPEG-1 video streams. Based on our empirical studies, we found that the OS consumed few resources (on the order of a few MHz) in its idle state, and hence we made the simplifying assumption that the OS does not consume any CPU resources.

3.1.2 Benchmark Modeling: Network Protocols and MPEG-1

We examined the Linux implementation source code of the TCP/IP and UDP/IP protocol stack and found that typically two copies are performed as a packet traverses the protocol stack, one from the network device to the kernel buffer, and then one from the kernel buffer to the user buffer.

We used Iperf to test TCP/UDP/IP's performance through the loopback address, which means we are only testing the protocol stack and not the NIC. The advantage of separating the testing of the network protocol stack and the NIC is that it yields the performance of the particular network protocol without dealing with an unpredictable real network. Based on the results from these tests, we obtained several important metrics, such as how long does it take for packets (of various sizes) to traverse the stack, and throughput (achieved with the limited memory bandwidth of the system and limited computing resources). These tests should give us a reasonable model regarding the network protocols and their performance on the particular hardware configuration we used for our tests. Starting with these models as input to the simulator, we validated that the simulator produces reasonable results. Since we observed a consistent trend in the performance of our testbed between the various machines, we assume that we can extrapolate the performance of future systems from the few empirical studies we made.

We found the overview of Linux Device Drivers [16] and IP Networking [17] to be a very good source of information that discusses in great detail the Linux TCP/UDP/IP protocol stack, which serves as the basis of the models we generate for both TCP and UDP. In modeling UDP, we assume that there are two copies of the network traffic in the UDP/IP protocol stack; one copy resides in kernel space, while the other resides in user space. Data not consumed fast enough from the kernel buffer is overwritten, and hence dropped packets occur. On the other hand, in modeling TCP, there are also two copies similar to UDP, but the model will be different in the fact that it requires more processing power and the fact that erroneous and lost packets must be retransmitted; based on our Iperf results, we found that no packets are lost or received out of order in the LAN environment we tested. In contrast to UDP, for TCP, user applications consuming data from memory slow down the achieved throughput, as the OS buffer gets full, the window closes and the network transfer stops and waits for acknowledgements (ACKs).

Our benchmark suite consists of applications that require high bandwidth video streaming, so it is important to gather empirical results regarding the performance of software MPEG decoding data stream characteristics. We used MPEG Analyzer [11] to gather the required metrics and characteristics.

For each benchmark, we profiled different video streams that have different compression characteristics. We also varied the kernel and user buffers to see how the performance varies as these two buffer sizes are changed. However, the most interesting benchmark is perhaps the RAW benchmark since it pushes all the components to the limit as it tries to push as much data between each component as possible.

Our benchmark suite consists of the following three tests shown in Table 2:

Table 2: The benchmark suite

Benchmark Name	Benchmark Description
MPEG_sw	Real-time (UDP) MPEG-1 with software decoding with 1:4:10 ratio between the I, P, and B frames
MPEG_hw	Real-time (UDP) MPEG-1 with hardware decoding with 1:4:10 ratio between the I, P, and B frames
RAW	Real-time (UDP) RAW video with hardware decoding

An overview of MPEG can be found in an earlier section 2.3.1.

3.2 Components

The simulator has five main components: kernel buffer memory, and user buffer memory, processor, network interface, and job profile. We discussed each component's configuration parameters in the next few sections. The system bus is implied, and is assumed to be faster than any of the five components we modeled, in which case we are assuming that the system bus will not be the bottleneck.

3.2.1 Memory

The memory is split between two main areas: the kernel buffer and the user buffer. The size of each buffer can affect the effective throughput achieved by bandwidth intensive application. At this time configurable parameters include the size of the buffers, the number of bits that can be accessed at once, and the clock speed of the bus accessing the memory. In the future, memory latency will be added to create a more comprehensive system. It was not pursued in this stage because it was determined to have little effect on the bandwidth of an entire application. Initial investigations showed that it simply added an insignificant delay to the overall runtime, since the latency was completely hidden as the pipeline between components filled up. However, further investigation is needed. Note that caches can be implemented the same way as memory, but with different performance characteristics.

3.2.2 Processor

The processor has two settings. The clock speed in Megahertz and the packets per second: that is, the number of packets that it can process per second. In our research we found a larger than expected dependency on the processor. If the network load is straining the processor too heavily, packets will either be dropped (UDP) or the network transfer will stall (TCP).

3.2.3 Network Interface

The network interface has two configurable parameters, the maximum bandwidth that it can sustain and the size of the internal buffer. This internal buffer is typically of proprietary hardware and can be accessed at least as fast as the network and memory systems. Its purpose is to allow the system to copy bytes from the device to the kernel memory since it cannot be placed there directly. These buffers are typically about 64KB in size.

3.2.4 User Job

The user job determines the size of user buffers needed at any given time, and when data will be copied from the OS buffer to the user buffer. This copy is very important because it affects the rate at which the network can proceed. In the case of UDP, if bytes are not removed fast enough they are lost, in TCP the sender will slow down its rate of transfer according to what free space remains in the OS buffer.

The job description is done in java code. The job has access to a read function, a free function, a sleep function, and a process function. These are called in a way best suited to mimic the benchmark. For example, in the MPEG video stream the process will read, then process the data it read, free the data and then sleep to ensure a constant frame rate. This process repeats until the entire movie is transferred. The user job runs in its own java thread, and calling into these functions results in a thread context switch which intentionally behaves similarly to real application making system calls and resulting in OS context switches.

3.3 Input Data to Simulator

This section covers the simulator input given. Parameters controlling the characteristics of the devices modeled by the simulator are passed in via the command line. The options are shown in Table 3.

Table 3: The input options to the simulator

Component	Description
MEMORY	Bandwidth
	Default user buffer size
	kernel buffer size
NIC	Bandwidth
	Card buffer size
CPU	processor speed
TCP / UDP	hertz per header byte of processing
MPEG	hertz per byte of decoding

The job description has the configurable characteristics shown in Table 4.

Table 4: Job description configurable characteristics

Option Name	Option Description
Protocol: UDP / TCP	Packet Size
Video Type: MPEG / RAW	Ratio of I:P:B frames
	Video resolution, bit depth, and average compression of I/P/B frames
	Video length

3.4 Output Data from Simulator

This section covers the output the simulator produces. The output of the simulator includes all the dropped packets, and which component caused them to be dropped. The current reasons for a dropped packet are:

- 100% CPU utilization
- OS Buffer is full
- NIC buffer is full

All dropped packets are time stamped based on start of transfer, which is logged in the background along with the state of the system on each dropped packet. The simulator also saves the state of each component every second regardless whether or not there were any dropped packets. The simulator is based on cycles, which get translated into nanoseconds per cycle based on the speed of the bus, memory or CPU. Table 5 shows each component logged:

Table 5: Components being logged and the metric descriptions

Component	Description of metric
OS Buffer	current amount used
	average amount used since last reported value
NIC buffer	current amount used
	average amount used since last reported value
CPU	current load
	average load since last reported value
User Buffer	current amount used
	average amount used since last reported value
NIC	Number of dropped packets since the last reported value

Based on the logs that the simulator generates, we can apply post-processing to obtain information relevant to our benchmarks, such as quality of the video stream, throughput, and latency.

4 EXPERIMENTAL RESULTS

This section covers the specific benchmarks used to test sysSIM and covers the simulation results and bottlenecks that we discovered based on our simulation results and historical trends.

4.1 Empirical Studies

This sub-section addresses the empirical study we performed to help us build the benchmarks and define the component characteristics.

4.1.1 Testbed Details

We performed an empirical analysis on a testbed consisting of three different computer systems: e55 (Intel Celeron 466 MHz), stealth (AMD K7 Athlon 1.3GHz), and diablo (AMD K7 Athlon 2.16 GHz). The empirical results allow us to validate the simulator and extrapolate the performance of future components. We used the results from this section to configure the simulator as realistically as possible as well as to define a set of benchmarks.

Table 6 depicts the hardware characteristics for the three systems we tested.

Table 6: Testbed hardware details

Computer Name	e55	stealth	diablo
Year of Manufacturing	**1999**	**2002**	**2003**
Processor Family	Intel Celeron	AMD K7 Atlon	AMD K7 Athlon
Processor Clock	466 MHz	1.3 GHz	2.16 GHz
L1 Cache Data/Code	16K / 16K	32K / 32K	32K / 32K
L2 Cache	128K	256K	256K
Front Side Bus (FSB)	Intel GTL+	DEC Alpha EV6	DEC Alpha EV6
FSB Width	64-bit	64-bit	64-bit
FSB Clock	67 MHz	520 MHz	332 MHz
FSB Bandwidth	533 MB/s	4160 MB/s	2654 MB/s
Memory Size	512 MB	512 MB	512 MB
Memory Bus Type	SDR SDRAM	SDR SDRAM	DDR SDRAM
Memory Width	64-bit	64-bit	64-bit
Memory Clock	66.6 MHz	133 MHz	332 MHz
Memory Bandwidth	533 MB/s	1064 MB/s	2654 MB/s
Chipset Bus Type	PCI	PCI	Via V-Link
Chipset Bus Width	32-bit	32-bit	8-bit
Chipset Clock	66 MHz	87 MHz	531 MHz
Chipset Bandwidth	266 MB/s	347 MB/s	531 MB/s
Network Interface Card	3-COM 10/100 PCI NIC	NetGear 10/100 PCI NIC	NetGear 10/100 PCI NIC
Operating System	Linux RedHat 9	Linux RedHat 9	Linux RedHat 9

We performed tests on each system to evaluate the memory, network interface, and CPU. We used three different utilities to make our measurements: Cache Burst 32 [12], Iperf [10], and MPEG Analyzer [11]. Cache Burst 32 tests the memory subsystem, including L1 cache, L2 cache, and main memory. Iperf was used to test the network interface card along with the local loopback tests which tested the entire ensemble as the NIC interacted with the processor and the memory. The MPEG Analyzer tested the performance of the MPEG decoding and allowed us to build our benchmarks.

4.1.2 Memory Subsystem Performance

As noted earlier, we used Cache Burst 32 to test the memory subsystem, including L1 cache, L2 cache, and main memory. The memory subsystem performance of the testbed can be found in Figure 9, Figure 10, and Figure 11, while the summary of the results can be found below in Table 7.

Table 7: Memory subsystem performance of 3 computer systems

Machine Name	e55	stealth	diablo
Processor Speed	466 MHz	1300 MHz	2166 MHz
Memory Type	SDR SDRAM @ 66MHz	SDR SDRAM @ 133 MHz	DDR SDRAM @ 332 MHz
L1 Cache Copy Bandwidth	1167 MB/s	3936 MB/s	6452 MB/s
L2 Cache Copy Bandwidth	521 MB/s	1286 MB/s	2207 MB/s
Main Memory Copy Bandwidth	117 MB/s	246 MB/s	429 MB/s
L1 Cache Latency	3 cycles	3 cycles	3 cycles
L2 Cache Latency	11 cycles	20 cycles	21 cycles
Main Memory Latency	89 cycles	223 cycles	276 cycles
L1 Cache Latency	6.4 ns	2.3 ns	1.4 ns
L2 Cache Latency	23.6 ns	15.4 ns	9.7 ns
Main Memory Latency	189.6 ns	171.1 ns	127.4 ns

In essence, we notice the bandwidth of the cache and the main memory essentially doubling from each system to the next. It is interesting to note that the latency of the cache improves similarly as the memory bandwidth, but the main memory latency decreases much slower.

The next three graphs depict the actual memory subsystem performance of the three systems; the metrics measured were copy throughput in MB/s and the latency in cycles. The latency can be converted from cycles to nanoseconds by measuring the length of a cycle (for example, for a 466MHz processor, the length of the cycle is $1/466 \times 10^6$ seconds).

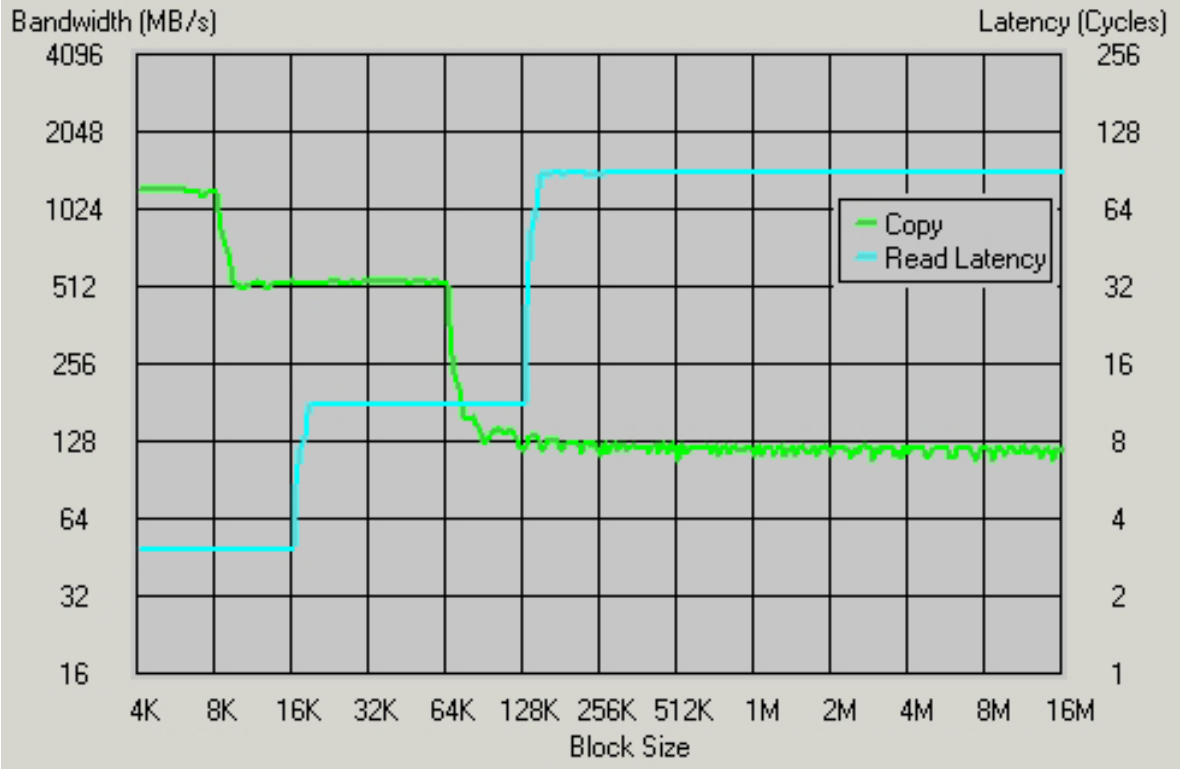

Figure 9: Memory subsystem performance of e55, a 466 MHz processor with 66 MHz SDRAM

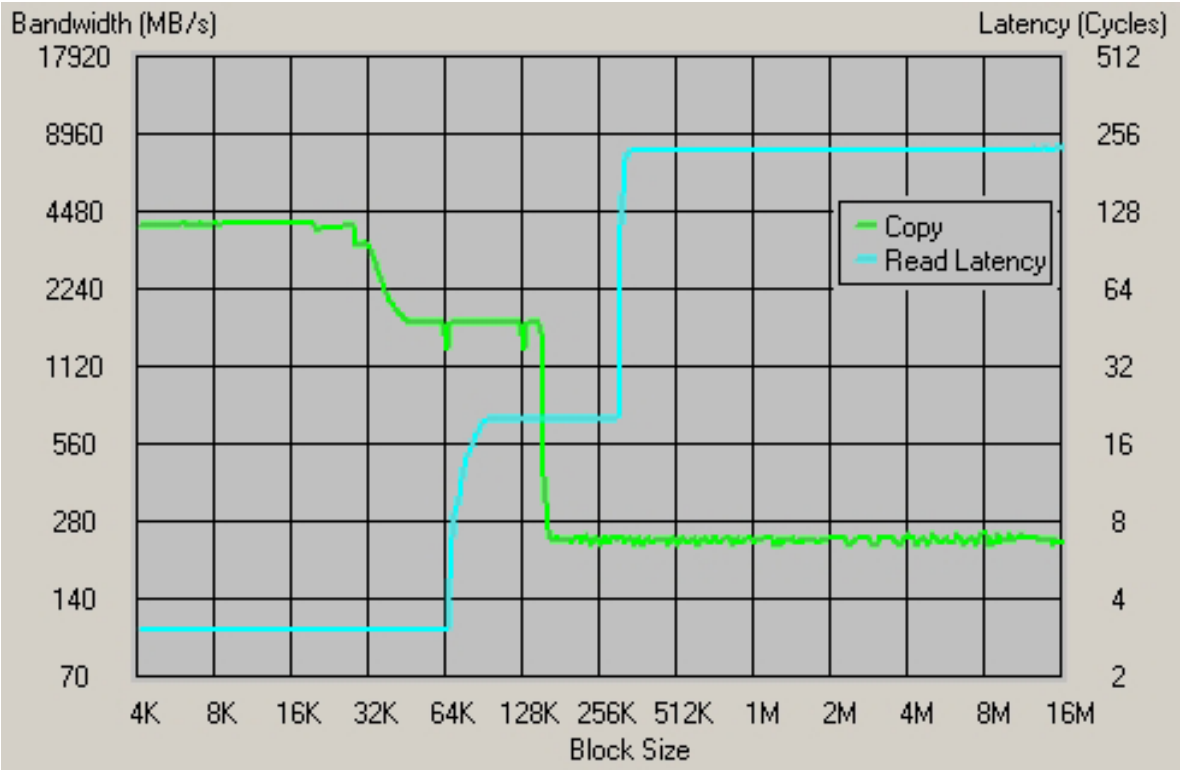

Figure 10: Memory subsystem performance of stealth, a 1.3 GHz processor with 133 MHz SDRAM

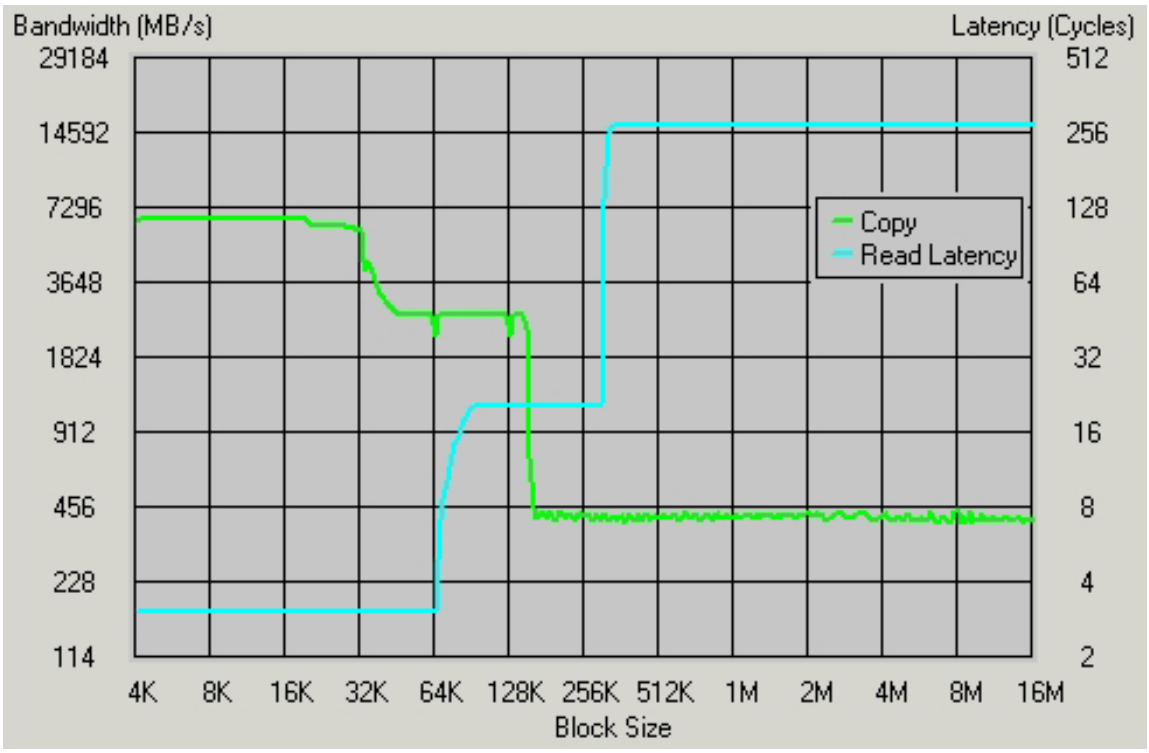

Figure 11: Memory subsystem performance of diablo, a 2.1 GHz processor with 332 MHz DDRRAM

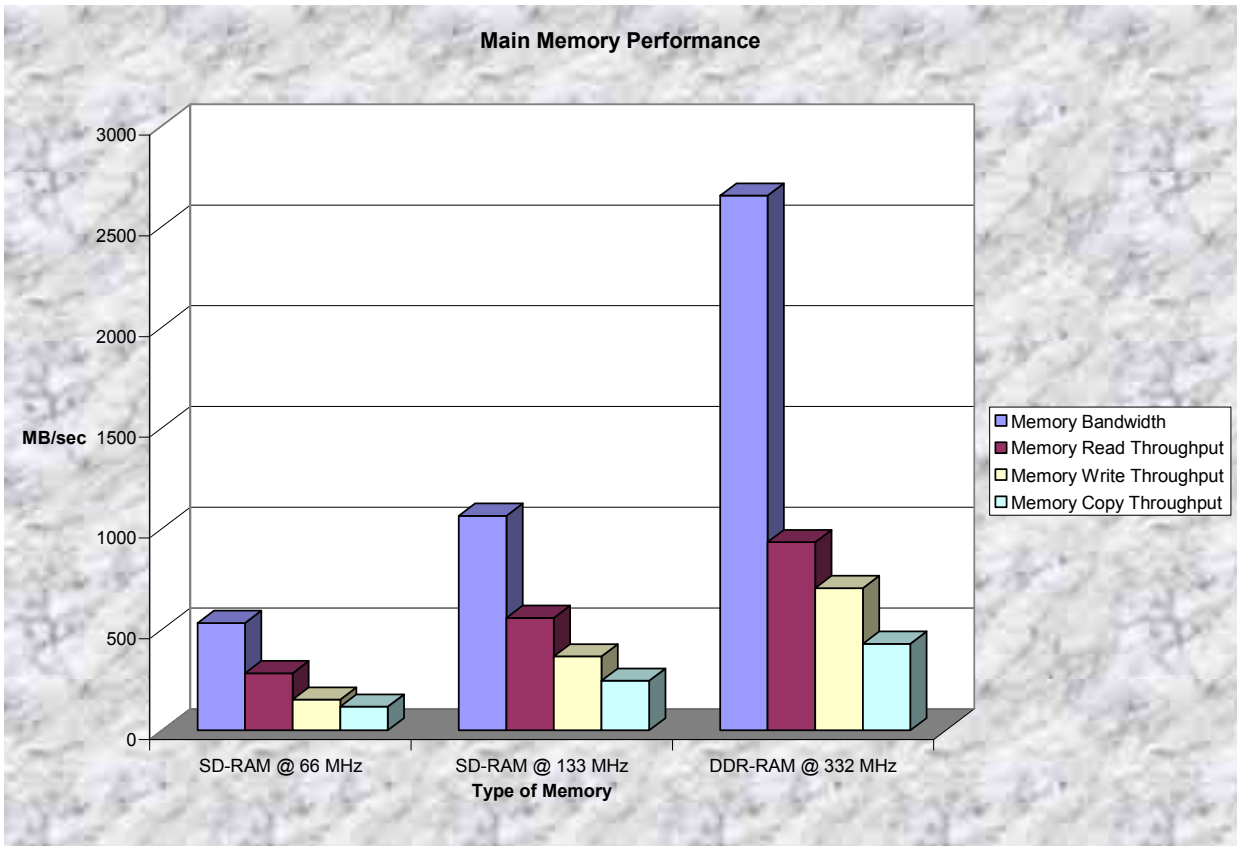

Figure 12: Main memory performance

Figure 12 shows a comparison of the advertised memory bandwidth for a particular memory type and the achieved read, write and copy throughput. Note that in the context of network protocol stacks, the most important metric from this graph is the memory copy throughput, since network buffers are copied twice, once from NIC buffer to kernel buffer, and again from kernel buffer to user buffer. It is also interesting to note the poor utilization of the memory bandwidth of only 22%, 23%, and 16% utilization for the SD-RAM @ 66 MHz, SD-RAM @ 133 MHz, and the DDR-RAM @ 332 MHz respectively.

4.1.3 Network Protocols Performance: TCP and UDP

We used Iperf to measure the performance of the TCP/IP and UDP/IP protocol stack and how they interact with the memory subsystem and the CPU. Since we did not have access to a 10Gb/s Ethernet network card interfaces, and the 1Gb/s Ethernet network card interfaces were still the bottleneck in the systems we tested (2GHz machines), we performed a series of tests over the local loopback address of the various machines in our testbed. The summary of our findings can be found in Figure 13, Figure 14, and in Table 8.

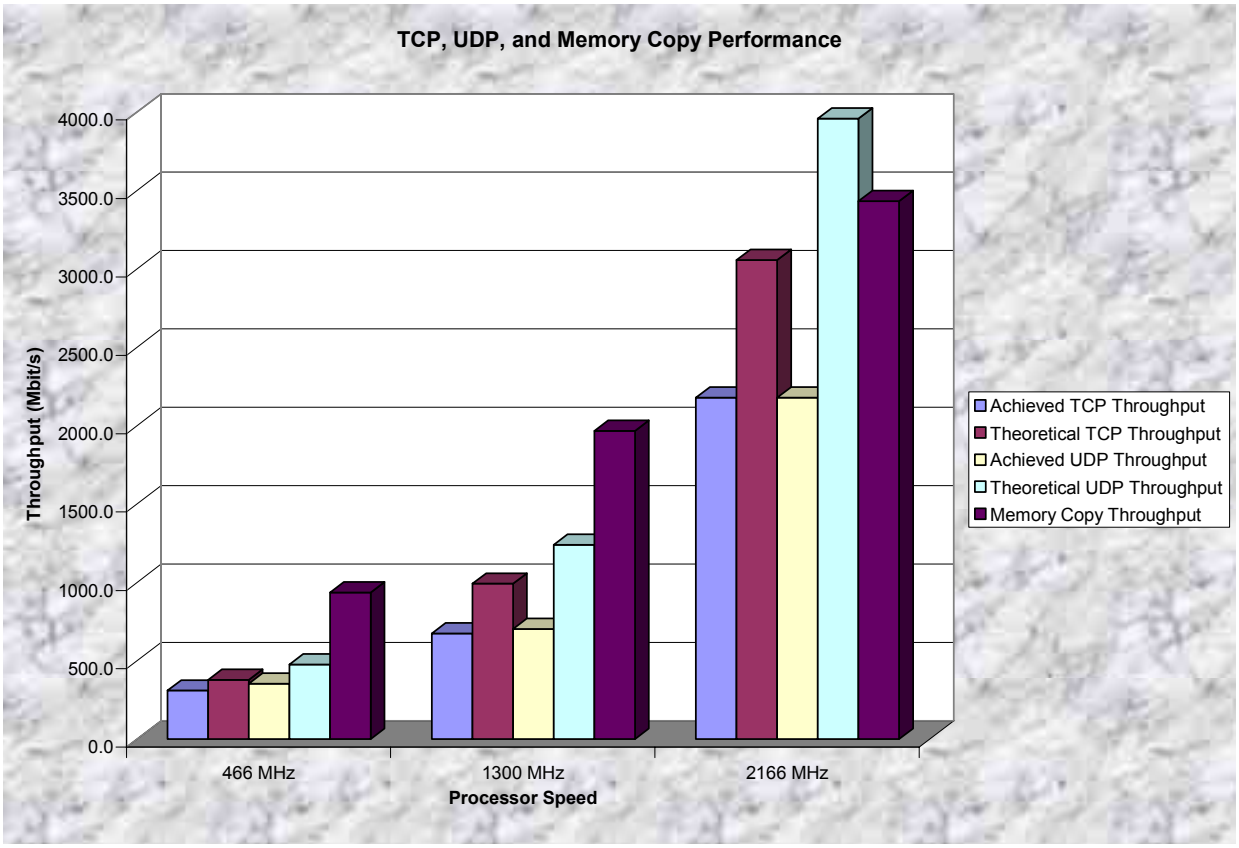

Figure 13: TCP, UDP, and Memory Copy Performance

The achieved TCP and UDP throughput (metrics from Figure 13) was actual measured throughput over the loopback address. The memory copy throughput was actual achieved throughput of main memory copy. The theoretical TCP and UDP throughput is computed based on the processing power of the processor in packets / second multiplied by the maximum size of an Ethernet frame (MTU size of 1514 bytes) minus the TCP/UDP/IP/Ethernet header. This theoretical maximum throughput represents the maximum performance in Mbit/s of the particular machine assuming a fast enough network medium (i.e. a NIC that is faster than the theoretical throughput). This implies that the achieved and theoretical throughput is constrained by the CPU processing power.

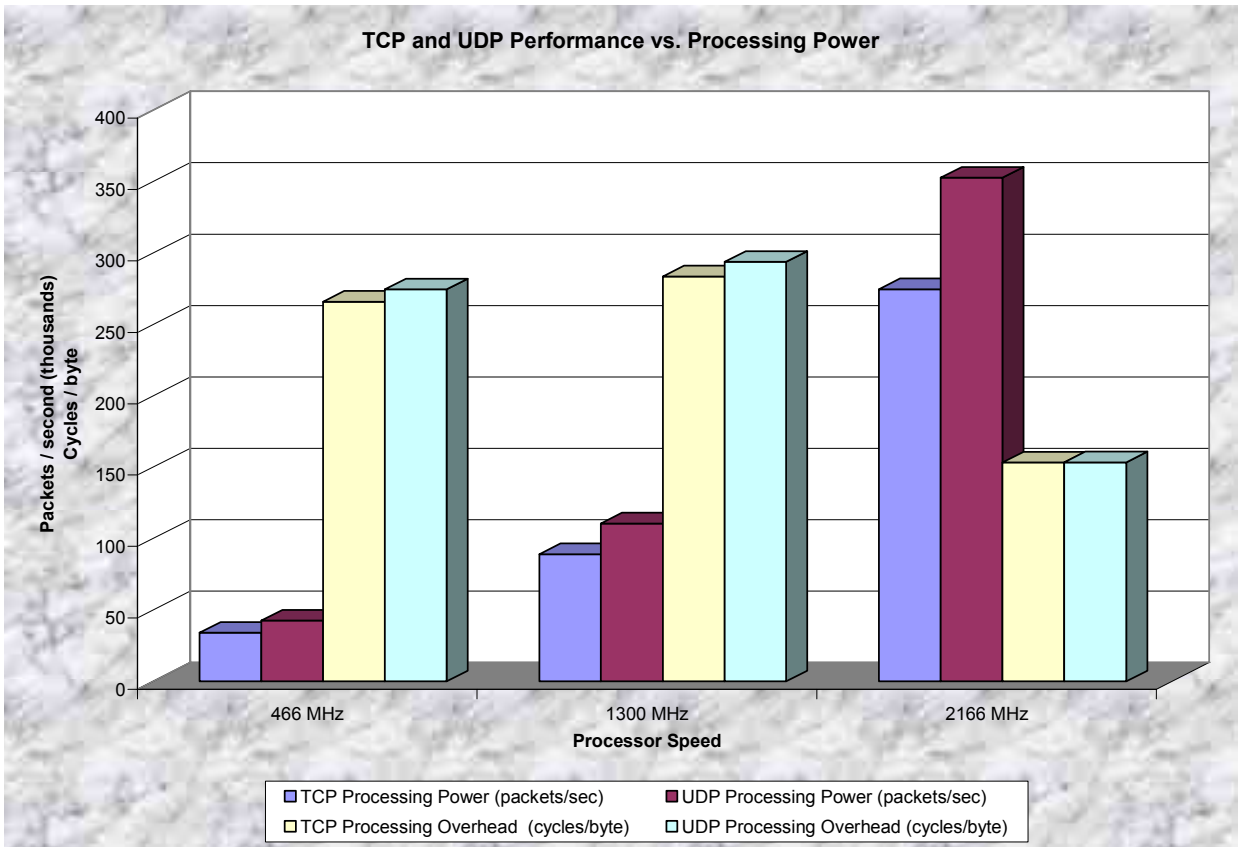

Figure 14: TCP and UDP performance vs. processing power

In Figure 14, the processing power in packets/second is computed by setting the TCP payload size to one and sending packets of minimum size (55 bytes for TCP and 43 bytes for UDP) over the loopback address as fast as possible. Since the actual payload transferred over the protocol stack is insignificant (a few hundred Kb/s), the majority of the time is spent in the protocol stack processing the TCP/UDP/IP/Ethernet frames. The number of packets per second that could be processed is in essence the power of the processor to process network packets. The cycle/byte metric summarized the number of CPU cycles needed to process a single byte of network packet header

information; note that this metric takes into consideration the processor speed, and hence if the systems are all identical, in an ideal case, the cycle/byte should remain relatively constant. The doubled efficiency of the 2.1 GHz processor in relation to the slower and older processors could be explained by the fact that many newer x86 processors are beginning to be implemented in a RISC like fashion with specialized hardware that translates complex x86 instructions into simple RISC instructions. From a personal discussion with a TCP/IP protocol developer from Sun Microsystems, we concluded that RISC architectures can process about twice as much TCP/IP traffic in comparison to x86 architectures! This would explain the nearly halved processing overhead in cycles per byte of processing.

Table 8: Performance of TCP/IP and UDP/IP protocol stacks; * denotes that the theoretical network performance numbers computed are CPU bound

Machine Name	e55	stealth	diablo
Processor Speed	466 MHz	1300 MHz	2166 MHz
TCP/IP/Ethernet Header Size	54 bytes	54 bytes	54 bytes
TCP Processing Power	34048 packets/sec	89050 packets/sec	274500 packets/sec
TCP Processing Overhead: Cycle/byte	265.8	283.5	153.2
TCP Throughput	310 Mb/s	676 Mb/s	2180 Mb/s
TCP Memory Utilization	33.12%	34.35%	63.52%
* Theoretical Maximum TCP Throughput	380 Mb/s	992 Mb/s	3058 Mb/s
* Theoretical Maximum TCP Memory Bandwidth Utilization	40.52%	50.40%	89.09%
UDP/IP/Ethernet Header Size	42 bytes	42 bytes	42 bytes
UDP Processing Power	42400 packets/sec	110500 packets/sec	352500 packets/sec
UDP Processing Overhead: Cycle/byte	274.4	293.7	153.4
UDP Throughput	354 Mb/s	704 Mb/s	2180 Mb/s
UDP Memory Utilization	37.82%	35.77%	63.52%
* Theoretical Maximum UDP Throughput	476 Mb/s	1241 Mb/s	3959 Mb/s
* Theoretical Maximum UDP Memory Bandwidth Utilization	50.87%	63.06%	115.35%
Memory Copy Bandwidth	117 MB/s	246 MB/s	429 MB/s
Memory Copy Bandwidth	936 Mb/s	1968 Mb/s	3432 Mb/s

We also used Iperf to measure the throughput achieved using TCP and UDP as a function of buffer/packet size and CPU utilization. While conducting the Iperf experiments, we also collected many performance metrics such as CPU load, and network card I/O throughput (which accounts for retransmissions on errors or lost packets, header overhead, etc…), and application I/O throughput. We found that as we increase the buffer size, we quickly approach the near optimum (well over 90% utilization of available network bandwidth running TCP for 10/100/1000 Mb/s networks) while the CPU and memory were not overloaded. The fact that memory size requirements leveled off quickly in regard to OS buffer sizes encourages the idea that a fast L2 style cache used as a stream buffer between the NIC and the CPU could significantly help memory speed maintain pace with NIC speeds while maintaining realistic cost figures. Solutions to bottlenecks will be discussed in further detail in section 5. The

performance numbers we found for TCP and UDP running over 10/100/1000 Mb/s Ethernet can be found in Figure 15 and Figure 16 below.

The performance of TCP for a range of buffer sizes from 2KB to 4MB and UDP for a range of packet sizes from 2KB to 64KB can be seen in Figure 15 and Figure 16. Note the fact that the processor is still not the bottleneck in any of these experiments. Also, the network bandwidth utilization is around 95% for all experiments with large enough network buffers.

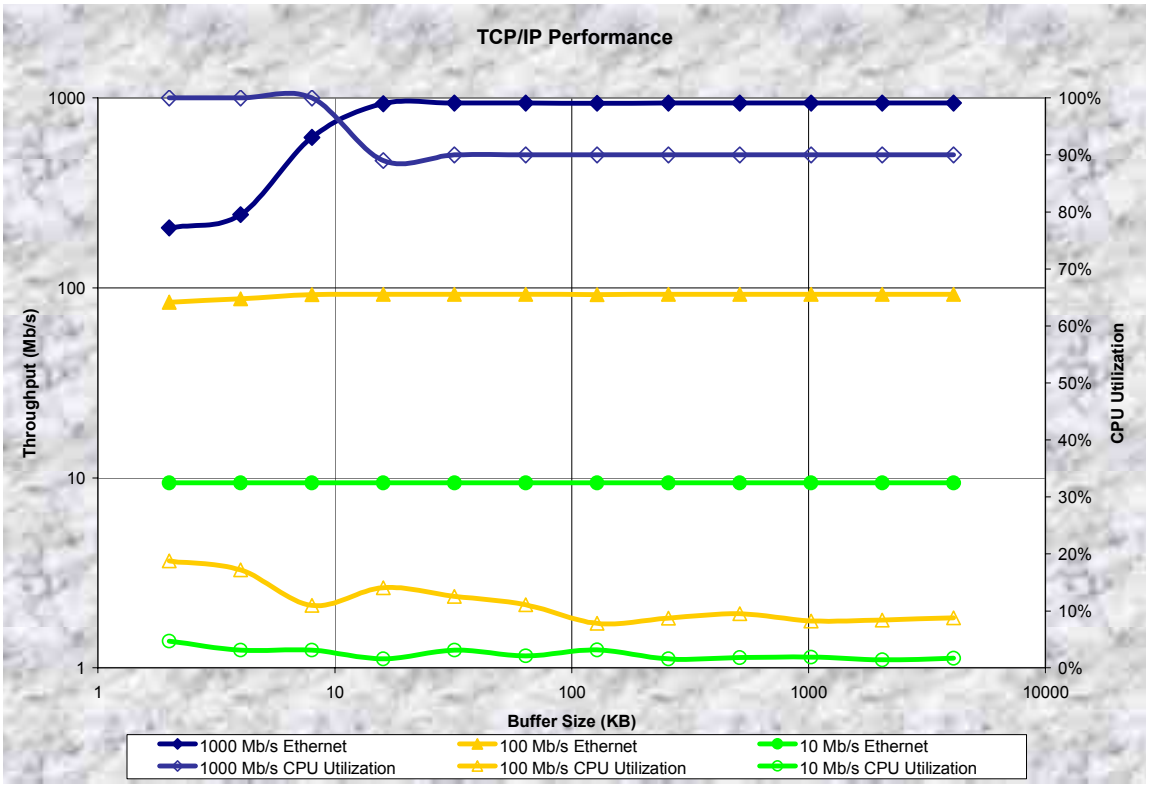

Figure 15: Performance of TCP/IP between two computer systems (2GHz CPUs) running over a LAN

It would have been very interesting to have a 10 Gb/s Ethernet NIC in order to replicate the same experiments we did with Iperf and see the achieved throughput over the network. Although the loopback tests we performed give us an overview and approximation with respect to what we should expect to obtain over the 10 Gb/s network, but nothing can substitute real tests over a real network!

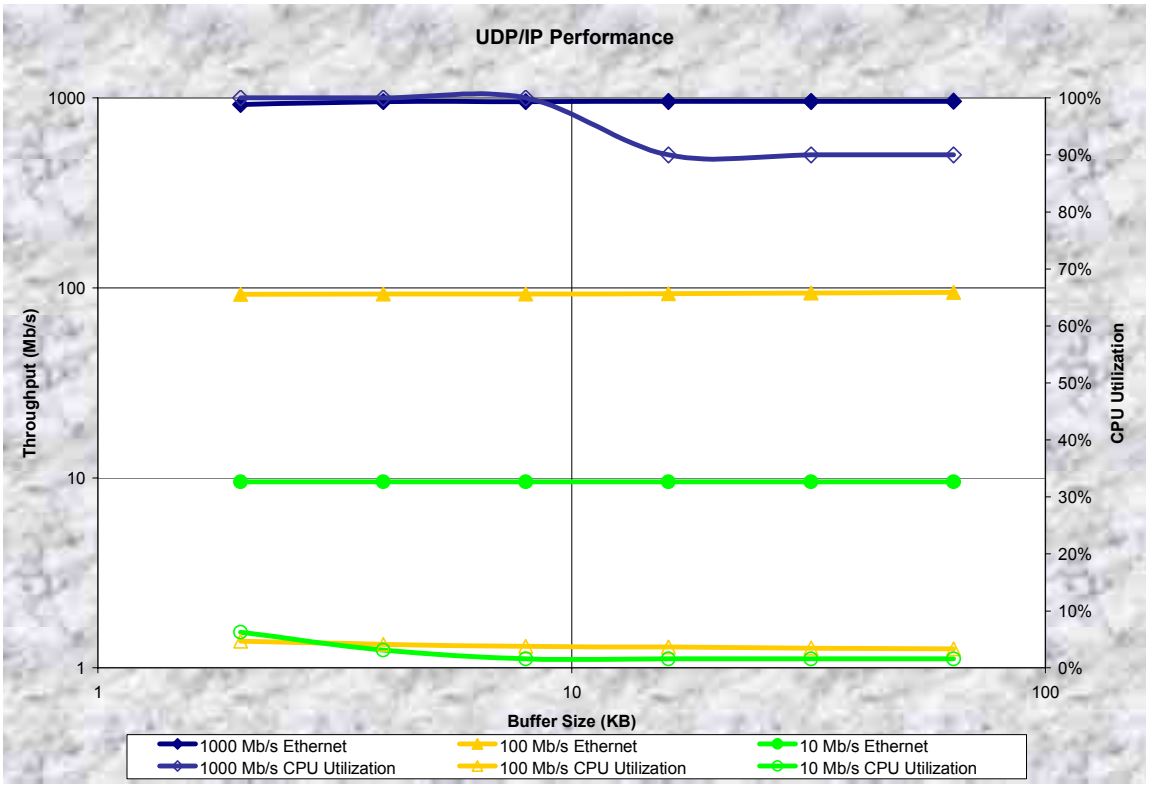

Figure 16: Performance of UDP/IP between two computer systems (2GHz CPUs) running over a LAN

4.1.4 MPEG Performance

We used the MPEG Analyzer to take measurements of various MPEG streams to aid us in building the needed benchmarks to test the simulator. Three of the four video streams we used represent the same video sequence of about 10 seconds and represents different levels of encoding and compression; the fourth video stream is a full length two hour video.

The characteristics of the video streams can be found in Table 9 below.

Table 9: MPEG-1 video clips characteristics

Video Clip Name	Width	Height	Frames / Second	Bits/pixel	Video Format	Audio Format
video1.mpg	352	288	25	24	MPEG-1	NONE
video2.mpg	720	480	29.97	24	MPEG-1	NONE
video3.mpg	720	480	29.97	24	MPEG-1	NONE
video4.mpg	720	480	29.97	24	MPEG-1	NONE

Video Clip Name	Rate (Mb/s)	Compression %	Compression Ratio	Average Frame Size (KB)	Frame #s	Length (seconds)
video1.mpg	0.9	0.16%	625	4	146985	5879.4
video2.mpg	5.7	2.41%	41	24	301	10.04
video3.mpg	9.5	4.02%	25	41	331	11.04
video4.mpg	12.5	5.29%	19	54	330	11.01

Video Clip Name	# I-frames	I-Frame size (KB)	# P-frames	P-Frame size (KB)	# B-frames	B-Frame size (KB)
video1.mpg	8743	19	41235	6	97007	2
video2.mpg	21	59	80	30	200	19
video3.mpg	23	58	88	50	220	35
video4.mpg	330	54	0	0	0	0

The decode performance is shown in Figure 17 and Table 10. The essence of our findings are that the decode performance in cycles per byte is relatively constant across all three machines. The deciding factor in the actual performance is mostly the compression ratio since more compression translated into more expensive decoding.

These four video clips were used to model the job profile as well as the processing overhead for the processor. Based on our findings, it appears that today's processors could only decode in the range of about 100 Mbit/s worth of video, which would automatically vote the processor as the bottleneck for any software decoding. A hardware decoder could be used to alleviate this bottleneck fairly easily, especially since hardware decoders are beginning to be widely available due to HDTV (High Definition Television) popularity.

Table 10: MPEG-1 video streams performance

Processor Speed	466 MHz	1300 MHz	2166 MHz	466 MHz	1300 MHz	2166 MHz
Video Clip Name	Cycles/byte	Cycles/byte	Cycles/byte	Decode time (seconds)	Decode time (seconds)	Decode time (seconds)
video1.mpg	282.2	322.2	275.8	351.9664	144.0341	74.0056
video2.mpg	195.3	181.6	191.3	3.0004	1.0003	0.6325
video3.mpg	177.0	197.5	173.5	4.9999	1.9997	1.0541
video4.mpg	162.0	150.7	158.8	5.9994	2.0005	1.2650

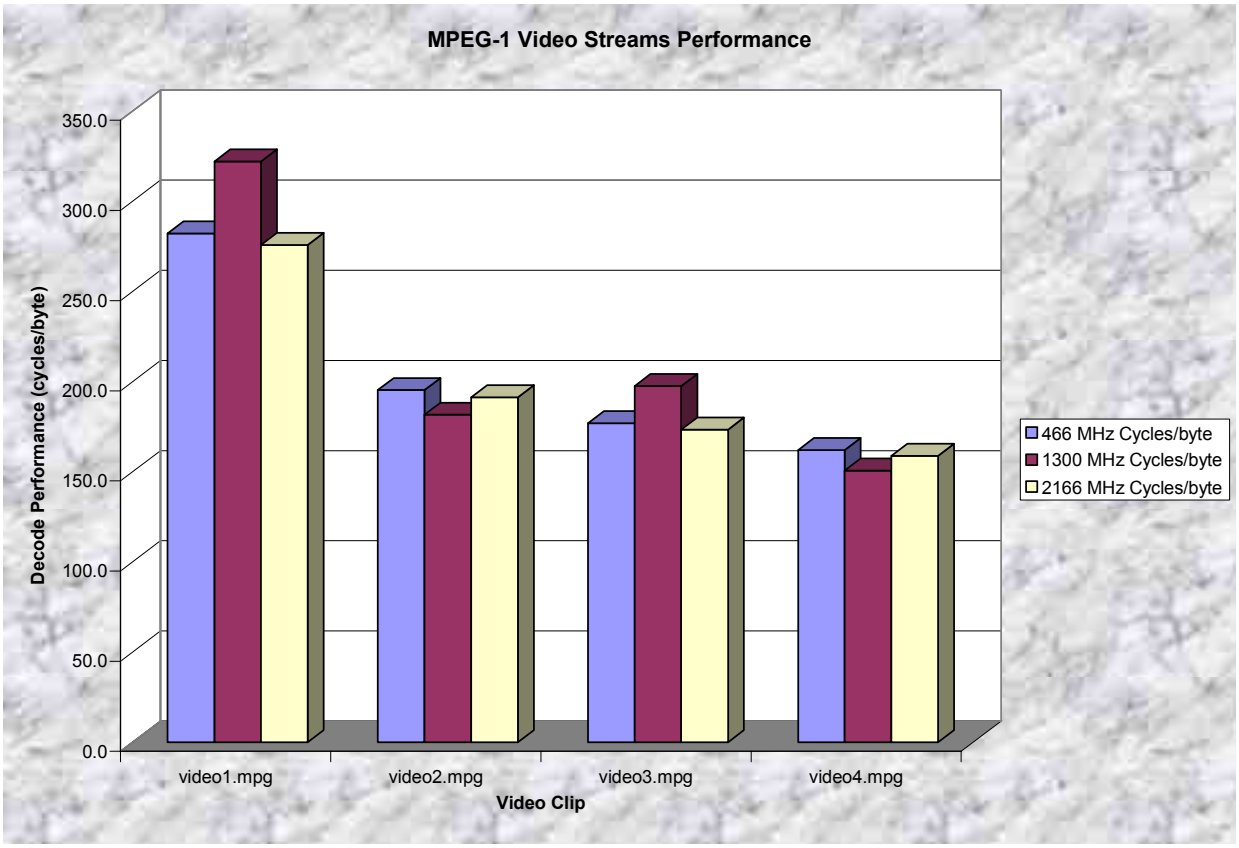

Figure 17: Decode performance measured in cycles/byte of processing

4.2 Benchmarks

This section discusses the benchmarks we used to test the simulator; the benchmarks were first introduced in section 3.1.2. Some of the issues we addressed are workload characteristics and performance metrics collected. Since we are investigating high bandwidth applications, our workload characteristics can be defined rather simply as trying to move as much data as possible between the memory and the network interface. The performance metrics collected will include effective throughput and latency incurred per component, as well as quality of the MPEG video stream, in the case of the real time tests running over UDP.

For example, assume for the MPEG_hw benchmark that the overall compression rate of 625:1 and has a 1:5:11 ratio of I:P:B frames. Assume the average I-frame is 19KB, the average P-frame is 6KB and the average B-frame is 2KB, and the overall frame rate is 25 frames / second. Note that these characteristics are those of the video1.mpg video stream. The graphical representation of the variable throughput required as various frames of different sizes are transmitted over the network can be seen in Figure 18.

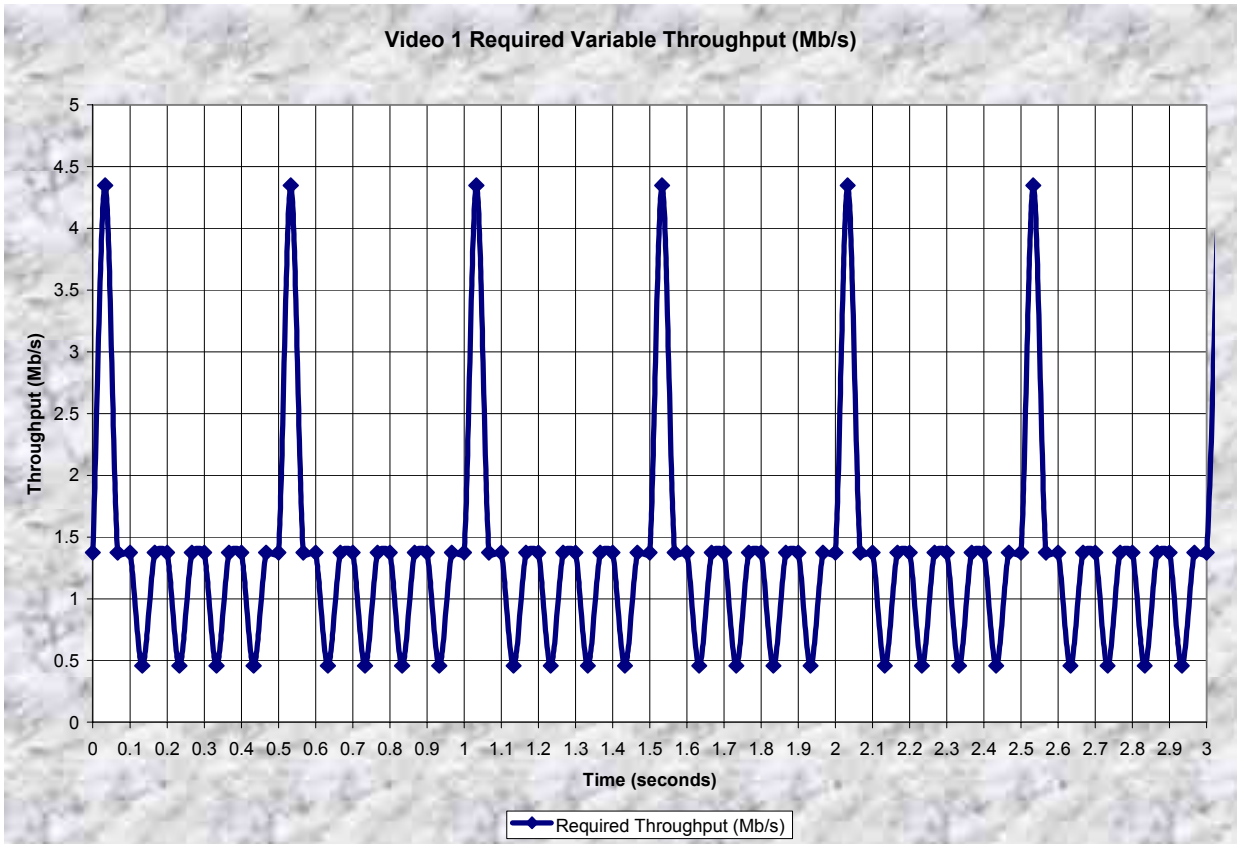

Figure 18: Video 1 required variable throughput requirement

The MPEG_sw benchmark would look identical in terms of network traffic, but would incur extra processing overhead that will take away from the network processing capacity. The last benchmark, RAW, would have a constant bit rate stream for the network throughput required and would require no processor cycles to decode the video stream.

4.3 Simulator Validation

The results obtained with sysSIM are pretty good on the three systems in our testbed. We obtained results within 6%, 18% and 20% of the actual achieved throughput for UDP for the 466 MHz, 1.3 GHz, and 2.1 GHz processors respectively. It is important to note that the results obtained from sysSIM are always lower than the theoretical throughput and lower than the memory copy throughput.

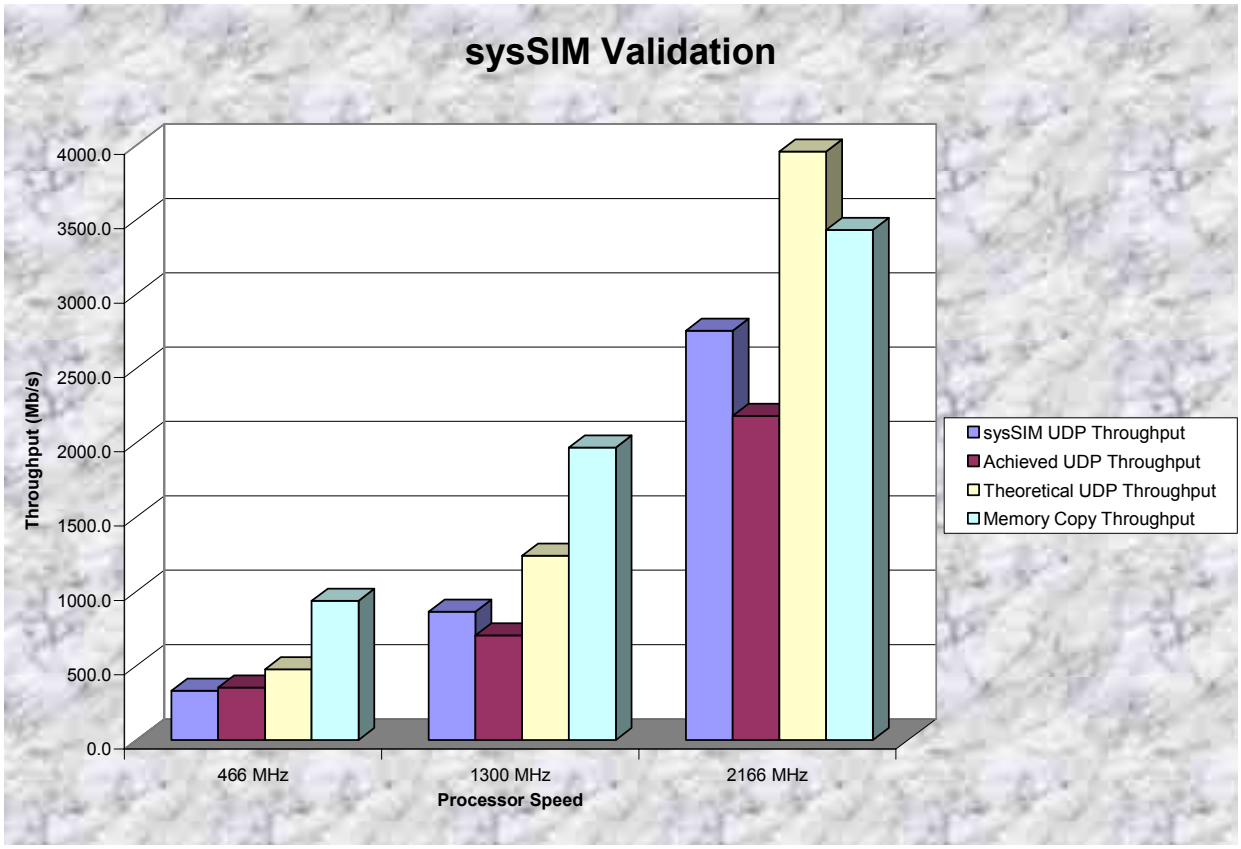

Figure 19: sysSIM validation results

5 CONCLUSION AND FUTURE WORK

We conclude with a survey of the past 25 years and with a good estimate with what the future (15 years) holds for general purpose computer systems. We have found that in the past, up to about 1995, the network was the main bottleneck. Then, between 1995 and 2007, the processor remained the bottleneck, and beyond 2007, the memory is left as the final bottleneck. Due to the steady increases in performance in all these components, there does not seem to be any change in trend, and therefore unless one of these components reaches a plateau in the future, the memory subsystem is likely to remain the bottleneck for many years to come. Figure 20 shows these trends with the year 1995 and 2007 highlighted in yellow.

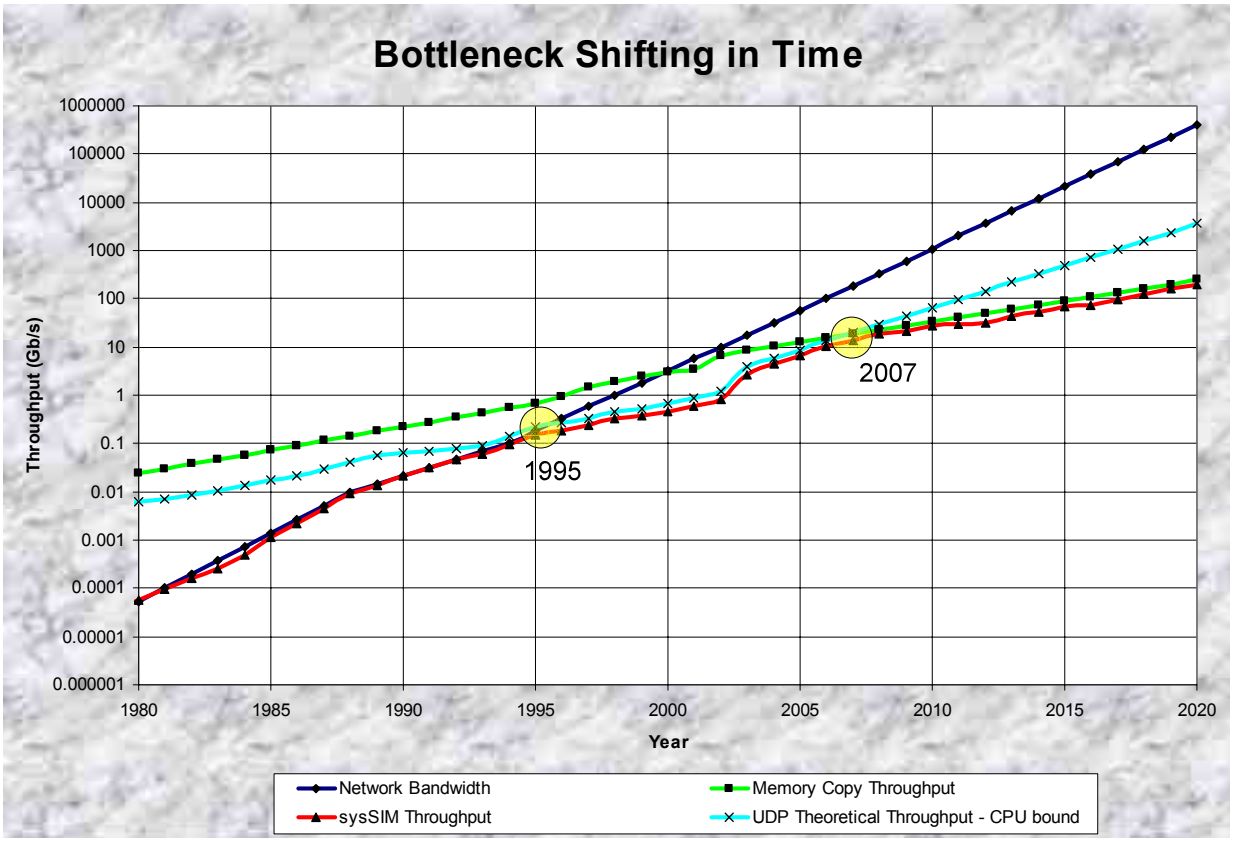

Figure 20: Bottleneck shifting in time from 1980 to 2020

The solutions we see feasible to alleviate some of the bottlenecks are depicted in Table 11.

Table 11: solutions to alleviate bottlenecks

Technical Solution	Aleviate bottleneck
Multiple memory banks	memory bandwidth
TCP offloading / Network processors	processor speed
Hardware threads	memory latency
Multiple processors (SMP)	memory latency and speed
Use high speed cache memory for buffers	memory latency and bandwidth
0-copy scheme	memory latency, bandwidth, and processor speed

As a realistic solution to lower latency and higher throughput, smaller but faster memory seems viable. From a software perspective, it is possible to reduce the number of copies of the buffers from two 2 down to zero. Where we cannot eliminate a copy, we will explore ways to keep then in faster smaller areas of memory that may be better suited for keeping up with the network interfaces. For instances where the processor is the bottleneck, we could explore various hardware solutions in which the network interface cards get more complex and smarter to offload processing from the general purpose processor. This would have a two fold advantage, both from freeing up the processor for other tasks and minimizing the memory bandwidth needed across the system bus since some packets might either be dropped or sent back out into the network without consuming resources outside of the network interface card.

6 REFERENCES

[1] S. McCanne and S. Floyd. "NS-2 Network Simulator". http://www.isi.edu/nsnam/ns/.

[2] MENDEL ROSENBLUM, EDOUARD BUGNION, SCOTT DEVINE, and STEPHEN A. HERROD. "Using the SimOS Machine Simulator to Study Complex Computer Systems." ACM Transactions on Modeling and Computer Simulation, Vol. 7, No. 1, January 1997, Pages 78–103.

[3] Carl Hein. "CSIM - Parallel Process and Diagrams Simulator", Lockheed-Martin ATL, 2004, http://www.atl.lmco.com/proj/csim/simulator/csim_doc.html.

[4] Nathan L. Binkert, Erik G. Hallnor, and Steven K. Reinhardt. "Network-Oriented Full-System Simulation using M5". Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW), Feb 2003.

[5] R. H. Arpaci-Dusseau, A. C. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, and D. A. Patterson. "The architectural costs of streaming I/O: a comparison of workstations, clusters, and SMPs," Proc. 4th Symposium on High-Performance Computer Architecture (HPCA-4), pages 90 - 101, February 1998.

[6] Mellanox Technologies. "Comparative I/O Analysis: InfiniBand Compared with PCI-X, Fiber Channel, Gigabit Ethernet, Storage over IP, HyperTransport, and RapidIO." Mellanox Technologies White Paper, 2001. http://www.mellanox.com/technology/shared/IOcompare_WP_140.pdf.

[7] Andrea Emilio Rizzoli. "A Collection of Modelling and Simulation Resources on the Internet", April 2004, http://www.idsia.ch/~andrea/simtools.html.

[8] Min Xu, Milo Martin+, Doug Burger*, and Mark Hill. "WWW Computer Architecture Page", 2004, http://www.cs.wisc.edu/~arch/www/tools.html.

[9] John R. Mashey. Big Data and the Next Wave of InfraStress Problems, Solutions, Opportunities. Invited Talk, USENIX 1999. http://www.usenix.org/events/usenix99/invited_talks/mashey.pdf.

[10] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, Kevin Gibbs. "Iperf", March 2003, http://dast.nlanr.net/Projects/Iperf/#whatis.

[11] Lawrence A. Rowe, Steve Smoot, Ketan Patel, and Brian Smith. "MPEG Video Software Statistics Gatherer." Computer Science Division-EECS, Univ. of Calif. at Berkeley, February 1st, 1995.

[12] Vladimir Afanasiev. "Cache Burst 32". October 17, 2002. http://user.rol.ru/~dxover/cburst/.

[13] A Beginner's Guide for MPEG-2 Standard. http://www.fh-friedberg.de/fachbereiche/e2/telekom-labor/zinke/mk/mpeg2beg/beginnzi.htm.

[14] J. Postel. "User Datagram Protocol", Request for Comments 768, Internet Engineering Task Force, August 1980.

[15] DARPA Internet Program. "Transmission Control Protocol", Request for Comments 793, Internet Engineering Task Force, September 1981.

[16] Alessandro Rubini & Jonathan Corbet. "Linux Device Drivers, 2nd Edition, Chapter 14, Network Drivers". June 2001.

[17] Glenn Herrin. "Linux IP Networking: A Guide to the Implementation and Modification of the Linux Protocol Stack". May 31, 2000.

7 LIST OF FIGURES

FIGURE 1: SYSSIM SIMULATOR OVERVIEW 4
FIGURE 2: A SCREEN SHOT OF SYSSIM 5
FIGURE 3: HISTORICAL TRENDS AND FUTURE PREDICTION 5
FIGURE 4: HISTORICAL TRENDS AND FUTURE PREDICTION: YEARLY PERFORMANCE IMPROVEMENT 6
FIGURE 5: TIMELINE OF PROJECT PROGRESS AND PARTICIPATION 7
FIGURE 6: A PICTORIAL DEPICTION OF THE DEPENDENCE OF I/P/B FRAMES ON EACH OTHER 10
FIGURE 7: OSI REFERENCE MODEL 11
FIGURE 8: TCP/IP AND UDP/IP PROTOCOL STACK 12
FIGURE 9: MEMORY SUBSYSTEM PERFORMANCE OF E55, A 466 MHZ PROCESSOR WITH 66 MHZ SDRAM 21
FIGURE 10: MEMORY SUBSYSTEM PERFORMANCE OF STEALTH, A 1.3 GHZ PROCESSOR WITH 133 MHZ SDRAM 21
FIGURE 11: MEMORY SUBSYSTEM PERFORMANCE OF DIABLO, A 2.1 GHZ PROCESSOR WITH 332 MHZ DDRRAM 22
FIGURE 12: MAIN MEMORY PERFORMANCE 22
FIGURE 13: TCP, UDP, AND MEMORY COPY PERFORMANCE 23
FIGURE 14: TCP AND UDP PERFORMANCE VS. PROCESSING POWER 24
FIGURE 15: PERFORMANCE OF TCP/IP BETWEEN TWO COMPUTER SYSTEMS (2GHZ CPUS) RUNNING OVER A LAN 26
FIGURE 16: PERFORMANCE OF UDP/IP BETWEEN TWO COMPUTER SYSTEMS (2GHZ CPUS) RUNNING OVER A LAN 27
FIGURE 17: DECODE PERFORMANCE MEASURED IN CYCLES/BYTE OF PROCESSING 29
FIGURE 18: VIDEO 1 REQUIRED VARIABLE THROUGHPUT REQUIREMENT 30
FIGURE 19: SYSSIM VALIDATION RESULTS 31
FIGURE 20: BOTTLENECK SHIFTING IN TIME FROM 1980 TO 2020 32

8 LIST OF TABLES

TABLE 1: PACKET BREAKDOWN AND OVERHEAD INCURRED BY HEADER INFORMATION FOR TCP/IP AND UDP/IP.....11

TABLE 2: THE BENCHMARK SUITE 16

TABLE 3: THE INPUT OPTIONS TO THE SIMULATOR 17

TABLE 4: JOB DESCRIPTION CONFIGURABLE CHARACTERISTICS 18

TABLE 5: COMPONENTS BEING LOGGED AND THE METRIC DESCRIPTIONS.....18

TABLE 6: TESTBED HARDWARE DETAILS 19

TABLE 7: MEMORY SUBSYSTEM PERFORMANCE OF 3 COMPUTER SYSTEMS 20

TABLE 8: PERFORMANCE OF TCP/IP AND UDP/IP PROTOCOL STACKS; * DENOTES THAT THE THEORETICAL NETWORK PERFORMANCE NUMBERS COMPUTED ARE CPU BOUND.....25

TABLE 9: MPEG-1 VIDEO CLIPS CHARACTERISTICS 28

TABLE 10: MPEG-1 VIDEO STREAMS PERFORMANCE.....28

TABLE 11: SOLUTIONS TO ALLEVIATE BOTTLENECKS 33