HIGH PERFORMANCE INTEGRATED NETWORK COMMUNICATIONS ARCHITECTURE - (INCA)

Klaus Schug[1], Anura Jayasumana[2] and Prasanth Gopalakrishnan[2]
[1]MiTech Incorporated, 8484 Georgia Ave Suite 950, Silver Spring, MD 20910-5604
kschug@mitechincorp.com
[2]Colorado State University, Department of Electrical Engineering, Fort Collins, CO 80523-1373
jayasuma@longs.lance.colostate.edu, prasanth@lamar.colostate.edu

ABSTRACT

Current communication subsystem mechanisms within workstation and PC class computers are limiting network communications throughput to a small percentage of the present network data rates. Even though CPU and computer network speeds have increased by more than an order of magnitude, the execution rate of computer functions and applications requiring network communications have increased only marginally. An integrated network communications architecture (INCA) is presented that is interoperable with all existing programs, computers and networks, and scales with network and CPU speeds. The INCA architecture minimizes internal system limitations and provides application level, as opposed to network interface level, internal machine data throughput rates near that of high speed network transmission rates. Test results of a software implementation of the INCA architecture on actual systems and networks are presented that show a 260% to 760% improvement in the application level throughput of network communicated data of workstation and PC class computers.

1. Introduction

In recent years, networks with media rates of 100 Megabits per second (Mbps) or more have become widely available: FastEthernet, Fiber Distributed Data Interface (FDDI), Asynchronous Transfer Mode (ATM) and Myrinet. Emerging network technologies are beginning to deliver bandwidths approaching 1 Gigabit per second (Gbps). Central processing units (CPUs) are now available in personal computer (PC) and workstation (WS) class computers that operate at 300 to 500 Megahertz (MHz) and execute more than 100 Million instructions per second (MIPS). Present internal network communication architectures are unable to provide anywhere near the application data throughput that networks can deliver and CPUs can process. The network data throughput limit of computers is the hardware limit imposed by the memory system bandwidth. Although present WS class computers have sustained data coping memory bandwidths in the 100 - 400 Mbps range [1], the present network communication architectures provide no more than several Mbps of network data to application programs. Networks can deliver data one to two orders of magnitude faster than computers can absorb, despite CPUs capable of operating at network speeds. The network and processing technology advances of recent years have shifted the network communications bottleneck to the internal computer architecture. Somewhere between the network interface and the CPU, the internal hardware and software architecture of computers is severely restricting data throughput and thereby negating network and CPU technology advances for network communication.

This paper presents the design and performance evaluation of a more efficient internal computer network communications architecture that can provide applications and distributed systems with a higher network data throughput and hence make use of high bandwidth, low latency networks and CPU technology advancements.

One difficult, but extremely useful requirement, interoperability, is added to the problem of defining a high performance, internal computer network communications architecture. Interoperability is defined as the ability to work with existing networks, communication protocols, computer systems (including operating systems and network interfaces), and application programs without requiring modification to any of these components.

As a result of the interoperability requirement, the described architecture presented is applicable to all computers connected to a network, whether the network connection is via a high speed link or via a slow speed dial up modem link. The higher the speed of the connection, the more noticeable becomes the performance improvement of the presented architecture. The remainder of section 1 includes a brief description of the major elements of computer internal network communications.

Section 2 presents the INCA design with a description of the five major elements of the high performance INCA. Section 3 details the experiment setup and performance evaluation results. Section 4 provides a summary and conclusion.

1.1. Network communication

High performance network communication can be defined as high bandwidth (high data rate), low overhead and low latency information exchange between physically separated, networked computers. High bandwidth or high data rate is almost exclusively a function of the network technology employed to connect the machines of the network. Given that the selected network technology fixes the maximum data rate, network communications performance becomes a matter of minimizing communications overhead and latency. For purposes of this paper, the time spent in the actual network hardware, in network interface units (NIUs) and in the network (e.g., transmission time in a local area network), is defined as network communications latency. The time spent by the processor sending or receiving a message is defined as network communications overhead. For purposes of this paper, latency is defined as the time required to perform the data link and physical layer functions at both the sender and receiver, and the time required to travel through the network connection media. Overhead is therefore defined as the time to perform the Network, Transport, Session, Presentation and Application layer communication functions at both sender and receiver.

1.2. Overhead versus latency

To illustrate the cost of overhead vs. latency, consider the transmission of messages that are 1000 bytes long. The communications cost of one message in milliseconds (ms), considering only communications latency, is taken as the propagation delay time required for transmission (TX) of one 1000 byte (B) message, plus one NIU time at the sender, plus one NIU time at the receiver and plus any network switch delay. Message throughput in terms of messages per second (messages/s) is calculated as one second (s) divided by the communications cost of one message. Data throughput is calculated as the number of messages/s times 1000 bytes per message, times 8 bits per byte, yielding Mbps. Network latency reduces data throughput and efficiency by 65% (Ethernet) to 32% (Myrinet), for an average cost of 49% in communications throughput or efficiency. However, the performance reduction because of communications latency costs can potentially be reduced, perhaps to 0, by performing processor computations of network application code during this delay time.

In contrast, communication overhead is made up of processor cycles that cannot be used for computation or “to perform real work.” To illustrate the relative effects of communications overhead versus communications latency, Table 1 depicts communication costs as a ratio of overhead versus latency. Communications overhead accounts for the differences in throughput and overall communications efficiency in Table 1. Communications overhead with communications (network) latency reduces data throughput and efficiency by 90% for Ethernet and to 99.6% for Myrinet. This places the cost of communications overhead plus latency at an average of 97.6% of the network transmission bandwidth or data transmission rate.

Table 1 clearly depicts communications overhead as the cause of poor network communications throughput. Communications overhead is responsible for 3.6 to 182 times the amount of communication performance degradation as is attributable to communications latency, even when latency includes Media Access Control (MAC) delays. The higher the data rate of the network, the greater the degradation due to communications overhead. This fact, combined with the physical limitations of improving communications latency and the ability to overlap latency with other computations, strongly suggests that communications overhead is the area to concentrate on for achieving efficient, fast network communications.

1.3. Internal communications architecture

Communications overhead begins at the NIU, inside the computer, and ends at the internal computer interface

Table 1. Communications Overhead versus Communications Latency Cost

	Network Technology			
	Ethernet	FDDI	ATM	Myrinet
Network TX Rate (Mbps)	10	100	155	640
Latency Data Throughput (Mbps)*	3.536	41.88	81.952	436.104
Latency + Overhead Data Throughput - Application Data Throughput (Mbps)	0.985	1.353	1.379	2.391
Overhead/Latency Cost Ratio	3.59	30.95	59.43	182.39

* Throughputs calculated and observed for typical network configurations and traffic patterns

to the application program. The hardware and software architecture through which data travels between the network and application can be defined as the internal computer network communications architecture.

The architecture can be divided into the following major software and hardware components: the NIU, the NIU interface to the internal computer system, communications protocols, the operating system (OS), the memory subsystem, the application program interface (API) and the CPU. From the previous discussion on communications latency and communications overhead, the NIU itself is considered to contribute only to communications latency and is therefore not included as a part of the INCA architecture. Since it is not a source of communications overhead and not a source of network data throughput limitation, nor likely to be either in the future, the CPU component is also not a part of the INCA architecture.

The overhead associated with internal network communications has been analyzed in a number of papers [2, 3]. Although it is difficult to compare the results because the measurements are made for different architectures, protocols, programming interfaces, and benchmarks, the conclusion is that there is no single dominating source of overhead. Implementing an efficient internal network communications architecture involves looking at the overhead impacting components of the internal communications architecture and their interaction. The research presented is therefore centered on defining and evaluating the performance of an internal computer network communications architecture that provides a more efficient software and hardware entity interaction in carrying out network communications overhead functions. Specifically, the NIU interface, protocol processing, the OS, memory management, cache operation and the API will be highlighted in the INCA architecture.

2. INCA

INCA is an all in one software architecture that combines the internal network communications functions into three major components: a computer network interface (CNI), Integrated Protocol Processing (IPP) and an API. All INCA components are available in the form of one software library for any existing application program to use with complete interoperability. INCA is a copy-once architecture because network data is copied only once before the application consumes the data. An overview of INCA is provided in Figure 1a.

The INCA is quite different from the current system architecture depicted in Figure 1b. The CNI handler contains a NIU driver to control the NIU and to set up the direct memory access (DMA) network data transfer from (for receiving) or to (for transmitting) the NIU and main memory - random access memory (RAM). In the receiving case, the CNI handler intercepts the NIU “message arrived” interrupt signal and sets up a DMA transfer to OS virtual memory space. The memory locations of the message buffers in the OS virtual memory (VM) address space are mapped into the application VM address space. No additional copy from OS to application address space is therefore required. As data becomes available in the message buffers, the CNI handler starts the IPP which currently includes the Internet protocol (IP), the user datagram protocol (UDP), the transport control protocol (TCP) and optionally, a presentation protocol - external data representation (XDR). The protocols reside in the application address space and read/write the data from the application address space VM locations to perform their processing.

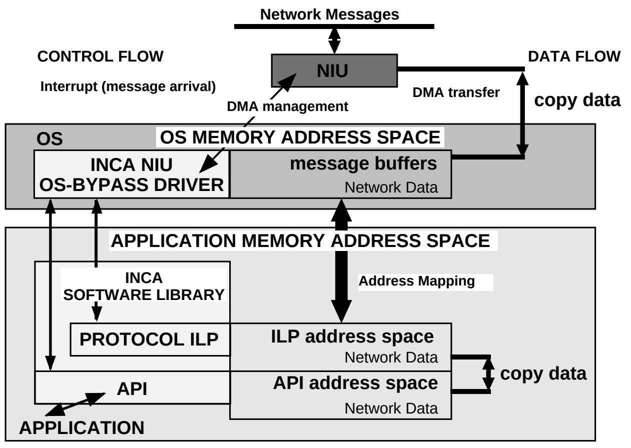

Figure 1a. INCA Network Communications

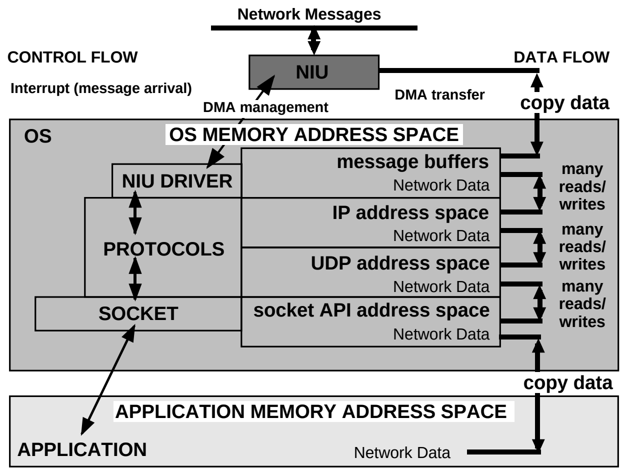

Figure 1b. Current System Network Communications

Unlike other efforts to move protocol processing into user space, such as X-kernel [4] and Mach [5], INCA makes the move to user space while retaining interoperability with existing operating systems, applications and other heavily invested in system components as well as providing a number of other performance improvements. Once the data processing is complete, the packets are demultiplexed to the corresponding protocol ports set up by the application through the use of the API calls.

2.1. CNI

The CNI functionality is accomplished via three main pieces of software, an NIU device driver, a semi OS-Bypass message handler [6, 7], and access via the INCA API. The NIU device driver is a software set of C programming language functions that support direct access to the network card. The driver also supports very low level communication primitives in which network packet reception and transmission can be tested and measured. The NIU driver has been implemented as a loadable module in the OS. The driver has features to support semi OS-Bypass and direct DMA transfer to the user space.

When the NIU signals via a hardware interrupt that a message has arrived, the INCA NIU driver sets up the DMA transfer from the NIU to the OS address space message buffer. In addition, the driver maps the OS message buffer addresses to the application making use of the INCA network communication library routines, which include the driver. The DMA network data transfer is accomplished with the message data buffers pointing to the application address space. Finally, the driver sets up an endpoint mechanism for data reception and transfer. The endpoint mechanism provides the user application with direct access to the NIU and for data reception and transfer. These endpoints are again created in the driver OS memory space and mapped on to the user address application memory space. Adding virtual memory buffer management functions to the CNI semi OS-Bypass handler would require changes in some or all of the internal network communications subsystem components such as the Commercial-off-the-shelf (COTS) OSs and NIUs.

To distinguish existing OS system calls from those that access the INCA OS replacement functions, the CNI driver and semi OS-Bypass functions are accessed via INCA library calls with different names from the OS system calls. One of these calls is used in place of endpoint tags to provide multiplexing of the incoming channels to different applications or different instantiations of the same application. This implements an OS-Bypass channel for network data routing. For existing applications that do not wish to modify their system calls, INCA allows the traditional system interfaces to applications (i.e., sockets).

2.2. Integrated protocol processing

IPP is an extension of integrated layer processing (ILP) to: 1) include protocols above the transport (e.g., UDP, TCP, TP0-5) layer, including the session, presentation (e.g., XDR) and optionally, the application layer and 2) integrate the ILP loop into the CNI message handler. The IPP INCA component performs protocol processing without invoking the OS resident protocols.

Earlier ILP research implementations [8, 9, 10] did not integrate ILP with a semi OS-Bypass message handler, did not add session and higher layers to the ILP loop, nor was ILP always performed in user application address space. Protocol execution inside the existing OSs and under the control of the OS is not used by INCA's IPP component. Modification of the OS to improve protocol processing efficiency would require changes that would violate interoperability. The purpose of any ILP based technique such as IPP, is to improve throughput performance by integrating protocol processing to eliminate one load and one store, a read and write, of every machine word (e.g., four to eight) bytes of network message data. To this performance goal, this research effort added the requirement for interoperability, requiring the IPP loops to use existing protocols and requiring INCA software libraries to accommodate the integration of all existing protocols into the IPP CNI message handler.

The interoperability requirement prevents integrating applications with the INCA IPP portion of the CNI message handler. Interoperability with existing protocols and integration introduces some amount of serial processing into an ILP loop in order to satisfy protocol processing ordering constraints. INCA executes protocols in three stages. The initial stages of a series of layers are executed serially, then the integrated data manipulations take place in one shared stage and then the final stages are executed serially. Message processing tasks are executed in the appropriate stages to satisfy the ordering constraints. Header processing is assigned to the initial stage. Data manipulation is assigned to the integrated stage. Header processing for sending and external behavior (e.g., error handling) are assigned to the final stage. Executing the tasks in the appropriate stages ensures that the external constraints protocols impose on each other cannot conflict with their internal constraints. The ILP loop has been implemented to start up directly after reception of data into the user message buffer. It does the integrated checksumming on data in the initial stage, protocol data manipulations and byte order conversions in the middle integrated stage, and flow control and error handling in the final stage.

The concept of delayed checksumming has been included in the loop. In the case of IP fragments, the checksumming is done only after reassembly.

Checksumming processing has been sped up in the INCA ILP loop through the use of a checksum routine optimized for reduced instruction set (RISC) CPUs. Once the data processing is complete, the packets are demultiplexed to the corresponding protocol ports set up by the application.

The INCA IPP protocol library consists of C computer language functions that implement the ILP loop and other pieces of the protocol control setting such as fragmentation, and in the case of TCP, maintaining the window, setting up time-out and retransmission, etc.

2.3. Memory management

INCA performs memory management three ways, by eliminating most network data copying, allowing the OS to perform virtual memory management, and performing its own message buffer management. The goal of memory management is to reduce data copying. INCA's CNI semi OS-Bypass message handler with embedded ILP loop protocol processing provides an integrated data path to achieve minimal data copying. The CNI, protocol processing, network message buffer management and API are integrated into one user level library routine. Network data is accessible for protocol processing without the current system data copies from the OS message buffer space to the protocol memory space and without the numerous reads and writes within the protocol processing entities. In the ILP loop, all protocols are executed before the data is written to another memory location. At the completion of the ILP loop, the data is written to the "incabuffers" structure in the application address space. The application accesses the data in the "incabuffers" and since the data is already in the program's address space, no copying of the data is required.

The CNI must be able to translate the virtual addresses to physical addresses, and for interoperability, the translations must be coordinated with the OS's VM subsystem. INCA has the OS perform VM management through the use of the UNIX mmap() or equivalent OS function. To enable the OS to perform VM management and address translation, the INCA driver must allocate message buffers in the OS address space initially as well as in the application address space to allow the OS to make the required mappings and perform its virtual memory management functions. Any other approach besides OS assisted VM management, e.g., NIU VM management or translation look-aside buffer (TLB) handler incorporation into the NIU, would seriously violate interoperability.

Message buffer management is performed by the CNI semi OS-Bypass handler. The CNI has complete control over the size, location, and alignment of physical buffers. In connections to networks such as FDDI or ATM that use message sizes of 4096 kilobytes (KB) and 53 bytes, respectively, then INCA's buffer manager can select an integer multiple of the network message size for the size of the message buffers. Each message is aligned to the beginning of message buffers with a single message per buffer for messages smaller than a chosen buffer size. For messages larger than the chosen buffer size, multiple buffers are used beginning with the first and intermediate buffers filled from start to end. The application specifies message buffers using offsets into the buffer region, which the network interface can easily bound-check and translate.

2.4. Cache operation

While caches are very effective in reducing memory accesses for many computations, the characteristics of strictly layered message processing are such that caching is not as effective, mainly because of poor data and instruction locality [1]. The cost of a cache miss tends to increase as processor speeds increase. The trend in CPU design to execute multiple instructions in a single cycle effectively increases the cost even more.

INCA increases cache effectiveness by increasing the cache hit ratio for network communications data processing through a number of methods inherent in the use of a single data copy, semi OS-Bypass handler CNI with integrated ILP. INCA's architecture and processing patterns reduce the cache performance degradation of four of the five main factors influencing cache performance: context switches, DMA transfers, TLB misses, self interference and not fitting into the cache. Each word is loaded and stored only once in the ILP loop, even though it is manipulated a number of times. Integration of protocols eliminates the load and store instructions themselves. This saves the time to transfer data between memory and the cache and saves the time to load the data from the cache and store data to the cache or a write buffer. Since there are fewer load and store instructions and they occur in an inner loop containing more instructions, the delay after a cache load is more likely to be overlapped with additional computations from the loop, and on machines with write-through caches and write buffers, stores are less likely to fill a write buffer. Finally, integration eliminates some buffer allocations and deallocations since data which would have been buffered between data manipulations is now streamed directly from one to the next. The result is that INCA provides a substantial improvement on cache operation by lowering the memory cycles per CPU instruction (mCPI).

2.5. API

The application program interface provides the connection between the network communications subsystem and the application programs. Since INCA bypasses some of the current system network communications subsystem functions, a new API is required to use the INCA library routines for high performance networking. Once again, the interoperability requirement dominates the design tradeoffs and implementation options.

The INCA API allows the application to: open a network by opening a NIU device driver, specify parameters to the NIU device driver, specify the protocols and their options, and set the characteristics of the data transfer through the OS-Bypass channel. The INCA API consists of alternative system calls for the current OS system calls such as socket(), connect() and listen(). The INCA API looks very much like the current UNIX OS API set of "open()", "read()", "write()" and "close()" named function calls. But these functions are not system calls, but rather are INCA specific functions that will implement ILP and semi OS-Bypass as a user level communication primitive that is transparent to the user.

INCA can incorporate all the application, presentation and session (replacement for socket and streams calls) functions into the IPP loop of the semi OS-Bypass message handler. In current systems, the application programs many times contain the application layer protocol (e.g., hypertext transport protocol - HTTP) and the presentation layer functions (e.g., XDR). Only the socket or streams system calls are the API. Placing all protocol functions into INCA's IPP loop would have the benefits of not requiring application program upgrades because of changes in the application, presentation and session layers, and the data throughput would be increased in these applications. In the future, incorporation of the higher layer protocols can be accomplished dynamically at run time through application selection of the protocol stack configuration. INCA's API can be located anywhere between the application and any protocol of the protocol stack.

3. Experiment setup and results

The INCA testbed consisted of a SUN Microsystems UltraSPARC1 WS with a 143 MHz UltraSPARC1 CPU, 64 megabytes (MB) of RAM, 512 KB of level 2 cache, the Solaris 2.5.1 OS (also known as SUN OS 5.5.1), with a SUN SBus FastEthernet Adapter 2.0 NIU. The UltraSPARC1 was connected via a private (no other machines or hubs) 100 Mbps FastEthernet to a Gateway 2000 PC. The two machines were connected with a dedicated 100 Mbps twisted pair category – 5 cable. The Gateway PC was configured with a 167 MHz Pentium Pro CPU, 32 MB of RAM, 256 KB of level 2 cache, the LINUX 2.0.29 OS, and with a 3Com FastEtherlink (parallel tasking peripheral component interconnect (PCI) bus 10/100Base-T) FastEthernet NIU. This test configuration uses actual machines, OSs, NIUs and networks found in many computing environments. The results should therefore have wide applicability.

3.1. Experiment setup

A number of experiments were performed using different size messages transmitted between the SUN WS and the Gateway PC. The only application running on the PC was the test application, the World Wide Web (WWW) Hypertext Markup Language (HTML) browser Chimera from the French national research institute INRIA. The SUN UltraSPARC1 was running the normal set of OS background processes and the INCA message generation application.

For measuring current PC and LINUX system performance, software probes were inserted in three locations: 1) in the LINUX FastEthernet card driver, 2) at the start of protocol processing and 3) at the point where the application began reading the data. The probes wrote the system time to memory when they were executed during the course of network message handling.

For INCA measurements, the INCA library was executed, including the FastEthernet driver, CNI handler and the IPP loop with standard, full implementations of IP, UDP, TCP and optionally XDR functions, all external and separate from the implementations in the OS. Probes were also placed in the same three locations: in the INCA FastEthernet card driver, at the start of the ILP loop and at the point where the application began reading the data. By subtracting the times of the probes, throughput measurements were made.

An INCA message generation program transmitted messages over the FastEthernet at various message sizes ranging from 10 bytes to 65,512 bytes. Messages larger than this were not considered in order to avoid violating interoperability with existing protocol implementations. Messages in the range of 10 to 200 bytes are given more emphasis since 89% of UDP and 99% of TCP network messages are below 200 bytes [11]. Each test at each message size was repeated six times and the lowest times, best throughput, for both INCA and the current system configurations were used for analysis of the results.

3.2. Measurement tools and software

The INCA experiments were conducted using the freely available "ttcp" software used to measure the network throughput between two nodes over a network. Many modification were done to the "ttcp" software to make it work with Solaris and LINUX. A separate test program was written similar to the "ttcp" software to run with the INCA library. The timing measurements were made using the function CLOCK_GETTIME library function in the portable operating system interface (POSIX) 4 library. This timing function is able to provide nanosecond time stamping. Microsecond resolution timers were available for timing measurements on the PC platform. The measurements were taken using a client-server test bench. In this model, a transmitter and a receiver were defined and data was transmitted by the client using the specified protocol and received by the server routine.

3.3. Results

The performance of the INCA network communications architecture running on current systems, the SUN UltraSPARC1/Solaris 2.5.1 and Gateway 2000/LINUX 2.0, is compared to the network communications architecture of the same systems without INCA.

3.3.1. INCA versus SUN UltraSPARC1/Solaris. Table 2 and Figure 2 depict the results of the INCA implementation on the SUN WS. The percent (%) gain with INCA is expressed as the percentage of the current system throughput that INCA achieves. Since 99% of TCP and 89% of UDP messages are below 200 bytes in size [11], the region of interest is up to and below the 200 byte message size. INCA increases SUN WS UDP Throughput by 760% at message sizes between 10 to 40 Bytes. Both Table 2 and Figure 2 show that INCA clearly provides significant performance improvements while maintaining interoperability.

Table 2. INCA Implementation Increases Solaris Network Data Throughput by up to 570%

INCA THROUGHPUT PERFORMANCE ON SUN ULTRASPARC1					
	SOLARIS 2.5.1		INCA		
Msg. Size (Bytes)	Proc. Time (nsec)	Thruput (Bytes /Sec)	Proc. Time (nsec)	Thruput (Bytes /Sec)	Gain (%)
TCP					
50	168384	296940	48781	1024989	345
100	170285	587251	49452	2022163	344
200	174873	1143687	49900	4008016	**350**
1000	182480	5480053	59298	16863975	308
1400	189418	7391061	61199	22876191	310
UDP					
50	81675	612182	14321	3491376	**570**
100	86079	1161724	16000	6250000	538
200	89059	2245702	19803	10099480	450
1000	99911	10008908	49564	20175934	202
1400	104163	13440473	65228	21463175	160

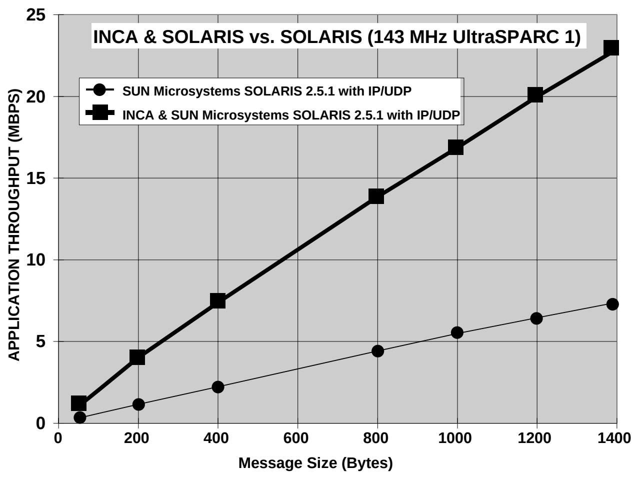

Figure 2. INCA Increases SUN WS UDP Throughput

3.3.2. INCA versus PC/LINUX 2.0. The overall network data application throughput performance improvement of the Gateway 2000 PC with LINUX running INCA versus the same system without INCA is listed in Table 3 and depicted in Figure 3. The percent (%) gain with INCA is expressed as the percentage of the current system throughput that INCA achieves.

INCA outperforms the present PC system anywhere from 255% to 587% depending upon the message size. The largest gain is in protocol processing, where INCA performs protocol processing 12,600% faster than LINUX without INCA.

It is apparent from the results in Table and Figure 3 that INCA performs protocol processing in a much more efficient manner than the current PC/LINUX system. An improvement by INCA of 12,600% for small message sizes and 1500% for large message sizes is quite substantial. Furthermore, the implementation of IP and UDP was retrofitted into INCA. A “from scratch” implementation of IP and UDP would optimize the protocol processing loop more, providing even greater throughput improvement with INCA.

Checksumming processing has been sped up in the INCA ILP loop through the use of a checksum routine optimized for a RISC CPU and through integration. In addition, the protocol data and control processing is separated.

Since the Pentium Pro is not a RISC processor, the protocol performance improvement comes mostly from the elimination of data copies (including many reads and writes within a protocol processing process), the ILP loop and the cache performance improvements from an integrated protocol processing loop.

As the number of protocols or protocol functions increase, INCA’s performance over the base system increases even more. With additional protocol processing,

Table 3. INCA versus LINUX IP/UDP Performance Results

INCA VERSUS LINUX 2.0, 167 MHz Pentium Pro PC (IP and UDP Protocols)									
Message	CNI Main Memory Transfer			Protocol Processing			Application Throughput		
Size	(µs)		INCA	(µs)		INCA	Mbps		INCA
(Bytes)	LINUX	INCA	% Gain	LINUX	INCA	% Gain	LINUX	INCA	% Gain
10	1829	1344	73	1889	15	**12593**	0.02	0.06	274
100	1893	1506	80	1991	18	11061	0.21	0.52	**255**
200	2013	1517	75	2055	42	4893	0.39	1.03	261
500	2193	1551	71	2008	45	4462	0.95	2.51	263
1000	2569	1595	**62**	2029	79	2568	1.74	4.78	275
1500	3043	2069	68	3927	110	3570	1.72	5.51	320
2000	3043	2069	68	4357	160	2723	2.16	7.18	332
4000	3280	2306	70	7463	305	2447	2.98	12.26	411
10000	4228	3254	77	18347	663	2767	3.54	20.42	576
65000	13234	12260	**93**	95056	6186	**1537**	4.80	28.19	**587**

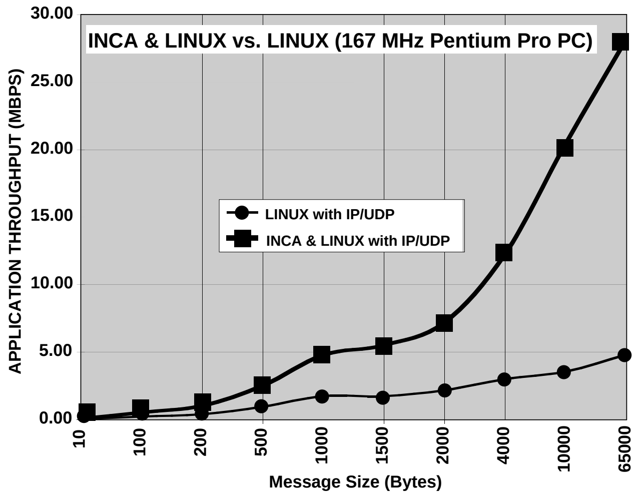

Figure 3a. INCA versus LINUX IP/UDP Application Throughput

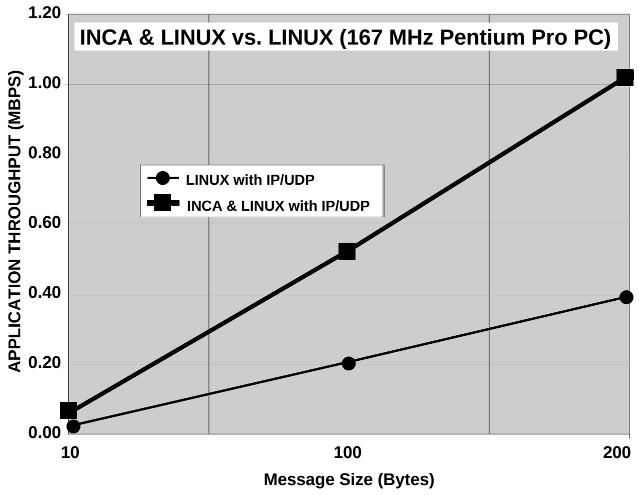

Figure 3b. INCA versus LINUX IP/UDP 10-200 Byte Application Throughput

such as XDR presentation formatting, INCA's and LINUX's throughput decreases. However, INCA's throughput decrease is half that of LINUX's. The improvement of INCA over LINUX increased from a maximum of 587% to a maximum of 613%.

The larger the message size, the more INCA improves throughput over the present LINUX system. The larger the messages, the more data to copy over the memory bus, the greater the performance improvement with INCA's reduced data copying. The more protocols that are used, for example the addition of XDR presentation protocol processing, the greater the performance improvement of INCA from the use of ILP.

Unlike many other performance techniques, INCA improves performance for small (≤ 200 byte) messages as well as for large messages.

4. Architectural considerations

The INCA architecture can be implemented within individual application programs, within the OS, or both. The software library comprising the architecture can be recompiled into applications/ linked to applications, or recompiled into the OS.

The amount of memory required for an application will increase by approximately the amount of the INCA library plus some additional RAM memory for intermediate data processing results. In light of the rate of increase in RAM densities (and hence capacities) and costs of a few dollars per Megabyte, the few additional Megabytes of memory required appear not to be a limiting issue. Even in a case of many large applications executing on one machine in a multi-tasking environment, virtual memory management and possible additional RAM can meet the need for more memory.

Data security may be somewhat reduced without an additional protection mechanism. As with any architecture that performs network data processing in user memory space, the effectiveness of existing OS protection mechanisms is reduced. Unauthorized access to user network data can become more likely.

The use of ILP affects protocol modularity in the sense of ease of swapping protocols in and out of the architecture. Unless designed for ILP, the incorporation of protocol modifications or new protocols will require manual coding, regardless of the implementation method. The process of integrating protocol changes could be automated at some point in the future.

INCA is bi-directional. One can transmit as well as receive using the INCA software library, thereby potentially doubling the performance benefit. For computers relying on a network connection in order to provide user services, such as a network computer, the INCA architecture would provide an even greater performance benefit. For such computers every application requires network data communication and processing. The INCA architecture becomes the processing path for an even greater percentage of user time, increasing the performance benefits of INCA.

5. Conclusions

Internal computer data processing overheads are limiting the throughput of network communicated data to a few megabits per second, despite the availability of near Gigabit networks and processors operating at near network speeds. Increased network communications throughput can be achieved through many design features, but the main options are optimizing the processing functions in the protocol architecture and reducing the number of data copies. Today's computer systems utilize the OS to perform both of these functions. Modifying the OS would result in major non interoperability costs and will not be supported by the commercial industry. Replacing the OS is also not an acceptable option. Good performance alone is not enough. Without the ability to interoperate with the existing application programs and machine/OS combinations by running application programs without modification, INCA research would suffer the consequences of almost all OS innovation research, a lack of applicability and a relegation to obscurity. Interoperability, including portability of the implementation, is required for acceptance and utilization of network communications performance improvements.

INCA improves network data throughput significantly without sacrificing interoperability. The combination of INCA architecture features and implementation methods: reduction in data copying between and within internal network data handling and protocol processing entities, message buffer management, checksumming and other protocol processing optimizations for RISC processors and level 1 "cache friendliness", make for substantial performance improvement.

Implementation on COTS WS and PC systems shows that INCA can function with all existing applications, OSs, communication protocols, machines, NIUs and networks while providing anywhere from 260% to near 760% greater data throughput than existing systems without INCA. The performance improvement is independent of the network link speed. Whether the network connection is via Gigabit or dial up modem speeds, or whether the machine requires the network in order to function (network computer), INCA provides the same percentage of improvement in data processing time.

INCA improves small message throughput as well as large message throughput by a significant percentage. The communication bandwidth of a system is often measured by sending a virtually infinite data stream from one node to another. While this may be representative of a few applications, the demand for high bandwidths when sending many small messages (e.g., a few hundred bytes) is increasing.

With such an improvement in network communications data throughput at the application program level, existing systems can utilize high speed networks and the exponentially increasing CPU power to provide a whole new realm of applications that require fast, high bandwidth network communications, either with many large messages or with many small messages.

As a stand-alone software only library, INCA can be ported to almost any existing and next generation machine, whether a home user's computer linked to a network via modem or whether a powerful server on a Gigabit link network. INCA performance scales with network, CPU and memory speed, memory bus bandwidth and NIU improvements, making the INCA results relevant for some time to come.

6. References

[1] P. Druschel, M. B. Abbott, M. A. Pagels, and L. L. Peterson, "Network Subsystem Design: A Case For An Integrated Data Path", IEEE Network (Special Issue on End-System Support for High Speed Networks), Vol. 7, No. 4, pp. 8-17, Jul. 1993.

[2] D. Mosberger, L. L. Peterson, and S. O'Malley, "Protocol Latency: MIPS And Reality", Technical Report TR 95-02, Department of Computer Science, The University of Arizona, Tucson, AZ, 1995.

[3] V. Roca, T. Braun and C. Diot, "Efficient Communication Architectures For Open Systems", Institut National De Recherche En Informatique Et En Automatique (INRIA), Sophia Antipolis, France, Jul. 1996.

[4] N. C. Hutchinson and L. L. Peterson, “The X-Kernel: An Architecture For Implementing Network Protocols”, IEEE Trans. Soft. Eng., Vol. 17, Jan. 1991.

[5] M. Accetta, R. Baron, D. Golub, R. Rashid, A. Tevanian, and M. Young, “Mach: A New Kernel Foundation For UNIX Development”, in Proceedings of the Summer 1986 USENIX Technical Conference and Exhibition, Jun. 1986.

[6] A. Mainwaring and D. Culler, “Active Message Applications Programming Interface And Communication Subsystem Organization”, Draft Technical Report, Computer Science Division, University of California at Berkeley, 1996.

[7] T. von Eicken, A. Basu, V. Buch, and W. Vogels, “U-Net: A User-Level Network Interface For Parallel And Distributed Computing”, Proc. of the 15th ACM Symposium on Operating Systems Principles, Copper Mountain, CO, Dec. 3-6, 1995.

[8] D. D. Clark and D. L. Tennenhouse, “Architectural Considerations For A New Generation Of Protocols,” in ACM Communication Architectures, Protocols, and Applications (SIGCOMM ’90), Philadelphia, PA, USA, Sep. 1990.

[9] M. B. Abbott and L. L. Peterson, “Increasing Network Throughput By Integrating Protocol Layers”, EEE/ACM Transactions on Networking, Vol. 1, No. 5, pp. 600-610, Oct. 1993.

[10] T. Braun and C. Diot, “Performance Evaluation And Cache Analysis Of An ILP Protocol Implementation”, IEEE/ACM Transaction on Networking, Jun. 1996.

[11] J. Kay and J. Pasquale, “The Importance Of Non-Data Touching Processing Overheads In TCP/IP”, Proceedings of the ACM Communications Architectures and Protocols Conference (SIGCOMM), pp. 259-269, San Francisco, CA, Sep. 1993.