An Efficient Implementation Architecture for Layered Communication Systems

Célio Vinicius Neves de Albuquerque Marcelo Dias Nunes
Otto Carlos Muniz Bandeira Duarte

COPPE/EE - Universidade Federal do Rio de Janeiro - Programa de Engenharia Elétrica
P.O. Box 68504 - CEP 21945-970 - FAX: +55 21 290.6626 - Rio de Janeiro - RJ - Brasil
Email: {celio, bill, otto}@coe.ufrj.br

Abstract

This paper proposes and analyses an efficient implementation architecture well suited to layered communication systems. High throughput is achieved by the design of specific layer interfaces and data structures, specialized mechanisms of memory management, timer management and task scheduling. Performance measurements were accomplished considering a complete OSI protocol stack. The obtained results show a throughput efficiency that attains 6 Mbit/s for MAC Layer user and 2.5 Mbit/s for the Presentation Layer user in a remote communication. A 43 Mbit/s throughput for the MAC Layer user is achieved in a loopback configuration. The most important bottlenecks are analysed and consist in the Transport checksum, acknowledgment step frequency, data copy, memory management and task scheduling.

1. Introduction

Transmission media and VLSI technology have evolved allowing communication systems capable of supporting data rates that are orders of magnitude greater than those used in current systems. Throughput in local area, metropolitan area and high speed networks is often limited by the processing part of the whole system. Several works [1,2,3] have stated that high throughput depends on a careful protocol implementation. Some authors exploit the intrinsic protocol parallelism [4,5,6], others propose completely new protocols [7].

In order to achieve high performance, this paper presents an implementation architecture suitable for layered communication systems based on specific layer interfaces and data structures, specialized modules of memory management, timer management and task scheduling. The proposed implementation architecture privileges flexibility, modularity and weak coupling between adjacent layers, avoids data copy in interlayer communications, minimizes the necessary retransmission buffers and takes into account memory fragmentation.

In view of evaluating the proposed implementation architecture, a complete Open Systems Interconnection (OSI) stack [8,9] was implemented according to this architecture and performance measurements were accomplished. Several functions required by a communication system were analysed, such as the throughput due to each layer processing time, checksum mechanism, acknowledgment step frequency and segmenting/reassembling. In the same way, the performance of the specific modules, that support the proposed architecture and replace the basic operating system functions, was computed.

The measurements show a good overall system performance and emphasize as the most important throughput bottlenecks Transport checksum computing, acknowledgment step frequency, memory management, data copy and task scheduling.

This work is organized as follows. Section 2 presents the implementation architecture. Its performance measurements are given in Section 3. Finally, conclusions are presented in Section 4

2. The Implementation Architecture

High performance, flexibility and modularity are the main objectives of this implementation architecture. It employs asynchronous communication and defines a standard interface between layers (Figure 1). This interface consists of a pair of FIFO queues per connection and a set of data structure access procedures. It aims at normalizing the access to the services offered by adjacent layers, allowing flexibility, modularity and weak coupling between layers.

Each FIFO element (Figure 2) is represented by a fixed size structure called *Primitive Structure* containing the following three fields:

- Primitive Structure Pointer - used for linking with the next FIFO element;
- Primitive Name - name of the incoming/outgoing service primitive;
- Parameter Structure Pointer - pointer to a structure that contains the current primitive parameters, called *Parameter Structure.*

The chaining method of inserting PCI avoids data copy and waste of memory, consisting in:

- allocating and filling up a PCI-size memory region;
- associating this region to a *chaining structure*;
- chaining this structure to the received SDU structure.

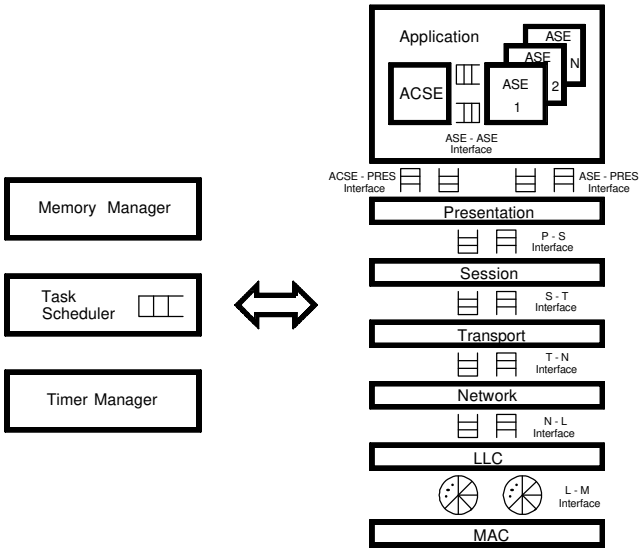


Figure 1: Implementation Architecture for the OSI stack.

This approach seemed to be the best way of inserting PCIs. Figure 2 shows the implemented data structures that make possible an efficient memory management. To implement this feature, the *Parameter Structure* was defined [11,15] containing the following five fields:

- Parameter Structure Pointer - used for linking with other Parameter Structure;
- User Data Pointer - pointer to (part of) the primitive user data parameter;
- User Data Length - length in bytes of the respective user data parameter;
- Parameter Pointer - pointer to the other primitive parameters;
- Number of Allocations.

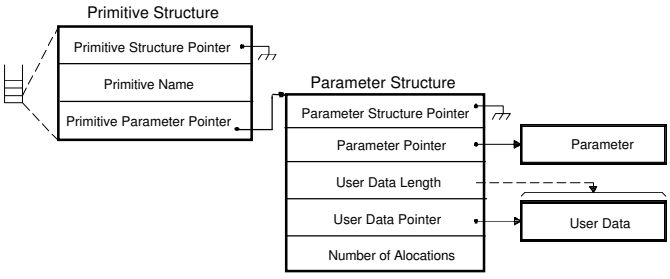


Figure 2: Interface Data Structures.

The *Number of Allocations* field plays an important role for the system. It avoids several retransmission copies when protocols based on error recovery by retransmission were used. This field is incremented/decremented every time a layer allocates/deallocates the structure. The correspondent retransmission buffer is released from memory only when this field reaches zero.

Memory Management

In order to avoid excessive fragmentation, the memory was divided in transmission and reception regions, each one subdivided in three sub-regions for *Primitive Structures*, *Parameter Structures* and data. An ordered memory utilization makes its management easier and faster.

Task Scheduling

A great improvement in performance was achieved with the creation of an elaborate task scheduler module [15]. The main modification was the inclusion of a FIFO queue of tasks ready to be executed. The sending of a primitive implies sending the identifier of the receiver task to the task scheduler queue. When a task is executed, data from a specific instance are accessed through a index.

Timer Management

The Timer Manager implementation follows Varghese and Lauck proposal [16], where an analogy with a real clock is simulated by a cyclic counter. This implementation proved to be very efficient, avoiding unnecessary logical checks at each hardware interrupt, thus allowing the creation of a great number of run-time timers.

3. Performance Measurements

The measurement analysis takes into account the following protocol stack: Presentation, Session, Transport (Classes 0, 2 and 4), Inactive Network Protocol (INP), LLC Type 1 and MAC. The implemented MAC Sublayer is the IEEE 802.3 (CSMA/CD) for 10 Mbit/s Ethernet media. In order to become network adapter independent, this sublayer directly accesses a set of routines, known as Packet Drivers, which support a large number of communication controller cards.

The implemented system is coded in C language and runs on IBM-PC computer environments using DOS operating system. The basic measurement scenario consists of two identical 16 MHz 386 SX processors connected to an Ethernet LAN through NE2000 network adapters.

This performance analysis emphasizes on the throughput achieved by each layer user. Hereafter, throughput is defined as the total useful information transferred (messages without headers and tails) per time unit. Figure 3 shows the throughput for the Packet Driver, MAC, LLC, INP, TP0, Session and Presentation Layer users.

As depicted in Figure 3, with the increase of frame length, throughput reaches 6 Mbit/s. This result shows protocol processing time negligible when compared to transmission time (T_{send}), for large frames, i. e., Packet

Driver access routines and LAN adapter constrain system throughput. A special loopback feature was developed at the Packet Driver level, consisting in frame copies from the transmission buffer to the reception buffer. It is important to emphasize that, within this configuration, T_{send} is replaced by the copy time T_{copy} and transmission/reception processing occurs on the same microcomputer. Just as an example, this time is about 0.22 ms for 1024 byte frames. MAC sublayer throughput in loopback configuration achieves 43 Mbit/s proving the network adapter limitation.

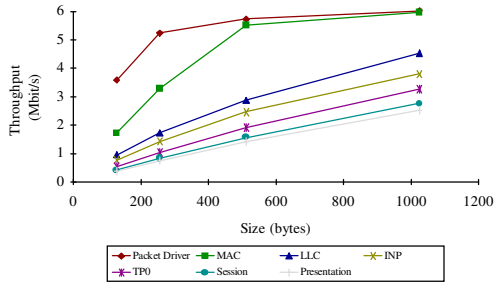
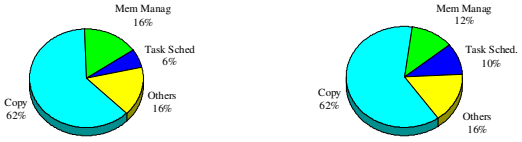


Figure 3: Throughput versus message size.

All communication controllers require sending data to be in a contiguous memory area. Thus, LLC sublayer performs an unavoidable copy of chained data to a memory region (transmission buffer in LLC-MAC interface). This data copy is an important performance degradation factor, responsible for a frame size dependent decrease in throughput. In order to visualize the actual limitation introduced by copy, Figure 4 shows two pie charts for the specific LLC sublayer processing. It is also observed that memory allocation and Task Scheduler queue insert procedures spend a considerable processing-time portion.



(a) originator (b) recipient

Figure 4: LLC sublayer processing a 1024 frame length.

Figure 5 shows a comparative throughput analysis between Transport Protocols (Classes 0, 2 and 4). These results give the performance degradation due to window flow control (acknowledgment processing) and error recovery (checksum computing). TP4 user throughput plots (with and without checksum) show a performance drop ranging from 19% to 56% for 128 to 1024 byte messages, respectively. TP2 Protocol and TP4 Protocol without checksum present almost the same result, proving that the proposed *Number of Allocations* field used in the retransmission mechanism [11] is efficient. TP2 Protocol plots also show system processing overcharge caused by Acknowledgment Protocol Data Units (AK-TPDUs) transmission. A performance improvement of 28% is obtained when an acknowledgment step of four is employed for 1024 byte messages. Acknowledgments are tied to window flow control, and a window size of eight was used. TP0 and TP2 Protocol performance results differ mainly due to flow control.

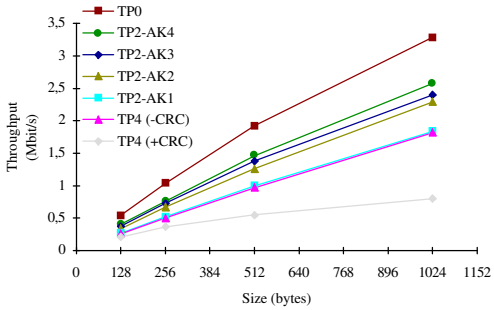
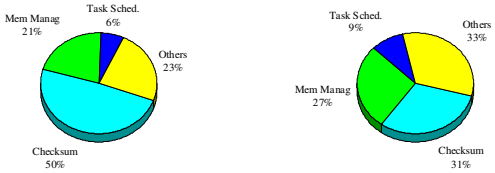


Figure 5: Transport user throughput versus message size.

Figures 6 shows the performance degradation introduced by the checksum computation in the Tranport class 4.



(a) originator (b) recipient

Figure 6: TP4 processing time for 128 byte messages length.

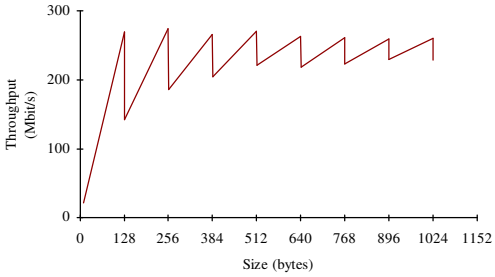


Figure 7: Transport user's throughput versus message size segmenting at 128.

Although the proposed implementation architecture allows segmentation/reassembly without copy (only pointer manipulation is performed), this function strongly penalizes the throughput due to the additional TPDUs to be processed. As depicted in Figure 7, as TSDU size increased, throughput increased steadily until the next segmentation. The drop in throughput is directly attributable to the time required to process one more TPDU.

Another analysis was performed consisting in identifying the normal data path through the TP2 code and counting the instructions. It was observed that TP2 protocol processing is in fact not the main overhead source. From the 108 sending instructions, it takes 32 for memory

management (*Primitive/Parameter Structures* and buffer allocation), 27 for the effective protocol processing (Primitive / PDU decoding and processing, etc.), 26 for task scheduling, and 23 for interface processing (queue tests, insertion and removal). Some additional instructions are required for the receiver, mainly to process the acknowledgments (54 instructions). From the 154 receiving instructions, 56 are either for buffer allocation or deallocation, 36 are spent in protocol processing, 34 in task scheduling and 28 in interface processing.

Finally, Figure 8 illustrates the relative weight of checksum computing, data copy and memory management for the whole system.

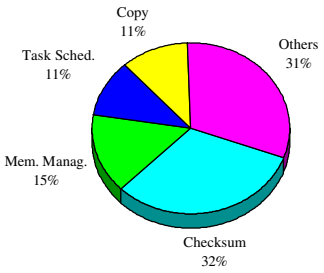


Figure 8: Whole system main overhead sources.

4. Conclusions

In this work an efficient layered communication system was presented. Its implementation architecture was described, which is based on:

- asynchronous message interlayer communications, resulting in weak coupling between adjacent layers and high modularity;
- interface data structures that avoid copies for interlayer communication, segmentation and retransmission;
- a high performance memory manager;
- a task scheduler that avoids unnecessary queue tests;
- an efficient timer manager implementation.

The measurement results show that the system performs well attaining the maximum throughput at the MAC Layer user (6 Mbit/s). In a loopback configuration this layer throughput has reached 43 Mbit/s. The unavoidable copy realized by the LLC Layer has shown that memory bandwidth is an important issue to obtain higher performance. It takes 62% of the total LLC processing time and 11% of the total system. The Transport checksum is too much time consuming and justifies a specialized hardware assist. Blocking the acknowledgments influences the performance results. The throughput was multiplied by a factor greater than two when acknowledgment step frequency was divided by a factor of four.

The throughput results are quite good and the proposed implementation architecture fits well for most of the layered communication systems.

Acknowledgments

Marcelo Lanza, João Pereira, Luis Baginski, José Rezende, Rainer Schatzmayr, Rogério da Cunha, Fernando de Barros, Frederico Liporace, Calixto Neto and Marcelo Achá have all contributed to the implementation of parts of the system.

References

[1] D. D. Clark et al, “An Analysis of TCP Processing Overhead”, *IEEE Com. Mag.*, vol. 27, no. 6, Jun 89.

[2] W. A. Doeringer et al, “A Survey of Light-Weight Transport Protocols for High-Speed Networks”, *IEEE Trans. Com.*, vol. 38, no. 11, Nov 90.

[3] L. Svobodova, “Implementing OSI Systems”, *IEEE JSAC*, vol. 7, no. 7, pp. 1115-1130, Sep 89.

[4] O. G. Koufopavlou, A. N. Tantawy and M. Zitterbart, “Analysis of TCP/IP for High Performance Parallel Implementations”, *Proc. 17th Conference on Local Computer Networks*, Minesota, Sep 92.

[5] M. Zitterbart, “Parallel Protocol Implementation for High Speed Networks”, in *Proc. SBT/IEEE ITS*, Rio de Janeiro, Brazil, Sep 90.

[6] C. Papadopoulos, G. M. Parulkar, “Experimental Evaluation of SunOS IPC and TCP/IP Protocol Implementation”, *IEEE ACM Trans. Networking*, vol.1, no. 2, pp. 199-216, Apr 93.

[7] A. N. Netravali, W.D. Room and K. Sabnani, “Design and Implementation of a High Speed Transport Protocol”, *IEEE Trans. Com.*, vol. 38, no. 11, pp. 2010-2024, Nov 90.

[8] The International Organization for Standardization, Data Processing, “OSI Systems Interconnection Basic Reference Model”, ISO Doc. DIS7498, 85.

[9] J. D. Day and H. Zimmerman, “The OSI Reference Model”, *Proc. of the IEEE*, vol. 71, no. 12, pp. 1334-1340, Dec 83.

[10] L. F. Baginski, F. M. C. de Barros, R. Schatzmayr and O. C. M. B. Duarte, “Implementação e Análise de Desempenho em um Sistema de Transferência de Dados”, in *X Congresso da Sociedade Brasileira de Computação*, Brazil, Jul 91.

[11] L. F. Baginski and O. C. M. B. Duarte, “Um Modelo de Implementação de Alto Desempenho para Sistemas Abertos”, in *IX Congresso da Sociedade Brasileira de Telecomunicações*, São Paulo, Brasil, Sep 91.

[12] C. N. Woodside and J. R. Montealegre, “The Effect of Buffering Strategies on Protocol Execution Performance”, *IEEE Trans. Com.*, vol. COM-37, pp. 545-554, Jun 89.

[13] L. Svobodova, “Measured Performance on Transport Service in LANs”, *Comput. Networks ISDN Syst.*, vol. 18, no. 1, 89.

[14] W. T. Strayer, A. C. Weaver, “Performace Measurements of Data Transfer Services in MAP”, *IEEE Network*, vol. 2, no. 3, May 88.

[15] L. F. Baginski, “Ambiente de Implementação para Sistemas de Alto Desempenho”, *Master Thesis*, PEE-COPPE/UFRJ, Jan 92.

[16] G. Varghese and T. Lauck, “Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility”, in *Proc. 11th ACM SIGOPS*, Texas, 87.