DESIGN OF A MODULAR AND EFFICIENT COMMUNICATION SUBSYSTEM

Torsten Braun[1], Jochen Schiller[1], Claudia Schmidt[1], Martina Zitterbart[2]

[1] University of Karlsruhe, Institute of Telematics,
Zirkel 2, 76128 Karlsruhe, Germany
Fax: +49 721 388097
[t.braun, j.schiller]@ieee.org,
cschmidt@ira.uka.de

[2] University of Braunschweig, Institute for Operating Systems and Networking, Bültenweg 74/75,
38106 Braunschweig, Germany
Fax: +49 531 391 5936
zit@ibr.cs.tu-bs.de

Flexible and efficient communication support is increasingly required by forthcoming applications, such as multimedia applications, interactive TV, or teleshopping. The communication subsystem presented in this paper is highly flexible due to its modular de-layered design. It consists of protocol building blocks that can be configured to service-tailored protocols according to the individual application service requirements. Thus, support for a variety of known and upcoming services can be supported. A QoS manager is designed in order to support and manage the services. Moreover, the use of dedicated VLSI components for performance critical tasks enables efficient implementations.

1 Introduction

During the last years, computer communication is experiencing drastic changes from simple text-based exchange to advanced communication facilities including audio and video streams. In this context, information superhighways can be seen as an increasingly popular buzzword pointing to new applications and technologies. Communication technology has drastically evolved towards fiber optic networks and enhanced transmission technologies. Very high-speed networks are becoming reality. Moreover, emerging networks (e.g., B-ISDN) are characterized by their service-integrated nature. This makes them suitable to support forthcoming applications that increasingly comprise video and audio components. Additionally, group communication is considered as basic requirement for upcoming application scenarios. Although most of today's communication subsystems include protocols and implementation techniques developed for basic communication requirements, they are used to serve any kind of applications. However, they seem not to be really adequate for emerging application requirements and, thus, for tomorrow's communication needs. Therefore, this paper presents the design of a modular and flexible communication system that is oriented towards evolving requirements. New service requirements are addressed by a novel service model and by a highly modular internal structure of the communication subsystem. It is especially targeted towards highly flexible and efficient implementations. Therefore, a suited implementation platform is needed. Efficient implementations of performance-critical protocol tasks are required and service guarantees need to be supported. In most cases the integration of communication components into general purpose workstations is not solved satisfyingly. Most of today's networking components consist of a simple network adapter inserted into a system bus and several software modules performing most of the tasks. Mostly, this leads to inefficient and highly inflexible implementations. Within the design of a modular and efficient communication system, we apply hardware/software codesign techniques in order to achieve an efficient implementation that still provides a high degree of flexibility. Performance critical components are mapped onto dedicated hardware and less critical components are formed by traditional software modules. Examples can be found in certain protocol support functions, such as management of dynamic data structures.

The paper is organized as follows. Section 2 presents the basic model of the communication system. Sections 3 to 5 focus on specific parts. The service model including specific quality of service support is discussed in section 3. Section 4 presents the PATROCLOS protocol that provides a high degree of flexibility due to its modular design. The integration of dedicated hardware components is subject of section 5. Section 6 summarizes the paper and gives an outlook on future work.

2 Basic Communication Model

Generally, the OSI basic reference model has been designed as the internationally standardized model for computer communications. Nevertheless, the model that is applied in most practical cases is the Internet model comprising protocols, such as UDP, TCP and IP. Neither of the models provides an easy and straight forward support of emerging applications concurrently requiring multiple diverse communication services. Thus, they are not well-suited for a migration to modern distributed application platforms. In order to provide a high degree of flexibility, we prefer a de-layered model comprising several components implementing well-distinguished tasks, such as protocols, operating system support, and network access. Internally, they consist of flexible subcomponents which can be composed to service-tailored communication subsystems including lean and service-tailored protocols.

Figure 1 presents the basic design of such a model. It consists of a network oriented component dealing with network access and with physical layer tasks. On top of the network oriented component a communication oriented component is located. It comprises all functionalities concerned with communication tasks. Internally, the communication oriented component is subdivided into a protocol subsystem and an operating subsystem. The protocol subsystem consists of protocol functions, such as acknowledgment, retransmission, or connection establishment.

Protocol functions need to be selected and combined in order to provide the service requested by the applications. More generally, the overall protocol subsystem can be viewed as a set of protocol state machines interoperating together. This differs from the recent view on communication systems which basically consists of a set of protocols (each provided by a single protocol state machine) that need to be hierarchically layered. Such an approach can only provide limited flexibility. In [13], a flexible communication subsystem providing such functionalities is presented.

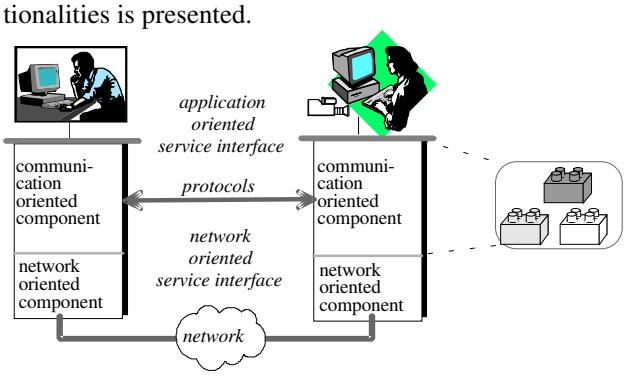

Figure 1: Modular communication subsystem

Today's communication scenario is characterized by an increasing variety of applications that may need different communication services. Especially the increasing incorporation of audio and video components as well as group communications drastically changes traditional communication scenarios. Traditionally, communication services are either characterized by connectionless or connection oriented services, associated with unreliable and reliable communication. Thus, the main focus has been in distinguishing unreliable and reliable communication services. However, more types of services in-between these two points may be needed, in order to properly serve emerging applications. Moreover, certain performance oriented parameters seem to get more and more important. For example, audio and video applications mostly require continuous delivery of data at the service interface. Additionally, communication delay can be a very important parameter for various applications. Therefore, new service models are required in order to properly express service requirements. However, at the application interface itself they cannot be directly applied. The application interface should be using a very simple design. Consider the following example. A secretary that is asked to set up a certain kind of communication including a video stream can neither request the service in terms of network requirements (i.e., peak cell rate, etc.) nor in terms of any details of the video coding algorithms (such as compression type). He or she, mostly, may just be capable of requesting very good, average, or low quality or refusing the provided offer. Thus, a very high level service interface to the applications is required. In Figure 1 that interface is called the *application oriented service interface*. Internally, each of the components offers an individual service, that is explained in detail in section 3.

3 QoS Support

In order to support highly diverse services, an abstraction, a so-called **QoS building block** is introduced. QoS building blocks are the basic elements in a QoS architecture that provide service guarantees by handling shared resources. The internal handling of resources is hidden by QoS oriented service interfaces. Other building blocks can access the internal resources through the service interface. At the **service interfaces** the service is described by a number of service parameters, such as bandwidth, delay, jitter, and reliability. Each parameter can be described by different values, e.g., *useful, average,* and *threshold* [13]. Useful and threshold values define an interval that should be enforced. Especially, *threshold* defines a hard bound that is not allowed to be exceeded. Moreover, QoS-parameters are of different importance to applications. This can be expressed by several *types of service* (ToS), such as the three types suggested by [3]: a deterministic, a statistical, and a best effort service. The ToS can be selected separately for each QoS parameter [2].

Internal **resource managers** are responsible for reservation and assignment of resources according to the individual service requirements. Typical tasks include admission control test, resource reservation, resource assignment, and resource release. Resources that are needed for data transmission must be reserved prior to data transfer in order to guarantee any QoS. Therefore, the resource manager checks during the admission control test, if the remaining resource capacity is sufficient for the new request without violating other guarantees. In case of a positive response, reservations are made. During the data transfer phase resources must be assigned accordingly. The protocol subsystem as well as the operating system and the network oriented component are each responsible for different resources. Thus, the resource manager of the operating system handles resources, such as processor capacity, memory, and timers. The communication subsystem manages a variety of protocol functions and protocol support functions (e.g., protocol memory management), and the network is responsible for bandwidth.

In contrast to the work presented in [1], we are targeting towards a de-layered approach for an integrated QoS management (cf. Figure 2). As already mentioned each subcomponent provides an individual service interface. The most common service interface is the *application interface*. It is crucial to design an application interface hiding as many communication related parameters and functions as possible. In general, applications can be classified and described by the media type (e.g., video, audio, text, graphics), and their interaction scheme (e.g., interactive, asynchronous). Applications, that are based on the same media type and interaction scheme build one *application class*. Each class is described in detail by a set of service specific parameters, e.g., compression, frame rate, frame size for interactive video transmission. The internal service interfaces are transparent to the application. However, typically different QoS parameters are applied at each service inter-

face. Therefore, a mapping of parameters is required. This is one of the tasks of the QoS manager. QoS mapping performs an automated translation between two different QoS specifications. The QoS manager first maps application level QoS parameters to an *internal QoS structure (IQS).* The IQS includes an internal service description used by the QoS manager. This description reflects application requirements but is represented by system-related parameters. In a second step, this structure is mapped to the QoS specifications, that are used by other subcomponents, such as operating system, protocol subsystem and network. This way, less mapping functions are needed and a higher flexibility for further enhancements is provided. For example, the QoS manager extracts the QoS parameters relevant to the protocol subsystem where an appropriate protocol is configured automatically.

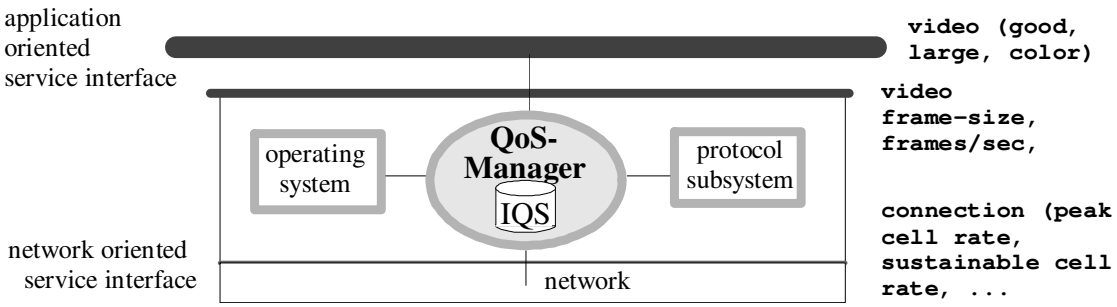

Figure 2: QoS management architecture

Forthcoming service integrated communication systems must be able to offer application-tailored services to a variety of applications. These services require appropriate protocols to efficiently support service requirements. Protocol functions that are not needed, should be switched off, since they reduce the achievable performance [4]. A lean and service-tailored protocol can be build by selecting the minimal possible set of protocol functions. Moreover, specific protocol mechanisms implementing a protocol function (e.g., window-based or rate based-flow control) can be selected and parametrized dependent on the requested service.

Service-tailored protocols are configured by a protocol manager located in the protocol subsystem. Thus, beside the information stored in the IQS, several rules are needed, that allow the protocol manager to select protocol functions. Examples of such rules are:

- If (determistic throughput guarantee | deterministic delay guarantee) then select connection with resource reservation
- If (deterministic reliability guarantee) then select retransmission-based scheme

Other rules depend on end-to-end values for delay, jitter, and throughput as well as on parameters such as round trip time. These parameters are stored in the IQS. They are used by the protocol manager together with several rules to build a lean service-tailored protocol. In fact, there is only a limited set of rules necessary. By the use of such a rule-based system, all communication specific functions are transparent to the application. An application-related service description is sufficient to automatically select an appropriate protocol.

4 PATROCLOS

The PATROCLOS protocol architecture consists of a set of building blocks, so-called extended finite state machines (FSMs). Each FSM performs a dedicated protocol function such as connection management or acknowledgment. The modular approach supports protocol configuration depending on the service requirements and the characteristics of the networks. FSMs belonging to the same PATROCLOS entity exchange internal messages for co-operation. All FSMs are designed as autonomous as possible in order to minimize interactions with other FSMs of the same entity. Communication among FSMs is uni-directional to allow a high degree of concurrency.

The protocol architecture distinguishes between interface and protocol FSMs. *Interface FSMs* are located at the interfaces to the application and the network. They support interactions of PATROCLOS with the service user or with the underlying network unit and are only involved in local communications within an entity. In contrast to protocol FSMs, they do not communicate with FSMs of the peer entity. PATROCLOS can also be easily adapted to several kinds of transport services. This has already been done with the service defined within the F-CSS [13] and OSI95 [14] projects.

Protocol FSMs communicate directly by separate, so-called FSM protocols with the corresponding FSMs at the peer entity. Protocol FSMs are designed to decouple the connection state information exchange from user data exchange and to allow parallelism between send and receive part. All FSM protocols use separate protocol data units (PDUs), individual error recovery mechanisms, and timers. PDUs contain only the necessary information for the dedicated protocol function.

The *connection management* FSM is responsible for establishment, management, and termination of a connection. Therefore, it has to activate and initialize the other FSMs of the same entity for each new connection, and has also to deactivate these FSMs after connection termination. Several options and parameters are offered.

User data transfer can be divided into a send path and a receive path. The main tasks of both user data transfer FSMs are formatting and analyzing data PDUs. Additional functions are segmentation and reassembly. In intermediate systems, functions such as routing & relay-

ing, route management, and monitoring are required. Furthermore, the protocol offers several options for data transfer. These options may be specified by the service user or may be calculated by protocol configuration tools. User data may be delivered to the transport service user as PDUs, i.e., all received PDUs are delivered in the same order they arrived, or as SDUs. For delivery on a SDU basis a sequenced or an unsequenced delivery can be selected.

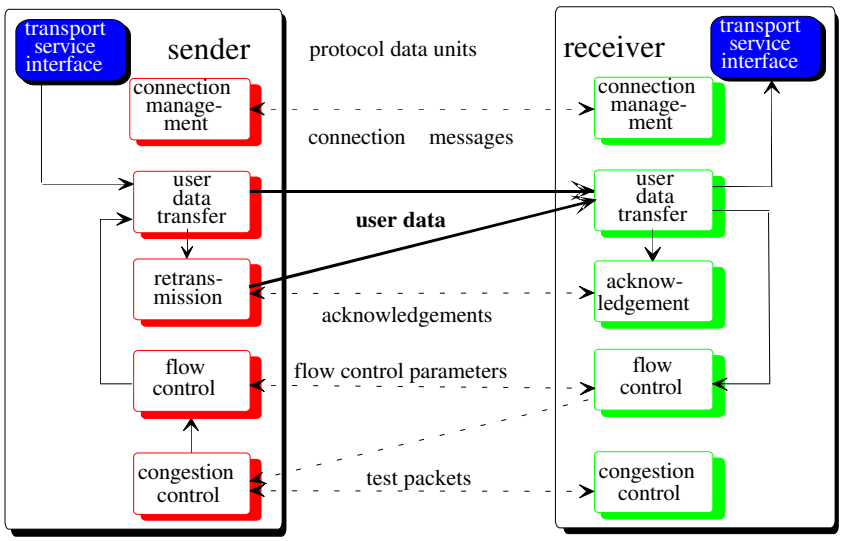

Figure 3: Protocol architecture of PATROCLOS

Because data transfer mostly performs the critical path in communication protocols, *retransmission and acknowledgment* are handled by two specialized FSMs. The FSM on the send side (retransmission FSM) requests and receives acknowledgments from the peer's FSM on the receive side (acknowledgment FSM). For unreliable data transfer, retransmission can be switched off. For reliable data transfer, selective or go-back-N strategy can be selected. While selective retransmission has benefits in networks with a large bandwidth-delay product, it requires significantly more buffer memory at the receiver. Because of the high degree of modularity, the FSMs can be configured highly independent of each other. The modular design concept supports the integration of additional functionality. E.g., multicast functions have been integrated in a later development step by enhancing the FSMs of the send part.

PATROCLOS has first been implemented on a transputer based multiprocessor environment. Compared to other transport protocol implementations on transputers, the PATROCLOS implementation achieves a very good performance. The separation of control functions from user data processing functions seems to be a successful approach, because it improves the critical path of user data processing significantly and allows a more efficient parallel processing. The PATROCLOS protocol is currently being implemented within an DEC Alpha OSF/1 environment. Furthermore, performance critical protocol functions are implemented as VLSI components as described in the following section.

5 Communication Co-Processor

Considering the use of dedicated hardware, so-called coprocessors appear to be very attractive. They avoid the bottlenecks of traditional adapter based solutions, like extensive data copy operations. Moreover, they can utilize most of the software approaches to speed up communication support, e.g., Integrated Layer Processing (ILP) and Application Layer Framing (ALF) [5].

Therefore, the proposed architecture was designed highly modular. Different functions can be mapped to the hardware depending on the individual requirements. Similar to a floating-point unit supporting an integer CPU by calculating floating point operations using dedicated hardware, a *Communication Co-Processor (CCP)* can support a standard CPU by performing many performance-critical protocol functions. Instead of calling subroutines in software the same command is sent to the coprocessor. The advantage of this approach can be seen in the required small changes to existing programs. Therefore, together with a *CCP* emulation identical software can run on systems with or without hardware communication support.

The *CCP* is the implementation framework for the subcomponents C_i and the central *Connecting Element (CE)* (cf. Figure 4). Examples for the C_i are protocol or support functions, such as management of timers and dynamic data structures, encryption/decryption, coding/decoding (e.g., ASN.1 [12]), and checksumming functions. PATROCLOS functions can be directly mapped onto C_is. All components inside a *CCP* are connected only via the *CE*. Data exchange among C_is takes place through the *CE*. To provide a maximum of flexibility the C_i are designed with identical interfaces. Furthermore, the *CE* manages an interface *I* to the standard processor outside the *CCP*. The *CE* can be implemented as a high-speed bus or simple crossbar switch to support concurrent communication inside the *CCP* similar to [6].

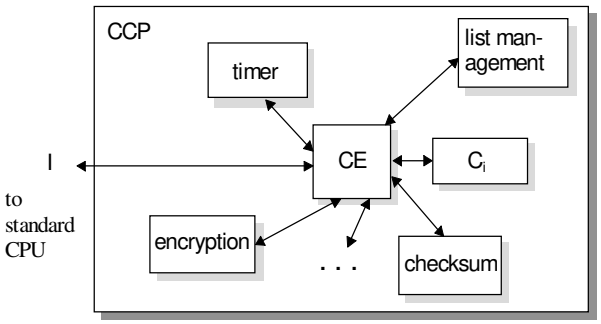

Figure 4: Modular implementation architecture of the Communication Co-Processor

5.1 Managing Dynamic Data Structures

Up to now, the well-known and often used data structure of a list has been used rarely in hardware due to the complexity of the manipulation functions. However, many functions for protocol processing rely on the power of lists to store dynamic data. One time-critical example is retransmission support. Therefore, a dedicated component has been developed to support retransmission.

If an application needs reliable data transfer, receiving of all data has to be controlled and enforced. Therefore, a dynamic data structure representing data that has to be acknowledged needs to be implemented. Every time data is sent or an acknowledgment is received, the retransmission list needs to be updated. Given a 1Gbps link and 1000 Bytes per packet, approximately every 10µs an update has to be performed. Therefore, this task is almost impossible for pure software solutions. The proposed hardware component acts like a specialized ALU with a certain set of powerful manipulation functions (c.f. Table 1 for examples). It also supports group communication which introduces a new dimension of complexity into protocol processing.

Figure 5 shows three lists representing states of three receivers (R_1, R_2, R_3) and, additionally, a so-called group list (G). Each receiver can have a different state concerning the missing data due to different qualities of the transmission paths. The white areas show known gaps in transmitted data, the black areas show positive cumulative acknowledged sequence numbers. The group list G stores gaps for the so-called *k-reliability* with k=2. Many applications do not necessarily require, that all receivers receive all data. In many cases it is sufficient, if a subset of receivers (e.g., 2 of 3 receivers) have successfully received the data packets. The shaded areas of list G show spans where more than 2 receivers missed the data. In order to guarantee a 2- reliability the appropriate data has to be retransmitted. An appropriate data structure for the list management is shown in Figure 6. It consists of lists for group IDs (*GL*), for the receiver IDs (*EL*), and the main list to store all necessary data. Only begin (s_a) and end (s_e) of a gap and a pointer to the next gap (n) are stored. The dashed lines indicate lists for group communication. The proposed component performs not only insert and delete operations for the list, but also joins two adjoining gaps and updates the group list for a given reliability.

The whole architecture with all its components has been described using the standardized hardware description language VHDL [10]. Therefore, all simulations and synthesis are based upon this language and were done using commercial tools [11]. The internal architecture of this unit specialized for retransmission support consists of memory to store the required information, registers, ALUs for standard operations, control units, and specialized comparators. The comparator units can, e.g., perform the operation $a \le b \le c,\ b \le c \le d$ in a single clock cycle ($a,b,c,d \in$ int). This operation is applied to search in a list. The developed retransmission ALU is fully microprogrammable. The operations of the ALUs, the control units, and the comparators are always executed in parallel in one clock cycle. Preliminary synthesis using a 1.0µm CMOS standard cell library show promising results. E.g., the above mentioned comparisons $a \le b \le c,\ b \le c \le d$ need 45ns on this coprocessor, while only the comparison $a \le b \le c$ requires 72.6ns on an DEC Alpha processor assuming one instruction per cycle and no context switching (11 instructions, 6.6ns cycle time, incl. load/store). The control logic of the processor needs 28800 gates, additional 64kByte RAM are needed to support the handling of more than 600 gaps in the transmitted data streams.

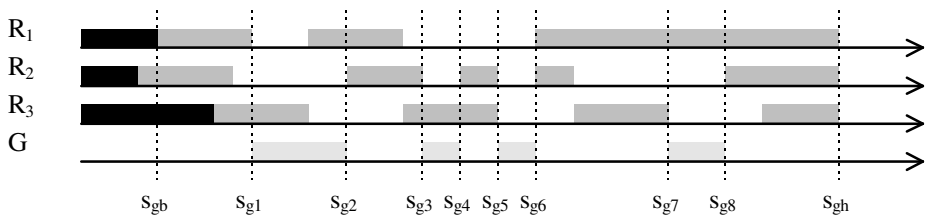

Figure 5: Example of the lists for group communication

6 Outlook

The presented paper describes the design of a modular communication subsystem that is targeted towards efficient service support of forthcoming applications. It follows new design principles that are centered around the use of communication building blocks each providing an individual service. They are composed to provide the overall service required by an applications. The modularity allows a high flexibility of the system. Moreover, it directly supports a hardware/software codesign in implementing such systems. A prototypical implementation in software has been implemented basically using protocol functions of the transport and net-

work layers. The PATROCLOS protocol has been efficiently integrated into this design. Currently software implementations of the quality of service parts and hardware implementations of performance critical tasks are under way. Additionally, an integration of already existing components in the overall architecture is performed. OSF/1-systems are used as target implementation platform.

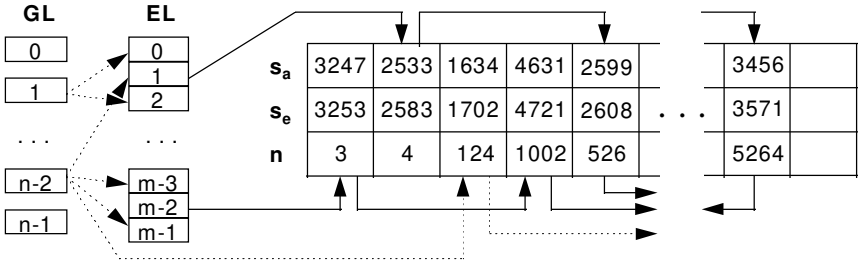

Figure 6: Data structure for the list management adapted to retransmission

operation	input parms.	out. parms.	comment
init_list	rec_id, seq_no		initializes a new list for the connection *rec_id* with the initial sequence number *seq_no*, sets the error flag if *rec_id* is already in use
add_mcg	mc_con_id rec_id		adds a new connection *rec_id* to an existing multicast group *mc_con_id*
set_gap	rec_id, seq_no, length		inserts new entry (*seq_no, seq_no + length*); overlapping entries are automatically joined or deleted, respectively
del_gap	rec_id, seq_no, length		deletes an existing entry, a part of an existing entry, or several existing entries, the deleted part is of the form (*seq_no, seq_no + length*); if necessary an entry is automatically divided into two new entries
get_gap	rec_id, ptr	seq_no, length, next	reads the entry *ptr* points to; if *ptr* = 0, the first gap is read out, if *next* = 0 the entry represented by (*seq_no, length*) is the last one, otherwise *next* point always to the next entry of the list

Dimensioning of the component: *rec_id, k* ∈ [0, 255]; *mc_con_id* ∈ [0, 63]; *seq_no, seq_no_1, seq_no_2, length, cont* ∈ [0, 2^{32}-1]; *ptr, next* ∈ [0, 2^{16}-1]

Table 1: Example operations of the retransmission ALU

References

[1] Campbell, A.; et al.: *Integrated Quality of Service for Multimedia Communications*, Proc. IEEE INFOCOM'93, San Francisco, USA, April 1993, pp. 732 - 739

[2] Campbell, A.; Coulson, G.; Hutchinson, D.: *A Multimedia Enhanced Transport Service in a Quality-of-Service Architecture*, MPG-93-22, Lancaster University, Department of Computing

[3] Ferrari, D. et al.: *Client-Network Interactions in Quality-of-Service Communication Environments*; in A. Danthine, O. Spaniol: High Performance Networking IV; IFIP, North-Holland, Amsterdam, 1993, pp. 221-234.

[4] Richards, A.; Seneviratne, A.; Fry, M.; Witana, V.: *Tailoring the Transport Protocol for Giga Bit Networks*; Australian Telecommunication Networks and Applications Conference, December 1994

[5] Clark, D.D.; Tennenhouse, D.L.: *Architectural Considerations for a New Generation of Protocols*, ACM SIGCOMM'90, September 1990, pp. 200-208

[6] Texas Instruments: *TMS320C80 Multimedia Video Processor (MVP)*, Technical Brief, Texas Instruments Inc., June 1994

[7] Braun, T.; Zitterbart, M.: *Parallel Transport System Design*, in: Danthine, A.; Spaniol, O.: High Performance Networking, IV, IFIP, North-Holland, 1993, pp. 397-412

[8] Varghese, G.; Lauck, T.: *Hashed and Hierarchical Timing Wheels: Data Strucutres for the Efficient Implementation of a Timer Facility*, ACM, 1987, pp. 25-38

[9] Steenkiste, P. A.: *A Systematic Approach to Host Interface Design for High-Speed Networks*, IEEE Computer, March 1994

[10] IEEE: *Standard VHDL Language Reference Manual;* IEEE Std 1076-1987

[11] Synopsys: *User Manuals*, Synopsys Inc., Mountain View, CA, 1994

[12] Bilgic, M.; Sarikaya, B.: *An ASN.1 Encoder/Decoder and its Performance*, PSTV X, Logrippo, et al., (eds.), Elsevier Science Publishers B.V. (North-Holland), 1990, pp. 141-154

[13] Zitterbart, M.; Stiller, B.; Tantawy, A.: *A Model for Flexible High-Perfromance Communication Subsystems*, IEEE Journal on Selected Areas in Communication, May 1993, pp. 507-518

[14] *The OSI 95 Transport Service with Multimedia Support*, Research Reports ESPRIT, A. Danthine (ed.), Project 5341, Vol. 1, Springer-Verlag