View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by INRIA a CCSD electronic archive server

Hight performance implementation of communication subsystems

Walid Dabbous

To cite this version:

Walid Dabbous. Hight performance implementation of communication subsystems. [Research Report] RR-2101, INRIA. 1993. inria-00074571

HAL Id: inria-00074571

https://hal.inria.fr/inria-00074571

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

High Performance Implementation of Communication Subsystems

Walid DABBOUS

N° 2101
Novembre 1993

PROGRAMME 1

Architectures parallèles,
bases de données,
réseaux et systèmes distribués

High Performance Implementation of Communication Subsystems

Walid Dabbous

Programme 1 — Architectures parallèles, bases de données, réseaux
et systèmes distribués
Projet RODEO

Rapport de recherche n ° 2101 — Novembre 1993 — 21 pages

Abstract: New applications with specific communication requirements are now being considered. These requirements put stringent constraints on both the performance and the service provided by communication protocols over high speed networks. The performance enhancement of these protocols is an essential step toward the building of high performance communication subsystems. In this paper, we present work on the high speed implementation of both presentation and transport functions. We show how Integrated Layer Processing based optimizations increase the performance of costly data manipulation functions. We also present a framework for the design of integrated communication subsystems based on Application Level Framing and Integrated Layer Processing.

Key-words: High speed protocols, Application level Framing, Integrated Layer Processing

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Implantation haute performance de modules de communication

Résumé : Avec l'apparition ds réseaux à haut débit, de nouvelles applications ayant des besoins de communication spécifiques sont en cours de développement. Les performances des protocoles de communication ainsi que les services fournis ne correspondent plus exactement aux besoins des applications. L'amélioration des performances des protocoles est donc une étape essentielle pour la construction de modules de communications performants. Dans cet article, nous présentons nos travaux sur une implémetation haute performance des fonctions de présentation et de contrôle de transmission. Nous montrons que les optimisations basée sur le principe du Traitement en Couche Integrée (Integrated layer Processing) permettent de réduire le coût des opérations de manipulation de données généralement assez coûteuses. Nous présentons aussi un cadre pour la conception de modules de communication intégrés en se basant sur les principes de Mise en Trame au Niveau Application (Application Level Framing) et du Traitement en Couche Integrée.

Mots-clé : Protocoles à haut débit, ALF, ILP

1 Introduction

The emergence of high speed networks has shifted the performance bottleneck from the transmission media bandwidth to the processing time necessary to execute higher level protocols. In particular there is a concern that the existing transport and presentation protocols are the most processing demanding protocols. The existing standard protocols were defined in the seventies. At that time, communication lines had low bandwidth and poor quality which was compensated for by elaborate protocol functions. The emergence of high speed networks such as FDDI, DQDB and in the near future B-ISDN motivated many research work on the design of more efficient communications protocols (at both transport and presentation level):

- The tuning of existing transport protocols (e.g. TCP extensions for high speed paths [1], TP4 1992 revision [2], [3], [4]).
- The design of a new transport protocols (e.g. XTP [5]).
- The performance optimization of the presentation coding and decoding routines either by defining new "light weight transfer syntaxes" ([6]) or by implementation enhancements of the standard BER routines ([7]).

At the same time, the application environment is changing. New applications (e.g. audio and video conferencing, shared white board, supercomputer visualization etc.) with specific communication requirements are being considered. Depending on the application, these requirements may be one or more of the following: (1) high bit rates, (2) low jitter data transfer, (3) the simultaneous exchange of multiple data streams with different "type of service" such as audio, video and textual data, (4) a reliable multicast data transmission service, (5) low latency transfer for RPC based applications, etc.

The above requirements implies the necessity to revise the service model in order to fulfill the application needs. In fact, the applications had to choose between the connection-less and connection oriented transport services. In both cases, the application needs are expressed in terms of quality of service (QOS) parameters such as the transit delay or the maximum throughput. However, the applications need more than a set of QOS parameters to control the transmission. The applications require to be involved in the choice of the control *mechanisms* and not only the parameters of a unique transport service. The transport service model needs revision. the design of the so-called "communication subsystems" tailored to provide the service required by the application has been proposed. Early work in this domain proposed to design "light weight" "special purpose" protocols for specific application needs (e.g. NETBLT [8] for bulk data transfer and VMTP [9] for transactional applications). This approach has a major limitation: the diversity of the applications increases the complexity of the "communication subsystem" by the support of several communication

protocols. A more recent approach is to synthesize customized protocols from "building blocks" implementing elementary protocol functions such as: flow control, error control, connection management (e.g [10], [11], [12], [13]). The aim of these ongoing research activities is to automatically generate the communication subsystem from high-level specifications (written in a "suitable" language). The synthesis of "fine grain" protocol functions should replace the coarse grain protocol choice (e.g. TCP or UDP). These activities are focused on transport level functions and do not consider the application synchronization and data presentation requirements, in order to reduce the complexity of the communication subsystem. We think that the integration of the session and presentation with the transport functions in order to generate a single protocol graph for the application will result in increased performance gain. This aspect is discussed in section 5.

Good implementations techniques represent one of the most important factors in determining the performance of a given protocol [14]. These techniques depend on the environment and not on the protocol itself. The performance of workstations has increased with the advent of modern RISC architectures but not at the same pace as the network bandwidth during past years. Furthermore, access to primary memory is relatively costly compared to cache and registers and the discrepancy between the processor and memory performance is expected to get worser. The memory access is expected to represent a bottleneck [15]. On the other hand, the workstations performance may also be increased by the use of multiprocessor systems. However, current protocol implementations do not exploit this parallel processing potential.

The proposed solutions focused on the enhancement of the protocol implementation performance in a given software or a hardware environment:

- Outboard protocol processors [16] or hardware protocol implementations (early work on XTP/Protocol Engine [17]) were proposed to discharge the main processor from "transport layer" operations. In addition, it was argued that hardware support allow to perform costly operations like checksumming efficiently. However, the reduced flexibility of the hardware approach is a critical constraint because of the applications diversity. Furthermore, software protocol implementations on modern workstations have shown increased performance, thereby reducing the interest of a hardware solution.

- Parallel implementations of transport protocols were proposed to increase the protocol processing speed [18], [19], [20], [21].

- Integrated layer processing (ILP) [22] was proposed by Clark and Tennenhouse in order to enhance the performance of protocol implementation on RISC workstations. The main concept behind ILP is to minimize the memory reads/loads by combining byte manipulation oriented operations

within one or two processing loops. The merits of this approach will be discussed in more detail in section 5.

- The development of execution environments where protocols can be developed in a modular and yet efficient way (e.g. the x-kernel [23]). Even if the focus within this approach is on flexibility, further performance enhancement may be obtained if the ILP principle is supported.

2 Integrated Protocol Implementation

Protocol processing can be divided into two parts, control functions and data manipulation functions. In the data manipulation part, the actual data of a PDU is read from memory, manipulated and possibly loaded back to memory. Example of data manipulation functions are presentation encoding, checksumming, encryption and compression. In the control part there are functions for header and connection state processing. Jacobson et al. have demonstrated that the control part processing can match gigabit network performance for the most common size of PDUs with appropriate implementations [14]. However, data manipulation functions present a bottleneck [22], [24]. They consist of two or three phases. First a read phase of data, then a manipulation phase followed by a load phase for some functions. For very simple functions, such as checksumming, byte alignment, etcetera, the time to read and write to memory dominates the processing time. For others, like encryption and some presentation encodings, the manipulation time dominates.

The data manipulation functions are spread over different layers. In a naive protocol suite implementation, the layers are mapped into distinct software or hardware entities which can be seen as atomic entities. The functions of each layer are carried out completely before the protocol data unit is passed to the next layer. This means that the optimization of each layer has to be done separately. Such ordering constraints is in conflict with efficient implementation of data manipulation functions [25].

One could accuse the layered model (TCP/IP or OSI) for causing this conflict. However, it is important to distinguish between the architecture of a protocol suite and the implementation of a specific end system or a relay node. There is nothing a priori that requires that the implementation should follow the architectural decomposition. ILP has been suggested for addressing this problem. With integration we mean that several layers are implemented within the same module. This does not mean that the implementation is unstructured. The basic idea behind ILP is to perform all the manipulation steps in one or two processing loops, instead of performing them serially as is most often done today. Within a loop we mean one read to memory, followed by all manipulations and one write back. Hence, time consuming memory references are reduced. To facilitate ILP a protocol architecture should be organized such that the inte-

ractions between the control and data manipulation functions do not interfere with their integration.

The layered protocol architecture may unnecessarily reduce the engineering alternatives available to an implementor. Clark and Tennenhouse have proposed Application Level Framing (ALF) as a key architectural principle for the design of a new generation of protocols [22]. According to this principle applications should break the data into frames (or Application Data Units (ADUs)) meaningful to the application. It is also desirable that the presentation and transport layers preserve the frame boundaries as they process the data. In fact, this is in line with the widespread view that multiplexing of application data streams should only be done once in the protocol suite [26]. The sending and receiving application should define what data goes in an ADU such that the ADUs can be processed out of order. The ADU will be considered as the unit of "data manipulation", which will simplify the processing. Thus, ALF is supporting the ILP principle.

ALF and ILP were proposed in 1990 and there are very few reported implementations done according to these concepts. Results reported in [22] indicated that a MIPS processor performing the copy and the checksum on data can run at 60 Mbps if these functions were done separately and at 90 Mbps if they were combined in a hand coded unrolled loop. The effect would be much more pronounced if several of functions necessary to the application (like e.g. presentation encoding, encryption and checksumming) were combined. Other work, ([24], [27], [28]) have demonstrated that there is a performance benefit with ILP.

We worked on the design and implementation of ALF/ILP based environment. Our goal is to integrate several of the data manipulation functions in a complete, operational stack in order to understand the architectural implications and the achievable speedup.

In this paper, we present the approach we follow to reach this goal:

- We worked on the enhancement of costly presentation encoding and decoding routines. The processing cost of byte oriented manipulation operations reduces the gain of ALF/ILP. Therefore any ALF/ILP based system should have an efficient presentation layer. In section 3 we detail the work on the enhancement of the presentation coding and decoding routines.

- In order to experiment integration techniques with transport level functions we performed a prototype implementation of the XTP protocol at the user level. The integration of presentation and transport level functions is more easily achievable if the transport is implemented at the user level.

Section 3 presents the work on the optimization of the presentation layer. Section 4 presents the XTP implementation and discusses the suitability of XTP as a substrate for ILP based implementations. In section 5, we discuss a general

framework for the building of the ALF/ILP based communication subsystems. Section 6 concludes the paper.

3 High speed presentation

Our work was centered around the optimization of the ASN.1 Basic Encoding Rules. The cost of the coding and decoding routines is attributed to the heavy Type-Length-Value oriented coding of ASN.1 BER. This motivated the work on "light weight" (LWS) or XDR-like transfer syntaxes [29] based on three design principles:

- avoid unnecessary information in the encoding,
- use fixed representation when it is possible and,
- simplify the mapping of the elements by using fixed length structures.

The first principle leads to the abandonment of the systematic "Type-Length-Value" encoding, which is replaced by new encoding rules for the ASN.1 constructs; the second and the third principles lead to a set of easily decoded *word oriented* encoding rules. The *word size*, is a parameter that characterize the syntax (16, 32, or 64 bits). The complete definition of these light wight encoding rules is given in [6]. Both BER and LWS syntaxes are supported by the MAVROS ASN.1 compiler developped at INRIA [30].

We have done some experiments to compare the performance of ASN.1 BER and the LWS encoding rules. The ASN.1 specification of the data type used in these tests is:

```
perf-object ::= OPAQUE CHOICE {
        ints[0] SEQUENCE OF INTEGER,
        strings[1] SEQUENCE OF PrintableString,
        real[2] SEQUENCE OF REAL,
        mpdus[3] SET OF MPDU}
```

The test object is either a basic data type such as SEQUENCE OF INTEGER, SEQUENCE OF REAL, SEQUENCE OF PrintableString (mean length of 26 characters per string). In order to test the encoding rules with more complex types, we chose the MPDU format (Message Protocol Data Unit) which defines the envelope of a message according to the 1984 version of the P1 Protocol (CCITT Recommendation X.411).

The number of the data values for each type in the performance tests is given in table 1.

The results of performance tests accomplished on a Sun 3/60 are given in table 2.

The LWS coding routines were 1.6 to 5.8 times faster than those of the BER depending on the data types. For basic data types, the improvement is

considerable (specially for the `real` type) because of the relative efficiency of the word oriented coding.

After this first test series we conducted several optimizations by analyzing the coding routines for typical types either "manually" or with the help of "profiling" tools. We tested a number of modifications in our ASN.1 BER coding and decoding algorithms, of which we cite:

- Inline the encodings for some types, and apply the "type reduction" technique in order to remove all "repeated indirections", replacing them as much as possible by direct references to the data type.
- Use "header prediction" techniques to speed up the decoding of tags and that of length fields,
- Speed up the encoding of tags and length field by using systematically the "indefinite length" encoding form.
- use static declaration and better memory management (reduce the calls to `malloc`).

These (and other) optimizations were reported reported in the code generation programs of the ASN.1 compiler, as well as in the "run-time" library. A new series of tests were conducted only with the MPDU type on different workstation types and the performance figures are shown in table 3: Before we analyze these figures, recall that the coding procedure consists in copying the `perf-object` from memory to the cache, looping on processing instructions and then storing the result (BER or LWS streamlined data) in memory. This cycle is reversed for decoding but the processing part is longer due to additional verifications. Therefore, if the overall decoding time is higher than the coding time, we can deduce that the processing speed is the bottleneck. Otherwise, if these times are comparable, then we may say that the memory access overhead (which is the same for both routines) dominates the processing time. In addition, BER encoding requires more processing than the LWS which generates, however, a larger code size.

In the case of initial tests on Sun3, it is clear that the processing speed is the limiting factor for both BER and LWS. In this case, it is advantageous to use the LWS. The optimizations led to a drastic reduction of the BER cost and the optimized Sun3 figures shows that the BER coding time is lower than LWS: what is gained in processing is lost due to the higher size of the data which will transit on the bus. However, the for more CPU consuming routine (decoding) BER is still slower than the LWS.

The results are more interesting with high performance RISC workstations. On the SparcStation 10, coding BER is more efficient than LWS which implies that the memory access dominates the processing time. However, BER decoding is still limited by the CPU performance. Both coding and decoding routines for

LWS are limited by the memory access (due to the large code size loaded and stored).

Similar results are obtained on both Dec 3000 and HP workstations. Note, however, that the coding time is the same for both BER and LWS which means that both memory access and CPU bottlenecks are balanced. For decoding, the processor speed limitation is more pronounced.

The best results are obtained on Dec-alpha where the 64 bit bus enhances the memory access performance. The figures are "classical": BER more costly than LWS and decoding (BER or LWS) is more costly than coding. This is typically due to CPU limitation.

The maximum decoding throughput is about 24.54 Mbps for BER and 88.9 Mbps for LWS on the Dec-Alpha.

From the above results we can learn the following:

- A drastic improvement of the speed of coding and decoding routines for both BER and LWS routines can be obtained by adequate implementation optimization on high performance workstations
- The LWS is interesting if the processor speed is the bottleneck, however, a more compact syntax is desirable in order to reduce the size of the data to be transmitted on the network.
- Memory access is a limiting factor in most cases on RISC workstations. This confirms that integration techniques should result in increased performance on such workstations.

The design of an integrated communication subsystem is a more general problem that still needs to be addressed. The optimized version of the encoding and decoding functions should lead to the implementation of the presentation layer as a filter with "streamlined" *encoding* and *transmission* of application data units.

4 High speed transport mechanisms

We now present our work on the transport level functions. This work has two goals:

- To experiment the effect of integration mechanisms on the performance of transport functions,
- To have a substrate suitable for the building of communication subsystems with support of the ALF/ILP principles.

We chosed to perform a prototype implementation of the XTP protocol. XTP was initially designed to be implemented on specialized hardware with execution efficiency as inherent part of the design process. Therefore, many syntactical and

algorithmic choices of the protocol are oriented to facilitate high speed operation over low error rate networks.

XTP provides a set of mechanism, the application can select the desired type of service according to its own requirements. We implemented XTP in the Unix user level as a transport functionalities library. This library is linked to the application code. The rationale for user level implementation of protocols is given in [31]. We argue that user level implementation does not necessarily have poor performance if proper optimization are adopted as will be shown in section 4.1.

The interest of a user level implementation of XTP is to provide a support for the test of the architectural design described in section 2. In the rest of this section we will describe the XTP implementation. level XTP

4.1 XTP implementation

The implementation runs on an extension of the 4.3 BSD socket interface. The extension of the kernel protocols was done in order to add the support of XTP on top of IP. The implementation architecture is depicted in figure 1.

Two functions **xtp_input** and **xtp_output** on top of IP serve as a demultiplexing layer. They process the **key** field in the XTP header. This minimal kernel support provides a secure allocation of network ports. The provision of "datagramm" XTP sockets enables the application Transmission control thus can be easily User applications have access to the XTP socket interface through the standard system calls (**writev, call attempts to gather an array of buffers of data and readv** scatters

The XTP connection set up, the reliable data transfer and the connection tear down functionalities were implemented. Other XTP functionalities like rate control, route management, multicast procedures have not been implemented. We now present some of the implementation issues, a detailed description of this work is given[32].

Context initialization An application using XTP should initialize a context before sending or receiving data. After this initialization, a field within the context structure points to a control some or This reflects the programmability of the XTP protocol.

Context scheduling An application may have several active contexts in the same time (several point to point associations with different application entities). The application examines the list of "active" contexts to determine the next context to be processed and the corresponding wait timer value as described in [33].

Buffer management The allocation of memory buffers is performed by the application. The sizes of the maximum receive or send buffers are specified in the corresponding fields in the control bloc structure. These buffers are directly used by the application to compute/process data. There is no intermediate transport level buffers. When an XTP packet is received only the packet header is consulted. After the necessary control functions had been performed, the data is copied into the corresponding place within the user level buffer. The system calls `readv` and `writev` are used to accomplish the scatter/gather I/O. According to application requirements (e.g when a complete Application Data Unit has been received) the application is invited to process the message.

Retransmission strategies The selective ACK upon request mechanism used in XTP facilitates the implementation of the well known retransmissions strategies, GoBack-N and Selective repeat. Both strategies have been implemented. During the performance tests presented hereafter the receiver has adopted the selective repeat strategy.

Checksum calculation The 32-bit XTP check function called CXOR, is defined as the catenation of two 16-bit functions: XOR which is a straight "vertical" exclusive-or of each 16-bit short word in a block of information, and RXOR a rotated "spiral" exclusive-or of each 16-bit short word. We experimented the effect of ILP based optimizations on the performance of the checksum algorithm and we had increased performance by a factor of 12 due to these optimizations as will be shown in the section 4.2

4.2 Performance tests

In this section we present performance test results of our prototype XTP. Performance of a kernel TCP BSD implementation will also be presented. The figures should be taken very carefully: the performance of a protocol depends on the environment, implementation tunings and enhancement technique as demonstrated in [7]. Many differences exist between the choices we adopted and the TCP kernel implementation, and the XTP implementation is not as well "tuned" as the TCP kernel implementation. Therefore, it is not meaningful to make precise comparison of both protocols based on the performance figures we have. However, the figures gives an idea on the possible performance of XTP when implemented in the user level.

4.2.1 Performance of the checksum algorithm

The speed of checksum calculation is known to be one of the important factors that determine the performance of a transport protocol [22], [34]. The XTP designers proposed to perform this task in hardware [35]. The following results

show that it is possible to optimize the software implementation of this routine thus removing of the bottlenecks for the protocol performance.

The performance of TCP and ISO TP4 checksum algorithms are given in the first two rows of the table 4 for four different machine hardwares. These are byte or short word oriented calculations. The "standard" implementation of the XTP checksum as a loop of short word oriented exclusive-OR and rotate operations was too slow (maximum throughput of 13.88 Mbps on a SS-10).

We analyzed the cost of the basic operations needed for the the checksum routine in order to determine the most costly functions. The results are presented in table 5. The `rotate(x,n)` macro is defined as `n` circular shifts on the short word `x`. This is a relatively costly operation and should be saved as much as possible.

We performed some hand optimizations on the checksum calculation algorithm. The results are given in table 6. A first optimization (op1) consisted in doing the XORs on words with the same RXOR rotation and saving the rotate operations until the end. This "enhancement" increased the speed slightly on Sun3 but resulted in slower operation on Sparc and Dec workstations. In fact, to implement this modification, we used an array of 16 short words to store the XOR values. As the manipulation operation on the Sun3 was the bottleneck, the saving in the number of rotate operations resulted in increased performance. On the contrary, the memory access on both Dec and Sparc workstations is the bottleneck. Therefore, the load operations of the array elements decreased the performance. A second optimization (op2) consisted in declaring 16 short words instead of the array in order to save indirections (unroll the loop). This resulted in increased performance on all machine types. In a third modification (op3), the number of XOR operations is divided by 2 by XORing the 16 words at the end to determine the CXOR value instead of doing it on the fly. This also increased the overall performance. In (op4) the same optimizations as (op3) are applied with long words calculations and then folding the results down to 16-bits. The results are interesting: the increase factor with regard the third enhancement vary between 2.6 for the DecStation and 5.6 for the SS10. The higher the value of this factor, the more costly is memory access relatively.

4.2.2 Throughput measures

Tests have been performed over both Ethernet and FDDI networks. We implemented simple a simple client/server application where the client sends raw data to the server using UDP, TCP or XTP protocols. For UDP, the client will send numbered packets, the test is considered terminated when the last packet is received at the server. It this packet is lost, the test is considered as non valid. For both TCP and XTP, the test is terminated when all the data is received correctly at the server. The XTP library is linked with both the client and server code.

The throughput is defined in the three cases as the number of received bits divided by the time interval between the reception of the first and the last packet at the server.

The following tables give the results of the tests over both Ethernet and FDDI. The machines running the client and the server are both DECstations 5000/200.

One may be tempted to say that the UDP figures give the raw performance that may be obtained on either Ethernet or FDDI. However, this assertion should be taken with care: packet losses at the **ipqueue** level due the high number of packet transmitted during a test (100 to 500) reduce the throughput as defined here above. The best performance for UDP are obtained for small number of transmitted packets (15 or 20). Over Ethernet, there is no bottleneck at the UDP level: the throughput is limited by the speed of the Ethernet interface. Over FDDI, the maximum raw UDP throughput is also limited by the performance of the FDDI interface. Note that the MTU is 1460 and 4312 octets over Ethernet and FDDI respectively. 9000 octets packets are transmitted as 7 Ethernet packets and 3 FDDI frames and reassembled by the kernel at the UDP/IP layer.

The TCP figures are more interesting: no bottleneck over Ethernet, TCP can easily saturate the 10 Mbps CSMA-CD LAN. However, the performance over FDDI are limited by the control mechanisms of TCP. The highest throughput (20.74 Mbps) is lower than the best UDP performance. The socket receive buffers have been set to 65535 octets in order to increase the window of TCP. TCP Packets shorter than 1024 octets show poor performance even over FDDI. The limitation comes from the buffering mechanism within the kernel: small buffers of 128 octets are used, thus limiting the overall performance of the transmission. For packets longer than 1024 octets, the performance are quite similar with a tendency to decrease.

For the XTP tests, one should distinguish between two notions: ADU and NDU as they are defined in [36]. ADU or *Application Data Unit* is the basic unit of data exchange between application entities. NDU or *Network Data Unit* is the unit of data processing and switching within the network. For the purpose of these tests NDUs over Ethernet are 1400 bytes long and NDUs over FDDI are 4000 bytes long. The packet sizes in table 9 are ADU sizes. Note that the results does not vary a lot with the ADU size, as the same NDU size is used. The most important result shown in this table is that XTP can saturate Ethernet and operate up to a speed of 21 Mbps over FDDI. This was made possible because of the following factors:

- minimize data copying by the use of writev and readv,
- optimize the checksum calculation by ILP oriented optimizations,
- minimal overhead by return key based context look up in the kernel

This, in fact, optimize the input processing in the normal case (no errors). This optimization is a capital condition to enhance protocol performance.

These results show that it is possible to have high speed communication with user level implementation of transport protocols. The throughput figures should not be taken as an indication of the performance of XTP with all control mechanisms implemented. However, the other control mechanisms (rate control, route management) do not manipulate data and they may be considered to have minimal additional cost. The good performance of our XTP implementation with the complete error and flow control procedures is an encouraging step in the direction of user level implementation of protocols. Such high performance transport implementation will provide building blocks that will facilitate the configuration of application specific communication subsystems.

However, some problems still to be solved before user level implementations provide optimal results: the execution environment should be more adapted to the support of network protocols by providing better memory management and process scheduling than Unix.

5 Communication subsystems design

XTP is a step towards the definition new high performance communication subsystems. XTP proposed a set of well founded protocol enhancement mechanisms like: (1) fixed-length and position fields, (2) efficient 1-way implicit handshake connection setup, (3) efficient context lookup with a key-based scheme, (4) data alignment on 4-byte boundaries, (5) selective retransmission, (6) acknowledgment control (7) a checksum algorithm that can be computed at high speed. These mechanisms contribute to the enhancement of the protocol performance. To give only one example, note the important speed increase factor for checksum calculation using 4-byte long words oriented operations. Some of these mechanisms are algorithmic enhancements like (2), (3) and (4) and some other (5, 6, 7) are driven by an important design philosophy of XTP: the programmability of the protocol mechanisms by the end systems. This programmability facilitates the profiling of the transmission control protocol by the transmitter. XTP provides a set of functional enhancements like rate control, priorities based scheduling, reliable multicast, No-error mode. The application selects the desired functions or policies via the `control block` structure.

However, if the XTP "programmability" is in line with ALF/ILP principles, it is not clear *how* the application should configurate the elementary transport and presentation "building blocks" in order to generate the complete communication subsystem. XTP can serve as a substrate for the implementation of this communication subsystem. The problem of the service selection is left to the application. Other design studies should determine how the elementary building blocks can be grouped together in order to provide the desired service with the best performance.

The application selects the options that govern the data exchange. In other words, this approach means that the application will select the policy used for the transmission control. In fact, only the application has sufficient knowledge about its own requirements to optimize the parameters of a data exchange in a convenient way. The application can easily adapt to the network resources changes by appropriate steps: instead of reducing the window size at the transport level in response to a congestion indication a videoconference application may chose to degrade either the quality or the frequency of the images in order to adapt to the available bandwidth.

This approach is not consistent with the layered OSI model, where clear separation of data transmission control mechanisms (lower layers) and data processing functions (higher layer) is made.

The OSI approach discharges the application from the transmission control functions by defining the transport service . The applications are *transport service users* according to the layered reference model. On the contrary, the integrated design and implementation approach put the application to define how the application level parameters will be mapped onto network parameters and control functions. According to the diversity of the applications two solutions are possible:

- either we define application classes and we select appropriate control functions to be integrated to these applications (i.e. we define typed transport services as proposed in [37]).
- or we use a suitable specification language to express the applications needs and we design a generic tool to derive the configuration of the protocol functions automatically.

The first solution means an extension of the transport service. There has been considerable work within the OSI95 project [38] and elsewhere [39], [40] to define high speed transport service(s).

Another possible way to solve the profiling problem is to have automatic generation of communication profiles based on a set of enhanced mechanisms (at the presentation and transport level) as described in sections 3 and 4. This corresponds to an horizontal approach to the layered architecture i.e. applications select and combine control functions based on the service parameters and on the cost of these functions. Some tools are required to implement this approach: (1) a specification language to express the application requirements, (2) an efficient implementation of elementary building blocks or generic protocol mechanisms, (3) a profiling tool that configures the appropriate implementation based on the service specification and on the generic protocol mechanisms.

We are working on the design and implementation of such tool within the HIPPARCH project: a European-Australian collaboration. The main objectives of the project are cited in the following:

1. The design of an architecture for integrated design of communication protocols. Based on the knowledge available from integration techniques for communication protocols, techniques for grouping protocol functions in order to meet specific application requirements will be proposed.
2. The definition/selection of a specification language to describe the application requirements.
3. The design and implementation of a compiler to automatically generate the customized communication subsystem based on integrated protocol modules.
4. The design and implementation of an efficient runtime system (execution environment) for the code generated by the compiler.
5. The building of prototype integrated implementation of communication protocols for test applications.

6 Conclusion

In this paper, we presented our work on high performance implementation of presentation and transport functions. The experiments show that it is possible to have increased performance by proper implementation techniques. We showed specifically that an important speed up factor can be obtained by applying ILP integration techniques for costly data manipulation functions. Another important result is the relatively acceptable performance of user level protocol implementations. Next step is to develop a compiler that combine automatically and according to the application needs elementary building block to generate customized communication subsystem verifying the ALF/ILP principles.

References

[1] V. Jacobson, R. Braden and L. Zhang. TCP Extension for High-Speed Paths, RFC 1185.

[2] X.224 Recommendation, *Transport Protocol Definition For Open Systems Interconnection (OSI),* Blue Book, Volume VIII, Fascicle VIII.5.

[3] Richard W. Watson, Sandy A. Mamrak. Gaining efficiency in transport services by appropriate design and implementation choices. *ACM transactions on computer systems,* Vol 5, No. 2, May 1987, pp 97-120.

[4] R. Colella, R. Aronoff, K. Mills. Performance Improvements for ISO Transport. Ninth Data Communication Symposium, *ACM SIGCOMM, Computer Communication Review,* Vol. 15, No. 5, September 1985

[5] *XTP Protocol Definition, Revision 3.6,* PEI 92-10, January 1992, xtp-request@pei.com, Protocol Engines Incorporated, Santa Barbara, CA 93101, USA.

[6] Christian Huitema. Definition of the Flat Tree Light Weight Syntax (FTLWS). Internal Document, INRIA Sophia Antipolis, July 1990.

[7] Walid Dabbous et al. Applicability of the session and the presentation layers for the support of high speed applications. *Rapport Technique RT 144*, Institut National de Recherche en Informatique et en Automatique, October 1992.

[8] Clark D., Lambert M., Zhang L., NETBLT: a high throughput transport protocol *Proceedings ACM SIGCOMM '87,* Stowe, Vermont, pp. 353-359, August 1987.

[9] Cheriton D. R., VMTP: a transport protocol for the next generation of communication systems, *Proceedings ACM SIGCOMM '86,* Stowe, Vermont, pp. 406-415, August 1986.

[10] Donald F. Box, Douglas C. Schmidt and Tatsuya Suda. 4^{th} IFIP Conference on High Performance Networking, Liège, Belgique, Décembre 1992.

[11] D. Schmidt, B. Stiller, T. Suda, A.N. Tantawy, and M. Zitterbart. Language Support for Flexible Application-Tailored Protocol Configuration. *Proceedings of LCN '93* TBA.

[12] Philipp Hoschka. Towards tailoring generic protocols to application specific requirements. In *Proceedings of INFOCOM '93.*

[13] Mark B. Abbott, Larry L. Peterson. A Language-Based Approach to Protocol Implementation. *IEEE/ACM Transactions on Networking,* Vol 1, No 1, Feb. 1993.

[14] David D. Clark, Van Jacobson, John Romkey, Howard Salwen. An analysis of TCP processing overhead. *IEEE Communications Magazine,* June 1989, pp. 23-29.

[15] Peter Druschel, Mark B. Abbott, Michael A. Pagels, and Larry Peterson. Network Subsystem Design. IEEE network, July 1993, pp. 8-17.

[16] Hemant Kanakia, David R. Cheriton. The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors. *Proceedings of the SIGCOMM'88,* Stanford, CA, 1988, pp. 175-187.

[17] Chesson G., XTP/PE Design Considerations, in *Protocols for High-Speed Networks,* H. Rudin, R. Williamson, Eds., Elsevier Science Publishers/North-Holland, May 1989.

[18] T. Braun and M. Zitterbart. Parallel Transport System Design. 4^{th} IFIP Conference on High Performance Networking, Liège, Belgique, Décembre 1992.

[19] Erich Rütsche and Matthias Kaiserwerth. TCP/IP on the Parallel Protocol Engine. 4^{th} IFIP Conference on High Performance Networking, Liège, Belgique, Décembre 1992.

[20] T. F. La Porta and M. Schwartz. A high-Speed Protocol Parallel Implementation: Design and Analysis. 4^{th} IFIP Conference on High Performance Networking, Liège, Belgique, Décembre 1992.

[21] M. Björkman and P. Gunningberg. Locking Effects in Multiprocessor Implementation of Protocols. Submitted to ACM SIGCOMM'93, March 1993.

[22] David D. Clark and David L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. *Proceedings ACM SIGCOMM '90,* September 24-27, 1990, Philadelphia, Pennsylvania, pp. 200-208.

[23] Hutchinson, N., Peterson, L. Design of the x-kernel Proceedings of SIGCOMM 88 August 1988 pp 65-75

[24] P. Gunninberg, C. Partridge, T. Sirotkin, B. Victor. Delayed evaluation of gigabit protocols. In *Proceedings of the* 2^{nd} *MultiG Workshop,* 1991.

[25] Ian Wakeman, Jon Crowcroft, Zheng Wang, and Dejan Sirovica, *Layering considered harmful,* IEEE Network January 1992, p. 7-16.

[26] David L Tennenhouse. Layered Multiplexing considered harmful. *Proceedings of the IFIP Workshop on Protocols for high speed networks,* Zurich, Switzerland, 9-11 May, 1989.

[27] C. Partridge and S. Pink. A Faster UDP. *Submitted to IEEE Transaction on Networking.*

[28] Mark B. Abbott, Larry L. Peterson. Automatic Integration of Communication Protocol Layers. TR 92-24, Department of Computer Science, University of Arizona.

[29] Christian Huitema, Assem Doghri. Defining faster transfer syntaxes for the OSI Presentation Protocol. *Computer Communication Review,* Vol 19, No 5, Oct 1989, pp. 44-55.

[30] MAVROS, Highlights on an ASN.1 compiler, INRIA research note, May 1991.

[31] Chandramohan A. Thekkath, Thu D. Nguyen, Evelyn Moy, and Edward D. Lazowska. Implementing Network Protocols at User Level. *Proceedings of the SIGCOMM'93,* San Francisco, CA, 1993, pp. 64-73.

[32] Walid Dabbous, Christian Huitema. XTP implementation under Unix. Technical report, OSI-95 project deliverable INRIA-6, INRIA, Sophia Antipolis.

[33] G. Varghese, T. Lauck, Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of a Timer Facility. *Proceedings of the* 11^{th} *ACM symposium on Operating Systems Principles,* ACM Operating Systems Review, Austin, TX, November 1987.

[34] Marshall T. Rose. *The Open Book, a Practical Perspective on OSI.* Prentice-Hall, 1990.

[35] W. Timothy Strayer, Bert J. Dempsey, Alfred C. Weaver, *XTP: The Xpress Transfer Protocol,* Addison-Wesley, 1992.

[36] James R. Davin. "ALF Protocol Specification", Internal draft, M.I.T. Laboratory for Computer Science, February 1992.

[37] Dabbous W., *Etude des protocoles de contrôle de transmission à haut débit pour les applications multimédias,* PhD Thesis, Université de Paris-Sud, March 1991.

[38] L. Léonard. "Enhanced Transport Service Specification". *Deliverable ULg-4,* OSI 95 project, October 1992.

[39] ANSI X3S3.3. Document 91-265: High Speed Transport Service definition. Expert Contribution to ISO/IEC JTC1 SC6/WG4, September 1991.

[40] Christophe Diot, Patrick Coquet and Didier Stunault. "Specifications of ETS the Enhanced Transport Service". *Rapport de Recherche RR 907-I,* LGI-Institut IMAG, May 1992.

Test identifier	Data Type	Number of test values
`ints`	SEQUENCE OF INTEGER	400
`real`	SEQUENCE OF REAL	200
`strings`	SEQUENCE OF PrintableString	124
`mpdus`	SET OF MPDU	18

Table 1: Number of Data values for each type

Coding routines	`ints`	`reals`	`strings`	`mpdus`
BER (total)	5999	13666	7833	62330
BER (per element)	15	68.33	63.2	3462
LWS (total)	1333	2333	5499	37998
LWS (per element)	3.33	11.66	44.35	2111

Table 2: Coding time in μs for different types.

	Coding time (μs)		Decoding time (μs)		Size (octets)		Decoding (Mbps)	
	BER	LWS	BER	LWS	BER	LWS	BER	LWS
Initial (Sun3)	3462	2111	16212	13527	480	1011	0.24	0.6
Sun3	1861	2018	6944	6398	540	1029	0.62	1.29
SS 10/30	112	199	318	225	540	1029	13.58	36.59
Dec 5000/300	208	208	372	212	540	1029	11.6	38.8
HP 9000/715	100	100	339	155	540	1029	12.74	53.1
Dec alpha	74	55	176	92.6	540	1029	24.54	88.9

Table 3: Speed of the presentation routines.

Checksum	Sun 3/60	Sparc IPX	DEC 5000/200	SS-10/41
TCP	2.61 Mbps	13.3 Mbps	30.52 Mbps	31.75 Mbps
ISO	1.21 Mbps	6.50 Mbps	10.69 Mbps	12.46 Mbps
XTP (std)	1.36 Mbps	7.02 Mbps	13.67 Mbps	13.88 Mbps

Table 4: Maximum throughput with different checksum algorithms

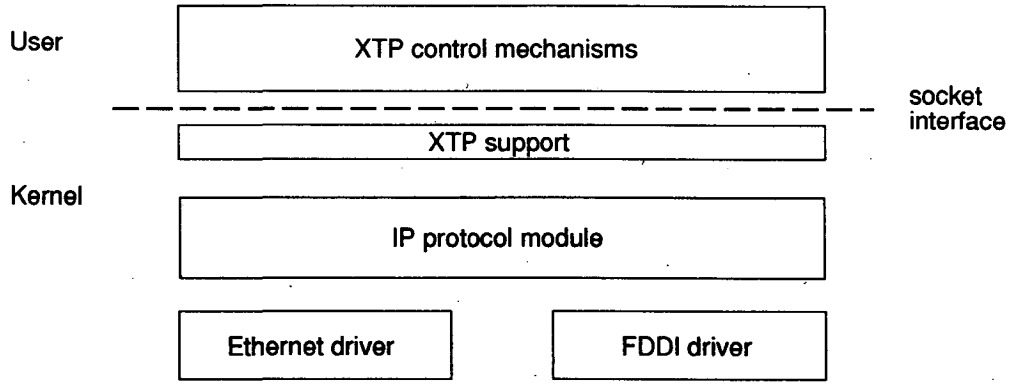

Figure 1: General architecture of the XTP implementation

Operation	Sun 3/60	Sparc IPX	DEC 5000/200	SS-10/41
rotate(x,1)	6.24 μs	1.00 μs	0.687 μs	0.651 μs
Short XOR	4.00 μs	0.88 μs	0.445 μs	0.45 μs

Table 5: Cost of the basic checksum operations (in a loop)

Checksum	Sun 3/60	Sparc IPX	DEC 5000/200	SS-10/41
XTP (std)	1.36 Mbps	7.02 Mbps	13.67 Mbps	13.88 Mbps
XTP (op1)	1.46 Mbps	6.83 Mbps	10.93 Mbps	13.14 Mbps
XTP (op2)	2.95 Mbps	16.54 Mbps	36.96 Mbps	28.47 Mbps
XTP (op3)	5.78 Mbps	31.25 Mbps	66.06 Mbps	52.63 Mbps
XTP (op4)	22.59 Mbps	172.05 Mbps	173.53 Mbps	296.3 Mbps

Table 6: Effect of optimizations of the XTP checksum algorithm

Packet size	Ethernet	FDDI
1000 o	6.90 Mbps	7.02 Mbps
1400 o	9.57 Mbps	10.32 Mbps
2000 o	9.52 Mbps	13.22 Mbps
3000 o	9.66 Mbps	16.66 Mbps
4000 o	9.47 Mbps	29.03 Mbps
4600 o	9.32 Mbps	24.53 Mbps
9000 o	9.21 Mbps	24.66 Mbps

Table 7: Maximum throughput for UDP

Packet size	Ethernet	FDDI
1023 o	7.25 Mbps	8.43 Mbps
1024 o	8.35 Mbps	19.05 Mbps
1400 o	9.91 Mbps	20.74 Mbps
2000 o	9.82 Mbps	18.86 Mbps
3000 o	9.16 Mbps	18.04 Mbps
4000 o	9.00 Mbps	17.60 Mbps
5000 o	8.71 Mbps	15.73 Mbps

Table 8: Maximum throughput for TCP

ADU size	Ethernet	FDDI
2000 o	9.62 Mbps	18.01 Mbps
4000 o	9.65 Mbps	19.72 Mbps
8192 o	9.71 Mbps	20.38 Mbps
16384 o	9.77 Mbps	21.24 Mbps

Table 9: Maximum throughput for XTP

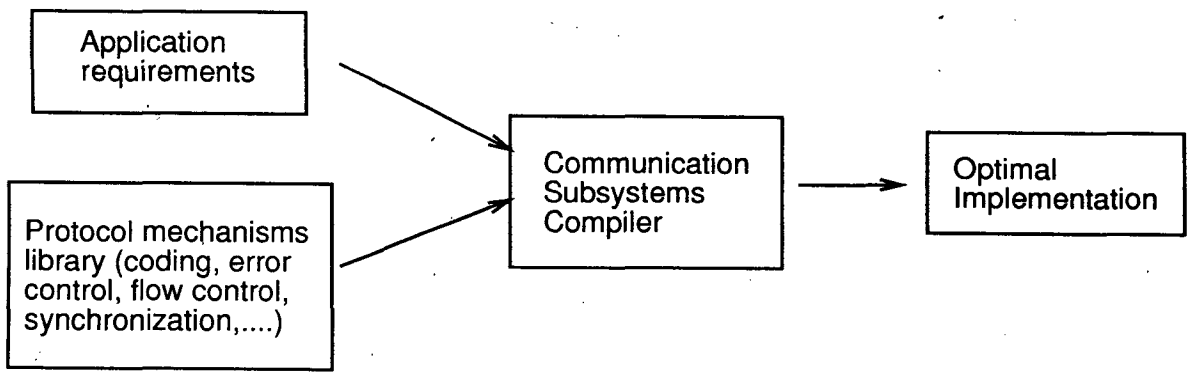

Figure 2: General architecture for automatic communication subsystems generation

Unité de Recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

Unité de Recherche INRIA Lorraine Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)
Unité de Recherche INRIA Rennes IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399