CTMP: A Configurable Token-based Multicast Protocol

Marcelo Dias Nunes **Otto Carlos M. B. Duarte**

Grupo de Teleinformática e Automação - GTA
Universidade Federal do Rio de Janeiro
COPPE/EE - Programa de Engenharia Elétrica
P. O. Box 68504 - 21945-970 - Rio de Janeiro - RJ - Brazil
FAX: +55 21 290.6626 - e-mail: bill, otto@gta.ufrj.br

Abstract

This paper presents a proposal and an implementation of a programmable protocol that supports statistically reliable multicast communications with a centralized token-based access management. Among other services, it offers point-to-point connection establishment and release as well as multicast group membership with the possibility of selection from a set of communication's mechanisms (checksum computing, window and rate-based flow control, error recovery through go-back-N and selective retransmission, bucket algorithm), using services provided by the MAC sublayer in point-to-point and multicast configurations. The implementation follows a high performance architecture. The results show that the protocol achieves high throughputs and seems to fit well for different traffic types: 8.65 Mbit/s and 4.87 Mbit/s for minimum and maximum processing overheads, respectively, over a point-to-point connection between two SPARCstations-20 via Ethernet.

1 - Introduction

The fast development of VLSI technologies and optical fiber transmission have significantly increased the communications network's bandwidth, reaching transmission rates up to hundredths of megabits per second. It is expected that, in a near future, wide area networks reach throughputs of a few gigabits per second. Definitely, the bottleneck has shifted from bandwidth to protocol processing, limiting an application's optimal bandwidth usage.

Besides, the services provided by B-ISDN support a wide range of applications: file transfer, client/server transactions, datagrams, multicast transmissions and real time applications.

Many protocols have been proposed to support high speed requirements, known as light-weight protocols [10, 12, 13]. Some of these protocols and their main contributions are:

- **Datakit** [6]: introduced the concept of sender-driven transmissions;
- **Delta-t** [21]: introduced the concept of implicit connection management;
- **NETBLT** [7]: introduced the rate control mechanism;
- **VMTP** [5]: transaction-oriented protocol;
- **XTP** [17]: light-weight protocol designed for hardware implementation; adopts the bucket algorithm for error control in multicast connections. This protocol combines the main features from the above protocols;
- **ST-II** [20] and **RSVP** [22]: allow quality of service parameters negotiation through resource reservation mechanisms;
- **MTP** [2]: token-based point-to-multipoint protocol;
- **RAMP** [4]: provides different levels of QOS according to the desired level of reliability.

Although these protocols try to achieve high performance by simplifying their processing overhead, each one has its specific characteristics, which make them best suited for a particular class of applications.

Thus, this paper defines a service and specifies a generic **C**onfigurable, **T**oken-based, statistically reliable **M**ulticast **P**rotocol (CTMP) [13, 15, 16]. One of the fundamental design steps was a thorough analysis of high performance communication mechanisms:

- **connection management**;

- **acknowledgment generation**;
- **flow control**;
- **error control**.

CTMP was developed within a specialized implementation architecture frame [3, 11, 13, 14], resulting in a significant throughput performance on the target environment. This environment consisted of two Sun SPARCstations-20 connected to a 10 Mbit/s Ethernet LAN.

Section 2 presents design decisions based upon the analysis mentioned above. Section 3 presents the main features of the protocol: service primitives, PDU formats and multicast connection management procedures. Section 4 presents an overview of the implemented system and section 5 describes the performance analysis methodology as well as the main results. Finally, conclusions are presented in section 6.

2 - Design decisions

The envisaged CTMP's facilities include reliable point-to-point as well as statistically reliable multicast data transfer. Furthermore, different mechanisms should be provided in order to support a wide range of applications that generate distinct traffic patterns.

A comparative analysis among connection management, acknowledgments, flow control and error control mechanisms was performed to help decide upon which mechanisms would best fit a flexible service. This section presents CTMP's design decisions based on this analysis.

- **Connection management**: CTMP adopted a handshake mechanism for point-to-point connection establishment and release. Multicast connections have no explicit establishment or release. A host may join or leave a group any time it desires without informing any other host, as it will be explained further. The major handshake's advantage is the possibility to negotiate some connection parameters, such as flow and error control options (including no flow or error control at all), because different bandwidth and reliability requirements, typical of current multimedia applications, claim for a **configurable** protocol.
- **Acknowledgments**: due to the complexity imposed by timer handling (mandatory whenever sender-independent acknowledgment generation is present) a sender-driven acknowledgment

generation was implemented. Explicit commands issued by the transmitter attenuates the receiver's processing overhead. Previous experience has shown [3, 11, 14] that the receiver is overburdened by a heavier processing per message.

- **Flow control**: window-based flow control is a means of controlling transmission and reception buffers in order to avoid memory overrun. On the other side, rate-based flow control seems to be best suited for applications where tighter throughput constraints are required, like constant bit rate (CBR) applications. However, timing mechanisms must be accurate enough to match the effective rate with the desired one. As far as possible, the underlying network must also interact with the protocol so that rate and burst values will be constantly updated. If the network does not provide such a facility, rate and burst must be negotiated at connection establishment. Thus, both mechanisms are valid as they address different communication aspects and are both implemented in CTMP.

- **Error control**: the responsibility over the most adequate error recovery scheme was left for the application, according to its needs. Therefore, CTMP provides two optional error recovery schemes, **go-back-N** and **selective retransmission**, with automatic repeat request (ARQ). PDU sequence numbers enable loss detection and checksums provide for error detection. As in XTP, there are separate checksums for PDU header and data segment, although only the header checksum is mandatory in every PDU.

- **Multicast connections strategy**: CTMP implements statistically reliable multicast connections. At least theoretically, this allows unlimited group sizes, as no member (sender or receiver) needs to know any information about the other members. When a host decides to send messages to a group, it must only know that group's address. The bucket algorithm [18], implemented in CTMP, is an error control scheme designed for point-to-multipoint connections that enables statistical reliability. CTMP also implements a token management strategy that allows for a subset of the whole group to act simultaneously as transmitters. There are N tokens, where N is less than the maximum number of members.

3 - Service definition and protocol specification

3.1 - Service primitives

CTMP's facilities are provided by a set of service primitives. The connection-oriented unicast service is divided in connection establishment, data transfer and connection release phases. The multicast service is connectionless in order to simplify the membership management. This service implies an underlying capability to support multicast addresses (Ethernet or IP Multicast). Two service primitives are offered for joining or leaving a group.

A quite original multicast transmission management scheme is supported by CTMP's service. It provides a centralized multiple-token management mechanism designed to allow for a great number of participants to take part in a conference. The user interface fits in OSI's Session-Transport interface as it contains typical transport protocol functionalities, like segmenting and reassembling, as well as end-to-end reliability guarantees. Table 1 briefly describes CTMP's service primitives.

Table 1: Service primitives.

Primitives	Description	Parameters
T-CONNECT.Request	Point-to-point connection establishment request	Destination address Source address Quality of service
T-CONNECT.Indication	Point-to-point connection establishment indication	Destination address Source address Quality of service
T-CONNECT.Response	Point-to-point connection establishment response	Responding address Response (acceptance or refusal) Quality of service
T-CONNECT.Confirm	Point-to-point connection establishment confirmation	Responding address Response Quality of service
T-DISCONNECT.Request	Point-to-point connection release request	Release type (normal or abrupt)
T-DISCONNECT.Indication	Point-to-point connection release indication	Reason
T-DISCONNECT.Response	Point-to-point connection release response	n/a
T-DISCONNECT.Confirm	Point-to-point connection release confirmation	n/a
T-DATA.Request	Data transfer request	User data
T-DATA.Indication	Data reception indication	User data
T-MULTIPOINT.Request	multicast group adhesion command	Category (server or non-server) Quality of service
T-TOKEN.Request	Transmission token request	Destination address Source address

Table 1 (continued)

Primitives	Description	Parameters
T-TOKEN.Indication	Token reception or devolution indication	n/a
T-TOKEN.Response	Token devolution command	n/a
T-LEAVE.Request	multicast group abandonment command	n/a

3.2 - The Configurable Token-based Multicast Protocol

Many features required by high performance protocols have been previously implemented in the Xpress Transfer Protocol (XTP). These features comprise fixed size headers, separate header and data segment checksumming, implicit connection setup, bucket algorithm, 32 bit alignment, etc. Therefore, CTMP follows XTP's basic PDU formatting and naming, adequately adapted to support its specific needs.

3.2.1 - PDU formats

3.2.1.1 - Header

All CTMP's PDUs contain the same fixed size, seven-field header (32 bit fields), as shown in Fig. 1.

Command field: this field (Fig. 2) contains information about PDU type (1 byte), number of padding bytes (1 byte) and configuration options (2 bytes).

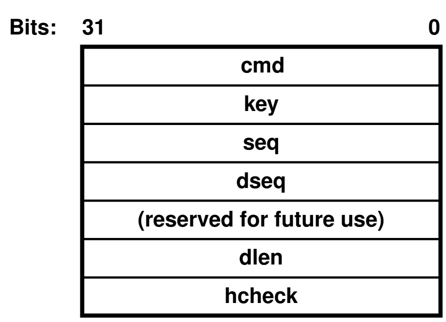

Figure 1: PDU header.

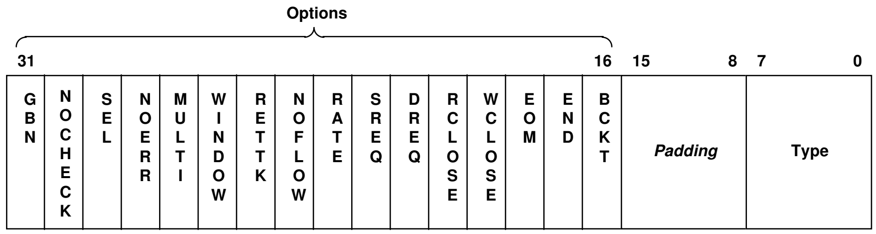

Figure 2: *cmd* field.

PDU types:

- 0x00: DATA user data;
- 0x01: CNTL control information;
- 0x02: FIRST point-to-point connection establishment;
- 0x04: TOKEN multicast connection token;
- 0x06: PASS token request;
- 0x08: ERROR error diagnostics.

Configuration options:

- `GBN` enabled: go-back-N error recovery mechanism.
- `NOCHECK` (*Suspend CRC checking*) enabled: no checksumming over data segment.
- `SEL` (*Selective retransmission*) enabled: selective retransmission error recovery mechanism.
- `NOERR` (*Suspend retransmission*) enabled: no error control.
- `MULTI` (*Multicast operation*) enabled: multicast connection option.
- `WINDOW` enabled: window flow control mechanism.
- `RETTK` (*Return Token*) enabled: token devolution command, issued by the token server.
- `NOFLOW` (*Suspend flow control*) enabled: no flow control.
- `RATE` enabled: rate control mechanism.
- `SREQ` (*Status request*) enabled: sender-driven acknowledgment request. An ack CNTL PDU must be immediately returned.
- `DREQ` (*Delivery status request*) enabled: similar to `SREQ`, but the CNTL PDU should be returned only after data delivery.
- `RCLOSE` (*Reader closed*) enabled: point-to-point "read-channel closed" indication.
- `WCLOSE` (*Writer closing*) enabled: point-to-point "write-channel closed" indication.
- `EOM` (*End of message*): must be enabled in the last message's DATA PDU.

- `END` enabled: indicates connection closing is complete.
- `BCKT` enabled: bucket error recovery algorithm.

Key field: connection identificator.

Seq (*Current sequence number*) field: DATA PDU sequence number.

Dseq (*Delivered sequence number*) field: meaningful to DATA and CNTL PDUs, indicates that all DATA PDUs numbered up to, but not including, *dseq* have been correctly delivered to the user. When received by the sender, indicates that all PDUs numbered up to, but not including, *dseq* that are stored for retransmission may be released.

Dlen (*Data length*) field: represents the total number of bytes of the data segment, including any occasional padding bytes.

Hcheck (*Header checksum*) field: checksum computing over the header. It is mandatory and cannot be disabled by the `NOCHECK` bit.

3.2.1.2 - Trailer

The PDU's trailer is a four byte field *dcheck*. This field is the checksum computing over the data segment, that has a variable size according to the PDU. For example, the data segment in DATA PDUs contains only user data, while the data segment in CNTL PDUs contains lots of fields corresponding to relevant state variables, such as sequence numbers, rate control parameters, etc. (a more detailed description of each PDU's data segment will be omitted at this paper, and may be found in [13]). *Dcheck* computing is only optional for DATA PDUs, according to the `NOCHECK` bit status. Fig. 3 shows a generic CTMP's PDU format.

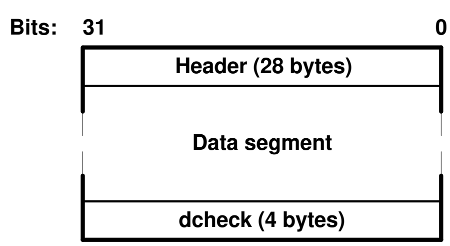

Figure 3: Generic PDU format.

3.2.2 - Multicast connection management procedures

CTMP's procedures for point-to-point connection management and data transfer are straightforward and will thus be omitted at this paper. They may be found in [13]. This section will then explore multicast connection management procedures in detail.

3.2.2.1 - Multicast connection establishment

As it was explained before, there is no explicit multicast connection establishment. The main CTMP's goal is to offer a **dynamic** group communications service: hosts can join or leave a group at any time, without interfering with communication. CTMP performs a centralized group management. Each group must have a transmission token server or centralizer. Its address should be well known by all network hosts.

The token server specifies a group address according to the underlying network (IP Multicast, Ethernet or whatever supports a multicast addressing support). Any host that knows this address may send data to the group, since it possesses a transmission token, and receive data addressed to the group, since it has been adequately configured. The centralization was adopted as a means of simplifying general processing and state machines. This becomes evident when CTMP's multiple token management is considered, wherein a timer is associated to each granted token. These timers are maintained only at the server, releasing sender hosts from this extra overhead, as they already keep their own timers for rate control and bucket algorithm procedures. It is important to notice, however, that the server is not just a simple monitoring entity. It also has the ability to send and receive data.

Before taking part in a multicast group, any host must configure its transmission options, which are the following:

- checksum computing over user data in DATA PDUs;
- rate control (rate in bytes per second and burst in bytes);
- bucket algorithm for error control;
- go-back-N error recovery (imposed by bucket algorithm's utilization).

Such parameters are not negotiated with any other host as each group member only controls its own data transmission. For example, a host may decide to disable error control over its own data flow, but it cannot refuse to respond to any other host's request for acknowledgment.

When this setup is completed, a T-MULTIPOINT.Request primitive is generated, shifting the state machine immediately to the "receive-ready" state, when it can only receive data (it does not have the permission to send data yet because it has not even requested the token). A command is also issued to the underlying layer in order to configure the communications controller, so that the host can properly receive multicast messages.

3.2.2.2 - Multicast connection release

As well as there is no explicit multicast connection establishment, there is no explicit release phase. A member may simply leave a group issuing a T-LEAVE.Request primitive that shifts his state machine directly to the "idle" state. No notification is sent to the server or any other member.

3.2.2.3 - Token-based data transfer management

CTMP defines two basic entity classes that may participate in group communications: the entities capable of sending and receiving data, called **producers**, and the **server**, that is a superset of a producer because it can also grant **tokens** and store token requests. Each group will always have only one server.

The token granting mechanism is straightforward. The server (whose address should be well-known throughout the network) sets up a group address (a network may have more than one group, each one associated to a different server). Any host that wishes to send data to a group must first request a token to the corresponding server issuing a T-TOKEN.Request primitive and a subsequent PASS PDU (this PDU is directly addressed to the server, not to the whole group). When the server receives a request, it places the requesting host's address inside a FIFO queue. If there is no space left in the queue, an ERROR PDU is returned. The server then verifies the availability of tokens. If the server holds at least one token, the first queue's address is removed, a TOKEN PDU is sent towards this address and the token timer (TTIMER) is started. This timer represents the maximum time interval a host can hold a token. Once it receives the token, the host is ready to start up its data transfer after notifying the user (T-TOKEN.Indication). When the host finishes its transmission, it must commence the devolution of the token (T-TOKEN.Response), triggering the immediate sending of a TOKEN PDU back to the server, wherein the TTIMER associated to that token is interrupted. As soon as a token is returned, the server verifies his request queue for any pending request. If the queue is not empty, the token is readily granted to the first address stored in it.

However, if TTIMER times out, a CNTL PDU requesting the immediate token's restitution is directed towards the host. A wait timer (WTIMER) is then started with a much shorter duration. At the destination host, the mandatory token devolution occurs at protocol level, that is, the user is only notified that his transmission permission is over and the token is returned. If the token does not arrive, WTIMER is successively restarted up to three times. After the third WTIMER time-out, the server considers that token is lost and increments a token counter.

Figure 4 illustrates a simple situation in which two hosts compete for a single token.

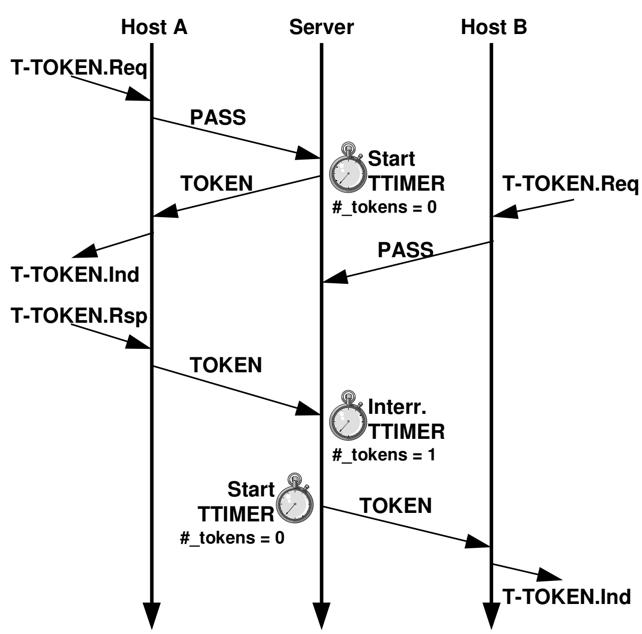

Figure 4: Token management example.

4 - CTMP implementation

High performance, flexibility and modularity are main objectives for the implementation architecture developed by UFRJ's Grupo de Teleinformática e Automação (GTA). Since the first system's version [3], it was clear that a careful implementation would be necessary, considering memory availabilities and operating system's overheads. Hence an efficient implementation architecture was defined in order to achieve significant performance results, with specialized memory management, task scheduling and timer management modules [11, 13, 14].

Fig. 5 shows CTMP's position inside GTA's implementation architecture. Four main modules stand out: **MAC**, **CTMP**, **User** (wherein test and performance analysis routines are implemented) and **Global**, comprising the Memory Manager, Task Scheduler and Timer Manager sub-modules. Besides, interfaces are implemented as FIFO queues, avoiding memory copies from one layer to another.

The Medium Access Control (MAC) layer used at the current system's Sun/Solaris version (there is also a PC/DOS version) implements basic access to the Sun's **le driver** interface. The le driver provides basic support for the LANCE hardware. The LANCE Ethernet driver [REF] is a multi-threaded, loadable, clonable, STREAMS hardware driver supporting the connectionless Data Link Provider Interface (DLPI), over a LANCE Ethernet controller.

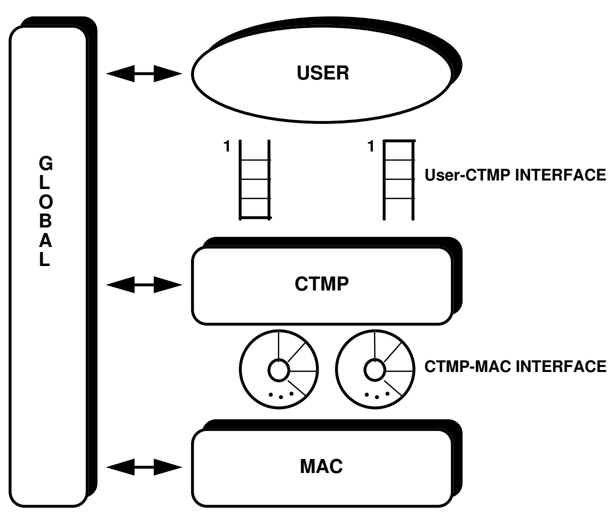

Figure 5: Implemented system.

5 - Performance analysis

The whole system is coded in C language and has two versions: a PC version, running under DOS and a Sun version running under Solaris 2.4. The target measurement environment consists of two Sun SPARCstations-20 connected to a 10 Mbit/s Ethernet LAN. Throughput measurements were performed by a specialized measurement **User** module (Fig. 5) for different CTMP configurations. In this paper, throughput regards to the total amount of user data transmitted within a certain time interval.

The ideal transmission time interval would be taken from the first message's transmission, at the sending side, to the last message's reception at the remote side. However, this kind of measurement would require clock synchronization and a distributed measurement system. The unavailability of such a system forced the creation of an alternative methodology: after all data has been transmitted, a final message tagged with an "end of transmission" mark is sent. When the remote side receives this message, it must respond immediately with an acknowledgment. The sender thus computes the time elapsed from the first message's transmission to the ack's reception. As it was not possible to discover the SPARCstation's timer granularity, due to lack of documentation, it was programmed for a 10 ms tick. Thus a minimum of 200 frames were transmitted to keep the measurement uncertainty at a reasonable level. It is important to emphasize that these measurements were taken under heavy network and CPU loads. Besides, the measurement application performed only memory to memory data transfers, avoiding any kind of disk access.

The present throughput results are actually lower bounds because they comprise user message's production/consumption, interfaces and memory allocation/deallocation processing, etc.

Fig. 6 shows the throughput (in Mbit/s) achieved by CTMP in different configurations as well as the le driver's throughput, for different message sizes ranging from 64 to 1024 bytes of user data. These measurements were performed over point-to-point connections. The following CTMP configurations were analyzed:

1) minimum overhead (no optional mechanisms enabled);

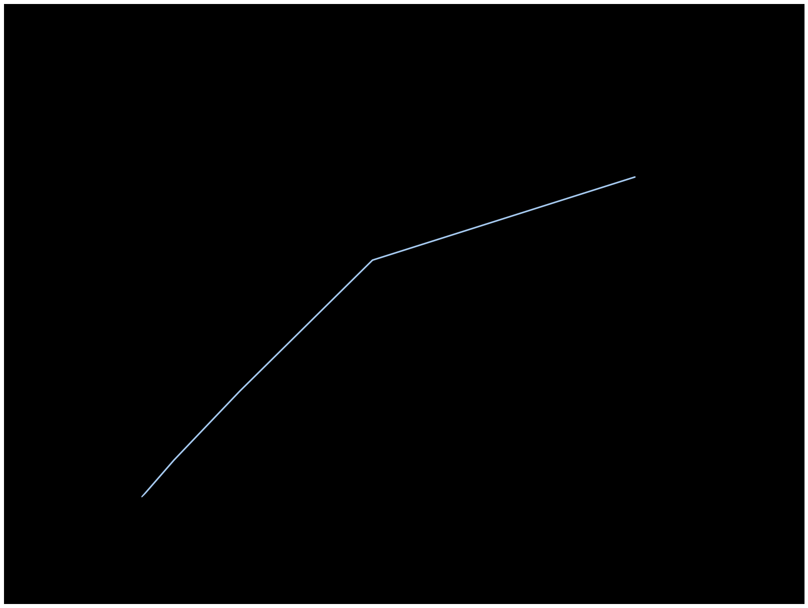

Figure 5: Throughput × message size.

2) checksum over DATA PDU's data segment enabled;
3) window flow control enabled;
4) checksum, window flow control and go-back-N error recovery enabled.

Table 2: Throughput results for 1024 byte messages.

Layer	Throughput (Mbit/s)
le driver	9.57
CTMP	8.65
CTMP w/ checksum	5.74
CTMP w/ window flow control	7.38
CTMP w/ error control	4.87

As it can be depicted from Fig. 6, throughput approximates maximum bandwidth for the le driver. Table 2 emphasizes the results obtained for 1024 byte messages.

Fig. 6 shows a significant decrease from the "pure" le driver's performance to CTMP's in configuration **1**. This is mainly due to the unavoidable time consuming data copy required by any communications controller: user data must be located at a contiguous memory region (specifically, CTMP-MAC's interface buffer). This result also includes the mandatory DATA's header checksum calculus overhead. However, its contribution to the overall performance degradation may be neglected in comparison to data copy.

CTMP's processing in configuration **3** presents another slight degradation in relation to configuration **1**. This is due to message storage inside the internal buffer, the acknowledgment CNTL PDU's processing and, mainly, the await for this ack: all transmissions remain blocked during this interval because the buffer is full. The buffer is released only when the ack is received, thus resuming the transmission.

CTMP's curve **2** in Fig. 6 shows the strong dependency between checksum computing and user data length. Checksum's costly processing causes an accentuated degradation when compared to configuration **1**.

As it would be expected, there is no big difference between curves **2** and **4** (the same results were obtained for go-back-N and selective retransmission options). The overhead imposed by window flow control/error control processing is negligible in comparison to checksum's overhead. Error control's processing consists of a retransmission's decision algorithm, based on information collected from ack CNTL PDUs.

As an evaluation of rate control's performance, Table 3 presents the effective results for different transmission rates. An average percentual deviation of approximately 2.5% is satisfactory if the 10 ms timing precision is considered.

Table 3: Rate control results.

Desired values (kbps)	Practical values (kbps)
64	64.3
128	129.6
256	256.8
512	518.6
1000	1047.3
2000	1966.0
3000	3002.4
4000	3703.7
5000	4743.6
6000	5849.0
7000	6899.5

6 - Conclusions

This paper presented a token-based configurable multicast protocol, CTMP. The increasing multimedia services' popularity arises the need for adaptive protocols capable of supporting different application's performance requirements. CTMP is a first step towards this scenario, providing the possibility of selection from a set of mechanisms:

- Optional checksum computing over user data segment.
- Two flow control mechanisms: transmission windows and rate control.
- Two error recovery mechanisms: go-back-N and selective retransmission.
- Optional bucket algorithm for multicast communications.

Therefore, a file transfer application that requires high reliability might enable checksum calculus, flow control and error control mechanisms; a datagram service would preferably turn off all optionals in order to take maximum advantage of the available bandwidth; on the other hand, rate control is best suited for real time applications in which rigid bandwidth and delay constraints are imposed.

The protocol presented a good performance when compared to the le driver's: approximately 11% of degradation for 1024 byte messages, resulting in an 8.65 Mbit/s throughput; this loss is due to the data copy required by communication's controllers. When the maximum processing overhead is considered (checksumming, flow control and error control), yet almost 5 Mbit/s are achieved for a point-to-point connection.

Concerning multicast communications, extensive tests proved the efficiency of the timer-controlled token-granting system. Due to equipment limitations, the multicast group was increased up to eight participants, each one in a different host. The implementation's client/server feature, with a centralized token server, allows continuous group size alterations without interfering with communication. Besides, the number of tokens may be arbitrarily increased (the current version implements two tokens). As a next step, a priority scheme may be associated with tokens.

Any protocol developed by GTA members follows an implementation architecture which main features are:

- asynchronous communications between layers, implemented as FIFO queues, resulting in high modularity and weak coupling;

- pointer-based standardized interface structures, carefully implemented to avoid memory copies between layers;
- an efficient memory management that separates physical memory into two distinct regions for transmission and reception, respectively. It also reduces memory fragmentation to a minimum;
- an efficient task scheduler that avoids unnecessary queue verifications;
- an "easy-to-use" timer system that efficiently manages the simultaneous creation of a great number of timers.

Finally, one of the next steps in CTMP's development is a further study and eventual implementation of Integrated Layer Processing (ILP) concepts [1, 8, 9]. Theoretically, an ILP protocol implementation reads once from main memory, keeps the read data within registers or cache memory and performs all the data manipulations for several protocol layers within a single loop. Processed data are then written to destination memory.

7 - References

[1] M. B. Abbott and L. L. Peterson, "Increasing Network Throughput by Integrating Protocol Layers", *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, October 1993.

[2] S. Armstrong, A. Freier and K. Marzullo, "Multicast Transport Protocol", RFC 1301, Network Information Center, SRI International, February 1992.

[3] 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", *X Congresso da Sociedade Brasileira de Computação*, Vitória, Brasil, July 1991, pp. 157-169.

[4] R. Braudes and S. Zabele, "Requirements for Multicast Protocols", RFC 1458, Network Information Center, SRI International, May 1993.

[5] D.R. Cheriton and C.L. Williamson, "VMTP as the Transport Layer for High-Performance Distributed Systems", *IEEE Communications Magazine*, vol. 27, no. 6, June 1989, pp. 37-44.

[6] G. Chesson, "Datakit Software Architecture", *Proceedings of the ICC*, 1979, pp. 20.2.1-20.2.5.

[7] D.D. Clark, M.L. Lambert and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol", Network Information Center RFC 998, SRI International, March 1987.

[8] D. D. Clark and D. L. Tennenhouse. "Architectural Considerations for a New Generation of Protocols", *Proceedings ACM SIGCOMM '90*, September 1990, Filadélfia, pp. 200-208.

[9] W. S. Dabbous, "High performance presentation and transport mechanisms for integrated communication subsystems", *Proceedings of the 4th International IFIP Workshop on Protocols for High Speed Networks*, Vacouver, August 1994.

[10] W. Doering et al, "A Survey of Light-weight Transport Protocol for High Speed Networks", IEEE Transactions on Communications, vol. COM-38, n. 11, November 1990, pp. 2057-2071.

[11] M.D. Nunes, C.V.N. Albuquerque and O.C.M.B. Duarte, "Performance Measurements in a Manufacturing Communication System", *ISCAS '94*, London, May 1994.

[12] M.D. Nunes and O.C.M.B. Duarte, "Análise de Mecanismos para Protocolos de Alto Desempenho", *VI Simpósio Brasileiro de Arquitetura de Computadores e Computação de Alto Desempenho*, Caxambu, August 1994.

[13] M.D. Nunes, "Protocolo Multidestinatário com Configuração de Facilidades e Gerenciamento de Acesso", *Master Thesis*, PEE-COPPE/UFRJ, April 1995.

[14] M.D. Nunes, C.V.N. Albuquerque and O.C.M.B. Duarte, "An Efficient Implementation Architecture for Layered Communication Systems", *38th Midwest Symposium on Circuits and Systems*, Rio de Janeiro, Brasil, August 1995.

[15] M.D. Nunes and O.C.M.B. Duarte, "Proposta e Implementação de um Protocolo Multidestinatário Configurável", *VII Simpósio Brasileiro de Arquitetura de Computadores - Processamento de Alto Desempenho*, Canela, Brasil, July 1995, pp. 229-242.

[16] M. D. Nunes and O. C. M. B. Duarte, "MCF: Um Protocolo Multidestinatário Configurável com Gerenciamento de Acesso por Fichas de Permissão", *XIV Simpósio Brasileiro de Redes de Computadores*, Fortaleza, Brasil, May 1996.

[17] Protocol Engines Inc., “Xpress Transfer Protocol Definition”, Revision 3.6, January 1992.

[18] H. Santoso and S. Fdida, “Transport Layer Multicast: XTP Bucket Error Control and Its Enhancement”, *4th. IFIP Conference on High Performance Networking*, Liège, Belgium, December 1992.

[19] Sun Microsystems Inc., “SunOS Reference Manual - Sections 6 and 7”, 1994.

[20] C. Topolcic, “Experimental Internet Stream Protocol”, RFC 1190, Network Information Center, SRI International, October 1990.

[21] R.W. Watson, “The Delta-t Transport Protocol”, *Proceedings of the 14th. Conference on Local Computer Networks*, Minneapolis, Minnesota, October 1989, pp. 415-420.

[22] L. Zhang et al., “A New Resource ReSerVation Protocol”, IEEE Network Magazine, vol.6, n.5, September 1993, pp. 8-18.