Performance Study of the Xpress Transfer Protocol

Doina Taler

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfilment of the Requirements

for the Degree of Master of Computer Science at

Montreal, Quebec, Canada

September 1992

ISBN 0-315-80957-4

Name DONA TALER

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

COMPUTER SCIENCE — SUBJECT TERM

0984 — SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences

Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences

Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the major report prepared

By: **Ms. Doina Taler**

Entitled: **Performance Study of the Xpress Transfer Protocol**

and submitted in partial fulfilment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Examiner

J. W. Atwood _____ Supervisor

Approved _____ 11/9/92

Chair of Department or Graduate Programme Director

Sept 11 1992 _____

Dean of Faculty

ABSTRACT

Performance Study of the Xpress Transfer Protocol

Doina Taler

This major report presents a study of the performance of the Kernel Reference Model (KRM) version 1.7, implementation of the Xpress Transfer Protocol (XTP) version 3.6.

XTP is a "new generation" protocol, combining the network and transport layers. It is designed to respond to present and future computer applications, and to take advantage of the hardware possibilities.

The performance measurements include unicast throughput and delay in an Ethernet network, in two different hardware configurations, one based on Sun-3 workstations, and the other one on Sun-4 workstations.

The performance of the KRM is compared with that of other protocols such as the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), all in the same environment and under the same conditions.

To the memory of my father

ACKNOWLEDGEMENTS

I wish to express my appreciation to Dr. J.W. Atwood for his professional assistance and guidance during the course of this project.

I would also like to acknowledge the support and help of the Analysts in the Department of Computer Science.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

As the communication world devoted serious efforts for finding a communication medium that provides a faster, cleaner and more reliable bandwidth, significant progress has been made with the fiber optics.

Fiber Distributed Data Interfaces (FDDI) support data transmission speeds in the order of magnitude of Gigabits per second, as opposed to copper cable networks with performance of 10 Mbps - Ethernet or 4 and 16 Mbps - Token Ring. In addition, the bit error rate and the security of fiber optics media, also outperform those of copper cable.

In the evolving world of computers, distributed, real-time, transaction-based and multi-media systems are becoming more prevalent.

However, the "old generation" of communication protocols do not take full advantage of the hardware capabilities and do not provide all the functionality and flexibility required by the current and future computer applications. Among these protocols are the ISO Transport Protocol Class 4 (TP4) and the Transmission Control Protocol (TCP).

The Xpress Transfer Protocol (XTP) is a "new generation" protocol. It had as a general design objective to be an effective protocol for the new generation and future generations of hardware and computer applications. It was also designed for both hardware (as a VLSI chip set) and software implementations.

This report presents the results of performance experiments performed on the Kernel Reference Model (KRM1.7) implementation of the XTP and on other protocols

such as the Transmission Control Protocol (TCP), the User Datagram Protocol (UDP), and the RAW Network Driver.

The report consists of six chapters.

Chapter 2 describes the XTP protocol, its packet structure, and its functionality divided into Transport and Network functions. It also includes the Upper Layer Interface to XTP.

Chapter 3 describes the KRM, its installation, its Application Programming Interfaces (API): the Socket Interface and the Character Driver Interface.

Chapter 4 describes the performance tests, the hardware and software configuration, the performance metrics used, the suite of tests, and the programs.

Chapter 5 analyses the results of the performance measurements, for each of the two configurations, and for each protocol in study, and compares the results.

Chapter 6 summarizes the results of the performance measurements and comparisons.

Chapter 2

XTP Description

2.1 Overview

The Xpress Transfer Protocol (XTP) is being developed by Protocol Engines Inc. (PEI), and its author is Dr. Greg Chesson.

In the Open Systems Interconnection (OSI) reference model language, XTP provides the functionality of the layers four (transport) and three (network). The latest version (XTP 3.6) of the protocol is the basis for the ANSI proposal to the International Organization for Standardization (ISO) for a High Speed Transport Protocol (HSTP) for the next generation OSI.

Protocols that combine the network and the transport layers are called transfer protocols. One advantage of such protocols is the elimination of the overhead imposed by the processing involved in the communication between the two combined layers.

The XTP protocol functionality categorized in network, transport, and other features, with accent on the way it meets the requirements of modern systems, is described in a subchapter below.

2.2 XTP Definitions

The following definitions are required for a better understanding of the XTP description.

A Context is the collection of state information kept within one end system (including sending and receiving control information), representing one instance of an active communication between two (or more) XTP endpoints.

An Association is the aggregate of a pair of active contexts and the data streams between them.

2.3 XTP Packet

The messages passed by the upper layer protocol (the user of the Transport Protocol) to XTP are called Transport Service Data Units (TSDU). The TSDUs are processed by XTP, and one or more Transport Protocol Data Units (TPDU) are built and passed to the lower layer protocol for transmission. The lower layer protocol is most likely a data-link protocol, but it could be a Media Access Control (MAC) or Network protocol. The XTP TPDUs are also called XTP packets throughout this document.

The XTP packet is encapsulated within the MAC layer frame. Although the XTP packet can directly follow the MAC header (802.3, 802.5, FDDI), in some cases, additional Layer 2 or Layer 3 protocol headers may be required (ex. LLC, Internet IP).

The XTP packet consists of a 40-byte header, a middle segment and a 4-byte trailer. The middle segment can be an Information Segment or a Control Segment, and its size is a multiple of 4 bytes. Null data packets can be sent ("zero length data segments"), for error control purposes. The XTP packet structure is represented in Figures 1a and 1b.

All these protocol fields are sent and received in "big-endian" order. Therefore, in little-endian systems, the XTP implementation has to make the appropriate conversion after reception and before transmission.

There are two types of XTP packets:

1) control packets (the middle segment is a Control Segment),

containing protocol state information for the sending end. The following control packets are defined:

- CNTL - state exchange
- RCNTL - intermediate system control

2) data packets (the middle segment is an Information Segment), containing user level data and transport layer messages. The following types of data packets are defined:

- FIRST - initial packet at association setup (may or may not carry user data)
- DATA - user data
- PATH - reconnection packet, locate switches
- DIAG - peer to peer diagnostic packet
- MAINT - network maintenance information
- MGMT - network management information
- ROUTE - route management

The maximum length of a packet is defined by the network's Medium Access Control (MAC) layer. The user data (TSDUs), may fill several XTP packets (TPDUs), as there is no restriction on its size other than the available memory limit and the 32 bit descriptors.

The minimum length of an XTP packet is 48 bytes. Such a packet consists of the fixed size header of 40 bytes, a minimum 4 byte information segment and the fixed size trailer of 4 bytes.

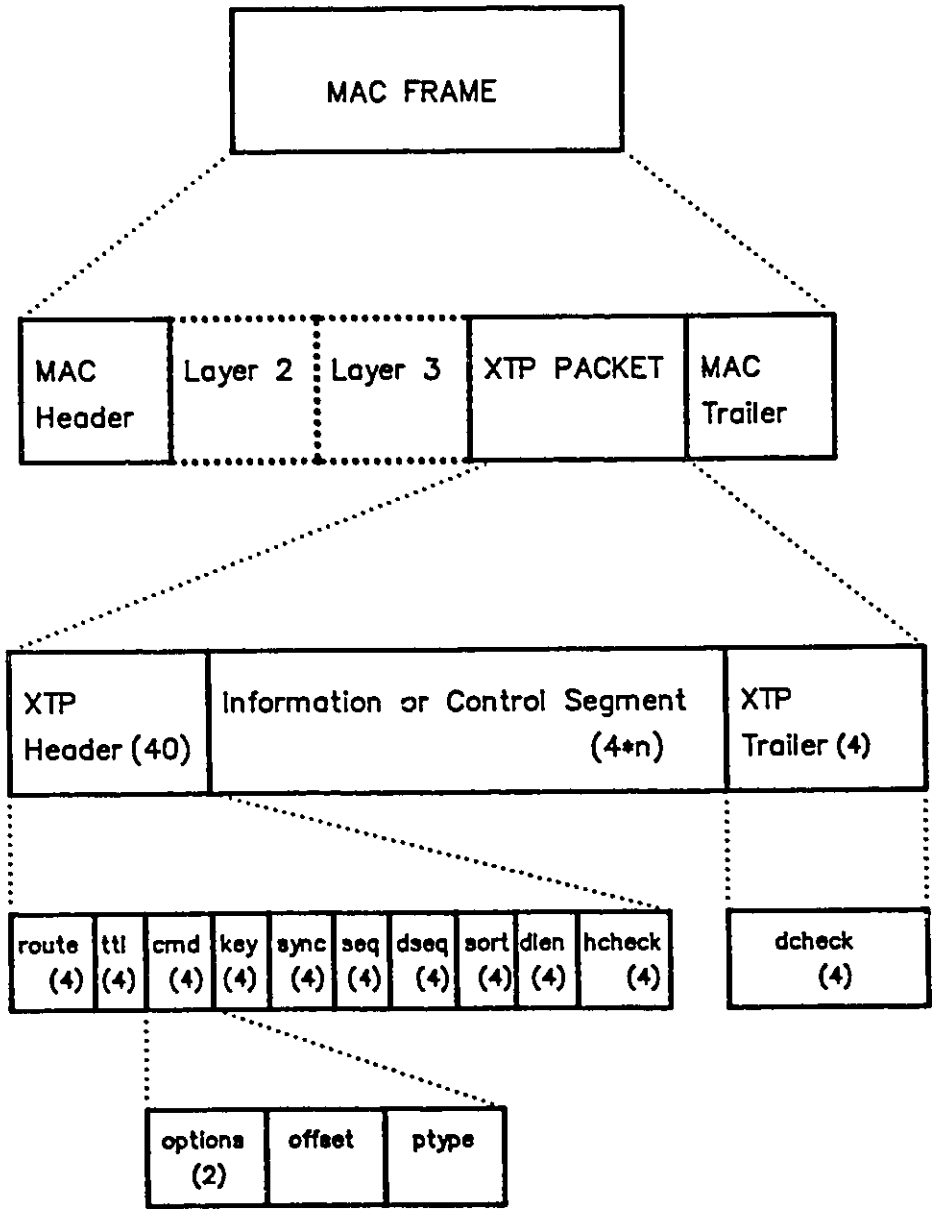

FIGURE 1a – XTP PACKET STRUCTURE

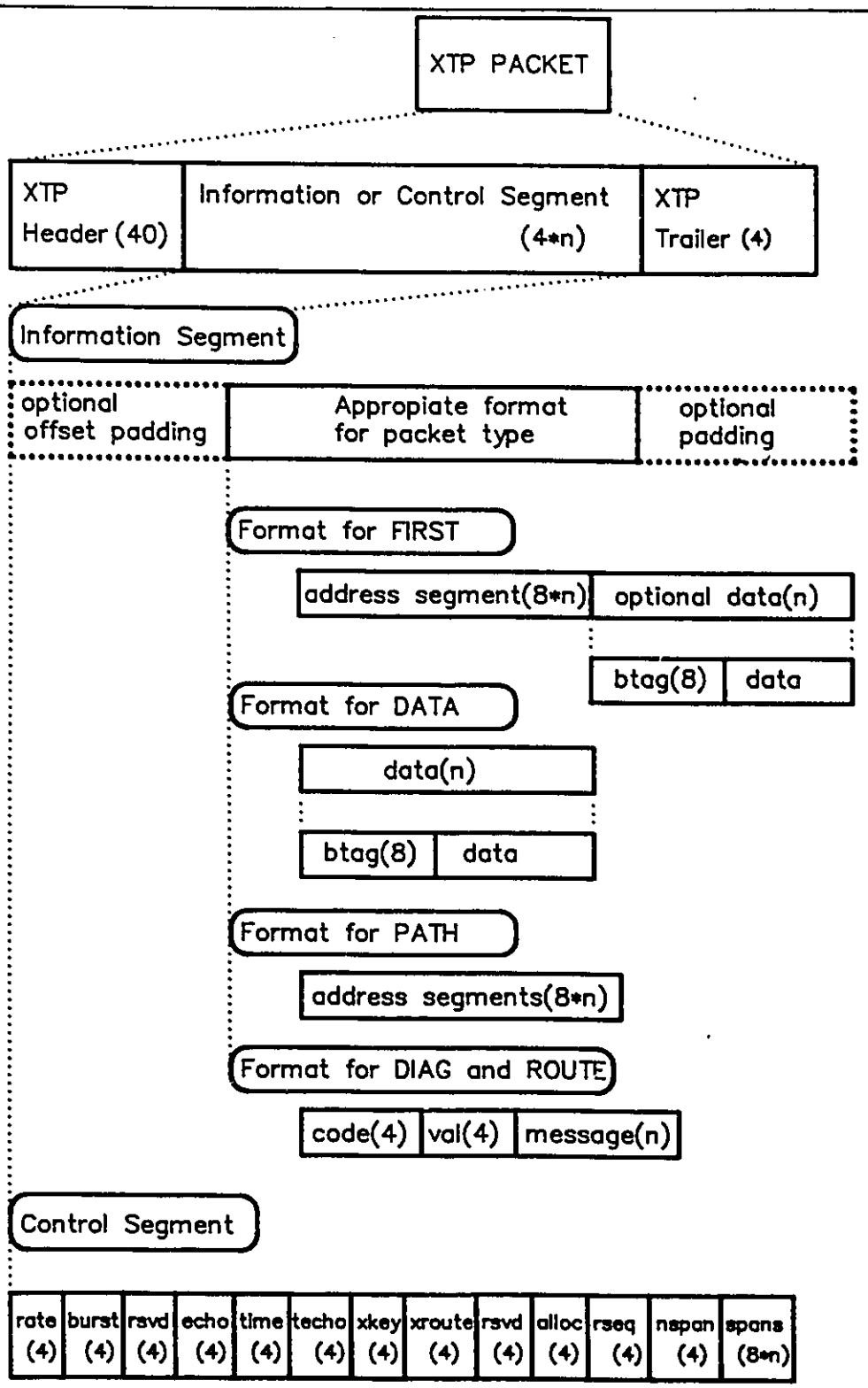

FIGURE 1b – XTP PACKET STRUCTURE

2.4 Transport Functions

As described by the OSI reference model, the transport layer should provide reliable, transparent transfer of data between end points. It also provides end-to-end error recovery and flow control.

XTP provides the following functions:

- establish transport connections between transport users,
- terminate transport connections,
- transport connection identification,
- data flow including: control packet processing, flow control, rate control and error control.
- transport of User Data (TSDUs),
- segmenting and reassembling (from TSDU to TPDUs and reverse),
- transmission of TPDUs

Some of these functions are described in more detail below.

2.4.1 Transport Connections

Association Setup

The association set up by XTP is a full-duplex bidirectional virtual circuit between two end systems. The collection of protocol state at each end is kept in each context. At connection setup, an association between two contexts is created. This is done by transmitting a FIRST packet from one end to the other.

Both implicit and explicit transport connection setups are provided, between two transport users.

When an implicit connection is set up, the sender may transmit data

immediately following the FIRST packet without waiting for any response from the receiver. The receiving system can also begin transmitting its outgoing data as soon as it has an active context. The advantage of this type of connection setup, is that it eliminates the latency of a preliminary end-to-end message exchange, and is therefore faster than the explicit connection setup. This approach constitutes a building block in fulfilling the requirement of distributed systems to have rapid request/response communication facilities.

When an explicit connection is set up, the sender will not transmit data until it receives a positive acknowledgement.

Association Termination

Graceful, abbreviated or abortive association shutdowns are provided.

In the association termination procedure, the two involved contexts are released and the two data streams are closed with a handshake.

At least one of the following two parameters are used in the closing procedure: a retry count which specifies the number of CTRL packets retries before declaring the association closed; and a timer (CTIMER) specifying the amount of time to wait while retrying CTRL packets before declaring the association closed.

The closing procedure can be initiated by any of the two participating ends.

Association Identification

The association identification is used at both reception and transmission, for updating the appropriate context, to deliver data to the appropriate user, and to send data to the appropriate destination.

The triplet <source MAC address, key, route> is used for this purpose. It uniquely identifies an XTP context in an inter-network environment. The MAC value uniquely identifies nodes on the inter-network. The key uniquely identifies a context in a node. The route selects an exit port for routing the packet in an inter-network.

The receiver performs a lookup operation on the triplet, but for the transmitter side, after the initial setup, a low overhead key table lookup is enough. Mechanisms are defined to exchange keys, to permit low overhead lookup by both contexts.

The key exchange works only between a pair of contexts and therefore it cannot be used in multicast mode.

2.4.2 Data Flow

Data flow includes:

- control packet processing,
- flow control,
- rate control,
- error control.

All these procedures are based on sequencing. XTP uses 32-bit byte sequence numbers (unsigned integers). All the data streams (user data only) sent and received are arbitrary length strings of sequenced bytes, where each byte is represented by such a sequence number.

Control Packets

CTRL packets exchange acknowledgement and state information between end systems, for the purpose of a reliable communication.

Error Control

Sequence error detection and correction is performed through positive or negative acknowledgement and selective retransmissions. The acknowledgement is made in terms of sequence numbers. Alternatively, go-back-n retransmissions may be used.

The selective acknowledgement and retransmissions represent a serious overhead. Therefore, in reliable networks, it may be desirable to use go-back-n retransmission instead, for better performance.

Packet error detection is through checksum calculations. XTP uses two checksums:

- header checksum (contained in the header), which cannot be disabled,
- middle segment checksum (contained in the trailer), which can be disabled. This checksum is calculated over the XTP information or control segment and includes any padding bytes.

Error control can be disabled. An aggressive error control mode (quicker acting) is also provided.

Flow Control

Output Flow Control is based on 32-bit sliding window of sequence numbers.

A do-not-exceed sequence number (alloc) is defined that limits the amount of data a sender may transmit. Initially alloc is set as a default parameter, and is updated later with the value sent by the receiver, which is responsible for adjusting the sender's window. This alloc value also defines the internal buffer space of the

receiving XTP machine. An exception is the so-called reservation mode in which the value represents the higher-layer application buffer space.

Flow control can be disabled.

Rate Control

Rate Control consists of output packet rate (speed) regulation.

An end system or an intermediate system can specify the maximum bandwidth and burst size it will accept on an association or on a group of associations related by the same route field.

Two parameters are important for the rate control: the rate, specifying the maximum data rate per second; and the burst, specifying the number of bytes in each "burst" of data. A refresh timer is also used.

Optionally, the rate and burst may not be constrained, which means the sender can transmit at will.

2.5 Network Functions

As described by the OSI reference model, the network layer provides upper layers with independence from the transmission and switching technologies used to connect systems. It is also responsible for establishing, maintaining, and terminating associations.

The XTP network functions are:

- addressing,
- routing.

The XTP network features allow the use of the protocol in environments that

include hardware switches as well as Local Area Network (LAN) and router technology.

Addressing

The XTP addressing is compatible with standard addressing schemes and provides for flexible address support.

The addressing mechanism enables a sender to locate a destination machine and select a destination service and establishes a return path for data and control packets. It is defined through the address segment found in the FIRST and PATH packet types.

The address segment contains destination and sender addressing information. As XTP does not have its own address format, the address format may specify one of a number of established routing protocols. Among those are: Internet Protocol, ISO Standard, XNS, IEEE 802-style Source Route, etc. Direct Addressing is also provided. When it is used, XTP does not define the address field, leaving provision for the systems having a known, fixed communication topology, or other applications where a Permanent Virtual Circuit facility is required.

Routing

Path is the basic concept used by XTP for routing.

The path is a concatenation of zero or more interconnect devices. Each interconnect device forwards packets to the next until the last device forwards to the destination end system. The path can also be seen as a concatenation of route values, for the interconnect devices.

The setup of the path is done when an association is created.

When intermediate nodes are to be used for establishing an association, the intermediate nodes use a technique named cut-through switching. In this method, the FIRST packet leaves a trail of references (a path) behind, as it passes between intermediate nodes. Each intermediate node forwards the packet, according to its address segment. The forwarding procedure is dictated by the address format code in the address segment. At this time a mapping between the source MAC address, route, and the exit port is also created in the intermediate node, which allows subsequent packets to take the same path.

The route value in the XTP packets specifies a route entry in the host, where the round trip time and rate control values are maintained by all the contexts using that route.

The route exchange allows receivers to assign the route value used by the adjacent sender. After the route exchange, the traffic in both directions implies very low overhead lookups in intermediate nodes.

The route exchange works only between a pair of contexts and therefore cannot be used in multicast mode.

When a new association is established between two endpoints that already have an active association, the existing route can be shared, reducing the resource usage.

The release of route assignments is performed through orderly handshakes.

2.6 Other Features

The following features are also available:

- message priority/scheduling
- support for encapsulation and convergence protocols
- security
- traditional stream capabilities
- padding
- timers
- reliable multicast mechanism

Message Priority/Scheduling

This feature is a requirement for Real Time systems that need to control the scheduling of packet processing among multiple contexts.

This feature is optional.

The priority ordering is controlled and applied at both input and output.

The higher priority contexts are served before those with lower priority. The contexts with this option disabled are served last. If a lower priority context is being serviced while a higher priority one become active, XTP switches to the higher priority context, at the next convenient time.

A maximum time limit for preemption between contexts is implementation dependent.

Support for Encapsulation and Convergence Protocols

The first eight bytes of a data segment may be marked or "tagged" for special use by the higher-layer application.

These are uninterpreted user data bytes. Therefore the application has a means

of mixing control information with data, at the same time avoiding another level of framing.

Security

The complete XTP packet could be encrypted and encapsulated within a "security frame". The design of such a frame is not in the scope of XTP.

This feature could affect the real-time performance, unless specialized hardware is used.

Padding

For the purpose of aligning the beginning of the middle segment, making the middle segment a multiple of 4 bytes, aligning the trailer on a multiple of four byte boundary, and meeting the minimum packet requirement for certain physical layers, the middle segment may contain padding at the beginning as well as at the end.

The padding bytes are excluded from sequencing, but are included in the data checksum.

The minimum size XTP packet is 48 bytes. For media with a larger minimum packet size, padding can be added at the end of the XTP packet, after the trailer.

Timers

XTP uses five timers. They are connection timers (CTIMER and CTIMEOUT), the round trip timer (rtt), the wait timer (WTIMER) and the rate control timer (RTIMER).

The round trip timer is continuously calculated using data in the control

packets exchanged between active contexts.

The CTIMER is a long-duration timer, used to generate keepalive messages.

The CTIMEOUT is a short-duration timer, used to limit the time XTP retries a message exchange.

The two connection timers measure the protocol inactivity, and are therefore used in the XTP recovery procedures for network failures.

The wait timer represents the time an XTP sender waits for a response to a status request packet, before retransmitting the packet.

The rate control timer is used to control the updating of rate control variables.

Reliable Multicast Mechanism

Multicasting designates the simultaneous communication between the entities of a specific subset of entities within a domain (e.g., network). This subset constitutes a multicast group, identified by a multicast address. A multicast conversation is composed of messages simultaneously addressed by one entity (the sender) to the others (the receivers).

Multicast capabilities respond to certain requirements in distributed systems and multimedia systems.

Although XTP multicasting is destined for media that provide broadcast or multicast facilities, the multicast functionality can be extended to non-multicast media, by using a link layer that defines output replication to a group of nodes on the network.

The data delivery in multicast mode is reliable or best effort, meaning that if a receiver cannot keep up a multicast conversation, it will drop out.

In general, in XTP multicasting mode the same protocol procedures are used, as in unicast mode. The major differences consist in addressing and the processing involved in the generation of responses to the control-request packets.

In this mode additional processing and delay must be inserted, between the reception of a control packet and its processing, to allow all the receivers participating in the conversation to respond. For this purpose a so-called Bucket Algorithm is used.

Multicasting procedures could add overhead to the protocol. In order to make this mode more efficient, two techniques are used to overcome the additional processing: damping and slotting.

Damping eliminates duplication of control packets, sent by the receivers in a conversation. The receivers monitor the packets on the network and will not send such a control packet if a similar one has already been sent by another node. The trade off is the overhead of the receivers monitoring all the control messages. The advantage is substantial when the number of receivers is high.

Slotting consists of imposing a random delay before responding to control packets, in order to distribute them in time, and decrease network load.

2.7 Upper Layer Interface to XTP

The Upper Layer Protocol is the user of the XTP protocol, also called the application user. An XTP implementation should provide means for an application to specify all the following options:

Service Type Specification

The user can specify the kind of traffic expected on the association,

representing the following available service types. This information is passed in the FIRST packet to the destination.

- connection - a long lived association
- transaction - a single arbitrary-length message is sent and a single arbitrary-length reply is received
- unacknowledged datagram - connectionless service, delivery confirmation is not provided,
- acknowledged datagram - connectionless service, delivery confirmation is provided,
- isochronous stream
- bulk data - large volumes of data

Protocol Parameters Specification

Default values may be used for all the following parameters, which can be changed by the user:

- address - destination address segment
- maxdata - maximum output data segment
- rtt - the initial assumption for round-trip time
- burst - sender's initial maximum packet burst per context
- rate - sender's initial maximum data rate per second
- oqmax - maximum size of output queue per context
- iqmax - maximum size of input queue per context
- ralloc - allocation permitted by receiver
- talloc - allocation assumed by sender

- CTIMER - connection activity timer value
- CTIMEOUT - connection timeout value
- WTIMER - initial value for acknowledgement wait timer
- retrycount - number of unacknowledged status request packets to cause a connection abort

Type of Error Control Specification

For better performance, it is desirable to be able to specify the go-back-n retransmission technique usage instead of the selective acknowlegement and retransmission. This option should be used for reliable networks.

Supervise FIRST Packet

An application might need to supervise the FIRST packet, in order to accept or reject the FIRST packet, according to its own criteria.

Enable Sorting

Sorting is used in XTP for designating priority ordering among multiple contexts. In this case a parameter is the sort (priority) value for the context.

The default is to disable sorting.

Enable Reservation Mode

In this mode the sender indicates to the receiver that the alloc value provided by the receiver in its control segment must represent the actual client buffer space available, and not the XTP internal buffer space. The value of alloc is also passed as

parameter. This mode should be used for bulk transfers to avoid overflowing the XTP buffers.

Enable Aggressive Error Control

This option enables aggressive Error Control at the receiving side. That means that, when an out-of-order packet is received, the receiver will immediately return a CTRL packet to the sender to indicate the error.

Tagging

This option allows the user to specify an 8 byte tag for its own use, separate from the user's data. The tag is written in the first 8 bytes of the data segment and it is not interpreted by XTP.

Disable Rate Control

Disable Data Checksum

Disable Flow Control

Disable Error Control

Enable Multicasting

Enable/Disable Damping

This option is available only if multicasting is enabled.

Enable/Disable Slotting

This option is available only if multicasting is enabled.

Chapter 3

KRM Description

3.1 Overview

The Kernel Reference Model (KRM), version 1.7, is a software implementation of the Xpress Transfer Protocol (XTP), version 3.6, within the UNIX kernel, by Protocol Engines Inc.

The UNIX systems supported are SunOS 4.0.3 or 4.1, on Sun-3 or Sun-4, and several versions of IRIX on several Silicon Graphics Inc. machines.

The current version of the KRM did not have performance optimization for specific machines or operating systems as a general design objective

This version of the KRM implements all the XTP 3.6 features except basic routing, which is in the development phase. Route Packets and Route Exchange are already implemented.

As this project has been developed on SunOS 4.1.1, all the next chapters refer only to this operating system.

3.2 Installation

The KRM distribution tape contains an installation script which has only to be updated for a specific environment (SunOS version, Sun Hardware type, etc.).

The KRM installation involves reconfiguration and generation of a new kernel, and it has to be run by the super-user.

The result of running the installation script is a new version of the kernel (vmunix). The final step of the installation is to reboot the machine with the new

vmunix in the root directory.

The default KRM configuration includes debugging and tracing code. To be able to compare the performance of XTP with other protocols, the debugging and tracing options have been disabled in the KRM. This has been done by modifying the file xtp_param.h and rebuilding the kernel.

3.3 Utilities

The KRM is delivered together with a set of utilities that test its operation. These are:

- modified versions of the remote BSD tools (rlogin, rsh, rcp), which use XTP sockets instead of TCP sockets.
- KRM debugging tools for configuring, formatting and displaying debug information.
- tests for each of the transport layer interfaces:
    - the socket interface
    - the character driver interface
    - the command block driver interface (not provided for SunOS)

3.4 The Application Programming Interfaces

The KRM provides three UNIX Application Programming Interfaces (API). They are:

- the socket interface
- the character driver interface

- the command block driver interface (not available for SunOS)

3.4.1 The Socket Interface

The socket interface has been designed according to the Berkeley socket model. As a result, the existing software written for the TCP/IP protocol using Berkeley sockets, needs only recompilation in order to be used with XTP.

All the Berkeley socket interface calls are available for XTP:

- socket
- bind
- connect
- listen
- accept
- send
- recv
- close

Some features are missing. For example, the KRM does not provide an out-of-band data mechanism such as the one provided for TCP.

The protocol-related functions supported by the KRM are described in a chapter below. All of those are supported for this API, except tagging.

3.4.2 The Character Driver Interface

This API provides the following standard primitives:

- open
- close

- read
- write
- select
- ioctl

As for other raw drivers, the read and write buffers are locked down into memory, so that they cannot be touched by the paging system. After the data transfer is completed, the buffers are unlocked and control is returned to the user. That means that the both the write and read operations will block until the data have been fully delivered and acknowledged. A different mechanism is used for short data buffers, which are copied.

For the detailed descriptions of the system calls refer to the KRM User's Guide.

The protocol-related functions supported by the KRM are described in the next chapter. All of those are supported for this API.

3.5 Protocol-Related Options

The following protocol related options supported by the KRM, are available to be specified by the users:

Service Type Specification

Only the connection type service is supported.

Protocol Parameters Specification

The following parameters can be changed by the user:

- rbufsize - receive buffer size,

- wbufsize - transmit buffer size,
- address - destination address segment
- rtt - the initial assumption for round-trip time
- burst - sender's initial maximum packet burst per context
- ouput_rate - output data rate per second
- input_rate - input data rate per second
- ralloc - allocation permitted by receiver
- CTIMER - connection activity timer value
- CTIMEOUT - connection timeout value
- WTIMER - initial value for acknowledgement wait timer
- retrycount - number of unacknowledged status request packets to cause a connection abort

Enable Sorting

Enable Reservation Mode

Enable Aggressive Error Control

Tagging (only for the character driver interface)

Disable Rate Control

Disable Data Checksum

Disable Flow Control

Disable Error Control

Enable Multicasting

Enable/Disable Damping

Enable/Disable Slotting

3.6 Interaction with Other Standard Protocols

XTP has been assigned a permanent Ethertype, by Xerox, as 817d(hex).

Some permanent XNS socket numbers have been obtained for XTP, but are not in use yet.

A group of multicast addresses has been assigned to XTP, by IEEE.

Berkeley address family and protocol family have been officially allocated for XTP (both 19 (decimal)).

An IP "protocol type" value of 36 (decimal) has been officially assigned to XTP.

3.7 Implementation

The KRM is implemented within the UNIX kernel.

The KRM has three interfaces with the UNIX kernel:

- Transport Layer Access Points
- Kernel Supporting Routines
- Link Layer Access Points (Network Driver)

These interfaces are pictured in Figure 2.

The Transport Layer Access Points represent the interface between the KRM and the application users. There are two modalities for the user to access KRM services: the Command Interface and the Event Signalling Interface.

The Command Interface is through command blocks. The entry points for this interface are described in detail in the KRM User's Guide.

The Event Signalling Interface is used by the KRM to notify user processes when specific events take place. The entry points for this interface are described in detail in the KRM User's Guide.

The Kernel Supporting Routines include mbuf memory allocation routines, socket support functions, routing functions, and standard UNIX services.

The Link Layer Interface is done through a network driver. For the Socket Interface API, this interface is through the Berkeley Socket Interface. For the character driver API, this interface is a direct interface to the network driver.

In order to use the Berkeley socket interface, the functions sosend and soreceive of this interface, have been modified to recognize XTP.

The direct communication with the network driver is strictly in terms of mbuf chains representing packets that are handed back and forth. An mbuf chain is simply a linked list of buffers called mbufs, containing the components of the XTP TPDU (header, middle segment, and trailer).

Most of the Link Layer Service Access Points (LSAP) table structure consists of an ifnet structure borrowed from Berkeley code, which has the right information for a general purpose link layer interface. In addition, two entries, one for input and one for output-done, have been added, in the protocol switch table, to specify "up" calls from the link layer to XTP.

Since the Berkeley code does not make use of "output-done", KRM uses the following technique: it implements a timeout for checking the completion of the output.

At input, when an mbuf chain is given by the network driver, the KRM interrupt function is called by the Berkeley interrupt function. The KRM interrupt function extracts the MAC header, and the XTP packet components and calls the KRM receive processing routine.

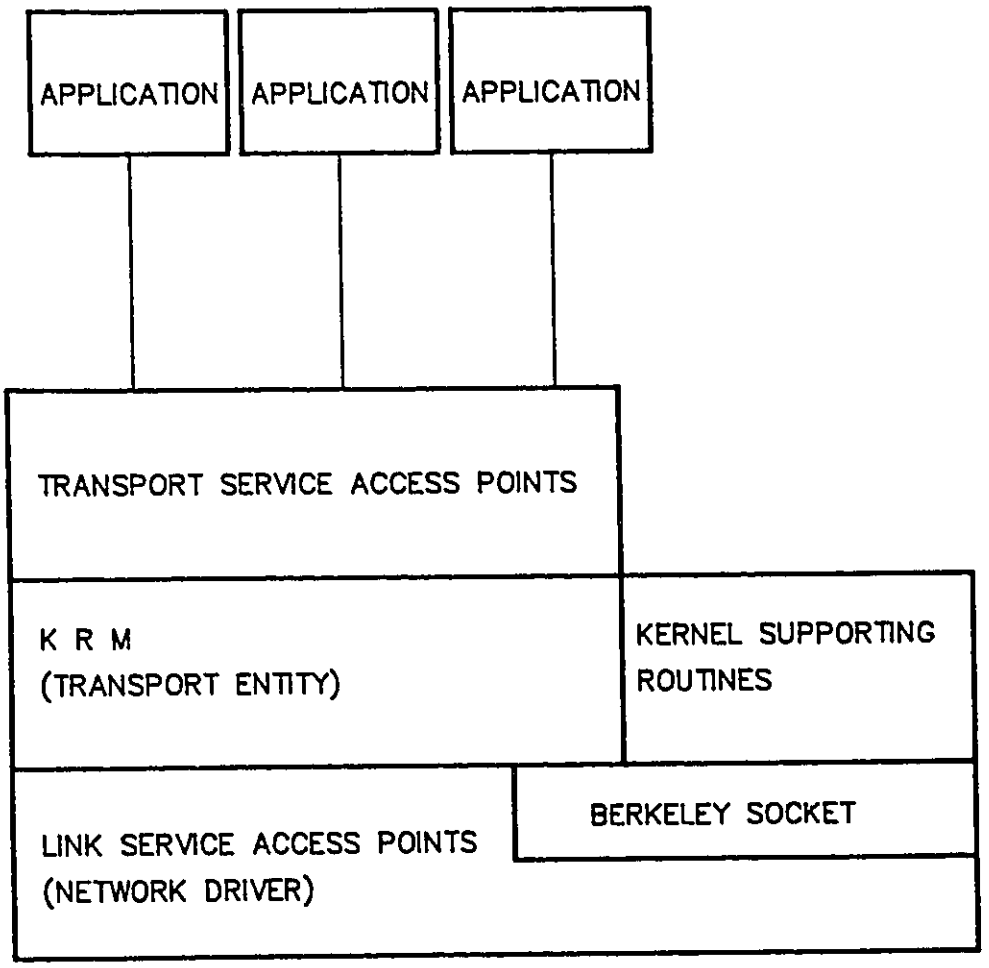

FIGURE 2 KRM – UNIX KERNEL INTERFACES

Currently the KRM interrupt routine does not take advantage of specific configuration representation of input data. It is expected that in future versions this approach will be modified to achieve better performance.

Chapter 4

Descriptions of Performance Tests

4.1 Hardware Configuration

Two different hardware configurations have been used for the performance measurements.

The first configuration is based on two Sun-3/50 workstations. The Sun-3/50 workstations have a MC68020 processor running at 15 MHz and 4 MBytes of main memory. They have an integer performance of 1.5 Million Instructions Per Second (MIPS).

One standalone (named xtp1) and one diskless (named xtp2) systems were used, xtp2 being the NFS client of xtp1.

The second configuration is based on two Sun-4 workstations. The Sun-4 workstations use the Sun's own implementation of Reduced Instruction Set Computing (RISC) technology, the 32-bit Scalable Processor ARChitecture (SPARC).

One workstation (named xtp3) is a 4/50 IPX system, running at 40 MHz, with 16 MBytes of main memory and an integer performance of 28.5 MIPS.

The other workstation (named xtp4) is a 4/40 IPC system, running at 25 MHz, with 12 MBytes of main memory and an integer performance of 17.8 MIPS, being the diskless NFS client of xtp3.

In both configurations, the network interfaces provided access to a 10 Megabits per second (Mbps) Ethernet through a controller using the ADM Local Area Network Controller for Ethernet (LANCE) Am7990 chip. Thin Ethernet cables (built-in thin Ethernet transceivers for sun-3 and independent transceivers for sun-4) were used to

connect to one of the Ethernet subnetworks of the University of Concordia network, as shown in Figure 3. For the second configuration, using sun-4, the four machines were isolated from the rest of the network.

4.2 Software Configuration

The operating system used was the SunOS version 4.4.1, based on Berkeley 4.2BSD.

The TCP/IP and UDP protocols implementations within the SunOS kernel are also based on the Berkeley version.

Beginning with version 4.1, SunOS includes the Network Interface Tap (NIT), a facility that provides link level network access.

The implementation of the XTP protocol version 3.6 within the kernel, the KRM 1.7, was installed with the debug off and traces off options.

4.3 Performance Metrics

The performance measurements included throughput and delay. The definitions of the metrics used are given below.

The **throughput** is the user data bits reliably transferred per second. The following formula has been used:

(BSIZE * NBUF) / ((1024 * 1024) * (TIMES)),

where,

- BSIZE is the user data buffer size in bits,
- NBUF is the number of user data buffers,
- TIMES is the elapsed time to transfer (BSIZE * NBUF) bits, measured in

seconds,

The results are in Megabits per second (Mbps).

The **delay** measured is the delay to send and receive back the same message. The following formula has been used:

TIMEMS / NBUF,

where,

- BSIZE is the user data buffer size in bits,
- NBUF is the number of user data buffers,
- TIMEMS is the elapsed time to send and receive back (BSIZE * NBUF) bits, measured in milliseconds (ms),

The results are in milliseconds.

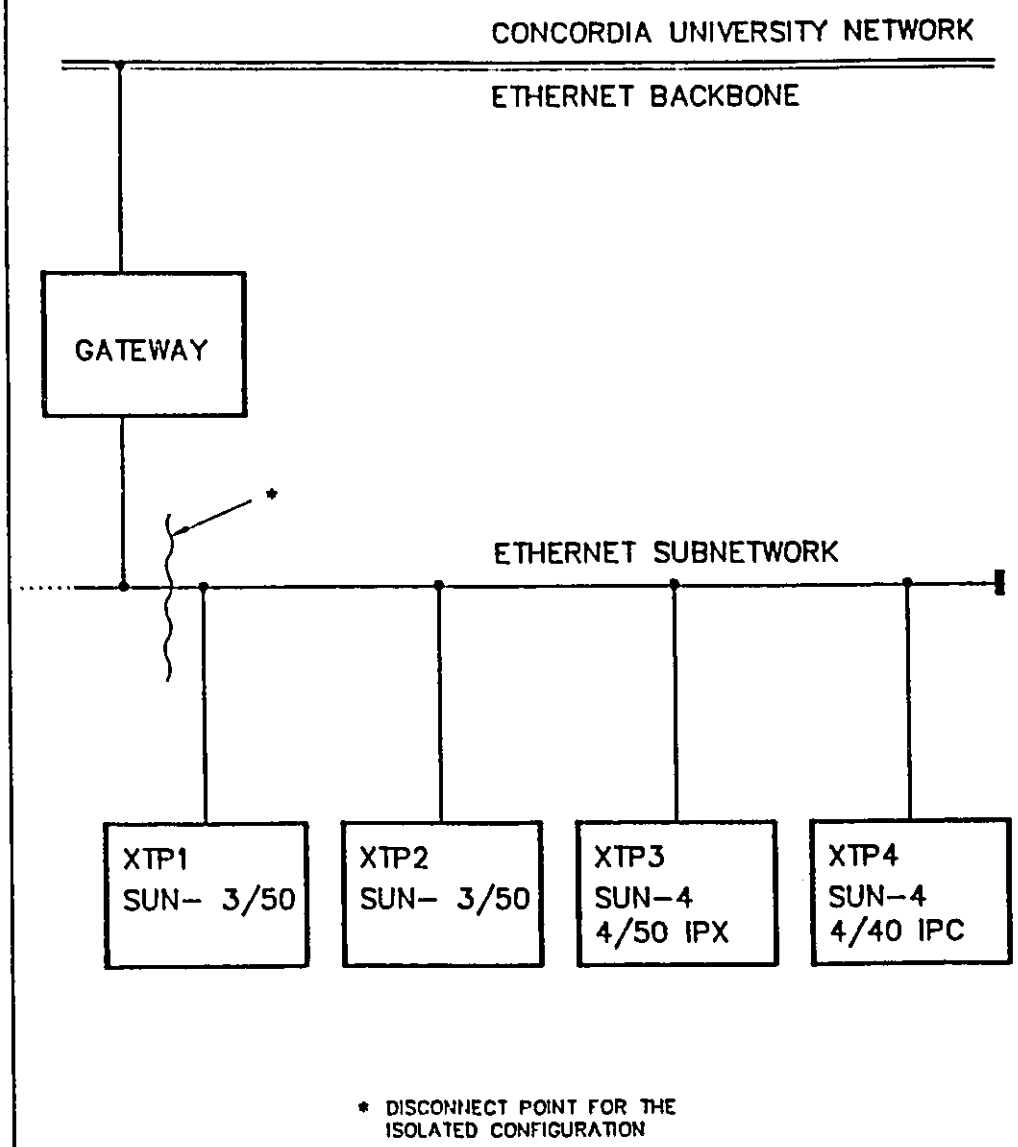

FIGURE 3 -
PERFORMANCE TESTS HARDWARE ARCHITECTURE.

4.4 Test Descriptions

This chapter describes the tests for measuring the throughput and delay performance of XTP (socket and character driver interfaces), TCP, UDP and Network Driver RAW interface.

For all the tests, the total number of bits transferred in a run was 2 MB, that is

BSIZE * NBUF = 2MB

All the experiments were performed 5 times (repetitions) and averaged. The buffer lengths (BSIZE) were between 16 Bytes and 16 KBytes: 16B, 32B, 64B, 128B, 256B, 512B, 1KB, 2KB, 4KB, 8KB, 16KB, for the majority of tests, except for the Network RAW driver interface (from 16B to 1KB) and for the UDP (from 16B to 8KB) tests.

All the tests for the XTP performance measurements were performed in unicast mode.

The following tests were performed for **throughput** measurements, for each of the two configurations (sun-3 and sun-4):

- XTP Socket Interface, one virtual circuit, receiver on xtp1, sender on xtp2,

I
- default XTP input/output queues sizes (16KB),
- data checksum enabled,

II
- default XTP input/output queues sizes,
- data checksum disabled,

III
- XTP receive, input/output queues sizes equal to 32KB,
- data checksum enabled.

• XTP Character Driver Interface, one virtual circuit, receiver on xtp1, sender on xtp2,

I ° default XTP input/output queues sizes (16KB),

° data checksum enabled,

II ° default XTP input/output queues sizes

° data checksum disabled,

III ° XTP receive, input/output queues sizes equal to 32KB,

° data checksum enabled,

• TCP/IP, one virtual circuit, receiver on xtp1, sender on xtp2,

• UDP, one process receiver on xtp1, one process sender on xtp2,

• Network Driver RAW interface (NIT), one process receiver on xtp1, one process sender on xtp2,

• XTP socket interface, two virtual circuits, receivers on xtp1, senders on xtp2, XTP input and output queues set to 32 KB,

• TCP two virtual circuits, receivers on xtp1, senders on xtp2,

• XTP socket interface, with XTP input/output queue set to 32 KB and TCP in the same time.

The following tests were performed for **delay** measurements, for the two configurations (sun-3 and sun-4):

• XTP Socket Interface, one virtual circuit, one process on xtp1, the other on xtp2,

I ° default XTP input/output queues sizes (16KB),

° data checksum enabled,

II ° default XTP input/output queues sizes,

° data checksum disabled,

III ° XTP receive, input/output queues sizes equal to 32KB,

° data checksum enabled.

• XTP Character Driver Interface, one virtual circuit, one process on xtp1, the other on xtp2,

I ° default XTP input/output queues sizes (16KB),

° data checksum enabled,

II ° default XTP input/output queues sizes

° data checksum disabled,

III ° XTP receive, input/output queues sizes equal to 32KB,

° datachecksum enabled,

• TCP/IP, one virtual circuit, one process on xtp1, the other on xtp2,

• UDP, one process receiver on xtp1, one process sender on xtp2.

• XTP socket interface, two virtual circuits, receivers on xtp1, senders on xtp2, with the XTP input/output queues set to 32 KB,

• TCP two virtual circuits, receivers on xtp1, senders on xtp2,

• XTP socket interface, with the XTP input/output queues set to 32 KB and TCP in the same time.

4.5 Program Descriptions

This chapter describes the programs for the performance measurements for the XTP, TCP, UDP and Network Driver Protocols.

All the throughput and delay measurement programs have some common parts and some parts that are specific to the protocol or to the delay and throughput measurements. A directory structure consisting of the common modules at the main level and the specific modules for each protocol grouped in a specific subdirectory has been used.

A makefile is provided in each protocol-specific subdirectory and can be used to build the test programs. Among other things, it specifies the directories where the include files can be found, the PROTO identifier, which is different for each protocol, and protocol specific identifier values, like the XTP protocol family (PF_XTP).

Beside standard include files, the programs include the following files:

"xtp_ioctl.h" - contains the XTP ioctl() definitions.

"xtp.h" - contains the XTP packet format definitions.

These two files are located in the directory /usr/share/sys/netxtp.

"perf.h" - contains common definitions for the performance programs, and is located in the main level programs directory.

4.5.1 Throughput Program Descriptions

The same design has been used for the main programs, which perform throughput measurements for all the tests. The exceptions will be described for each protocol. The program name is "thrput". It sets up a unidirectional data transfer and times the transfer. It has two parts: the receiver and the transmitter.

The receiver part performs the setup for reception, and then loops forever receiving user data sent by the transmitter. It is assumed that the receiver will be started before the transmitter, and block listening for the transmitter connect request.

The transmitter part performs the setup for transmission and then starts sending data according to the pairs (number of buffers, buffer size) read from an input file. Throughput calculations are performed independently for each such pair, for a specified number of times, and the results are averaged.

The program writes the results to the standard output. Redirection can be used to collect the results in a file.

The calling sequence for the receiver is:

thrput_r [-options]

and for the sender:

thrput_s [-options] host

The host is the host where the receiver resides, and can be specified as a host name or a host internet address.

The allowed options for both sender and receiver are:

-n## number of repetitions for the calculations performed on each pair (buffer length, number of buffers). The default is 1.

-c disables the data checksum. The default is data checksum computations and validations enabled.

-u mandatory for the UDP tests, not used for the other tests.

-p## the port number to send to or listen at. The default is IPPORT_USERRESERVED. This port is the first port reserved for non-privileged servers as defined on 4.3BSD systems.

-Q## max size of XTP input and output queues. The default is 16*1024 (16KB).

The main programs call the following common routines:

- init_time - initialize the time counters
- calc_time - calculate the time spent since init_time has been called

The following routines are also called. They are different from one protocol to another, and will be described in the section specific to each protocol:

- setup_rcv - setup for reception
- setup_transm - setup for transmission
- io_write - the transmission processing
- io_read - the reception processing

4.5.2 Delay Program Descriptions

The same design has been used for the main programs that perform delay measurements for all the tests. The program name is "dela". It sets up a bidirectional data transfer and times the transfer of a message from one host to the other and back. It has two parts: the "client" and the "server".

The server part performs the setup for communication with any client, receives the user data sent by the client, according to the pairs (buffer size, number of buffers) read from an input file, and then sends it back. It is assumed that the server will be started before the client, and will block listening for the connection request from a client.

The client part performs the setup for communication with the server, sends

data according to the pairs (number of buffers, buffer size) read from the same input file as the server, and then receives the data sent by the server. Delay calculations are performed independently for each such pair, for a specified number of repetitions, and the results are averaged.

The program writes the results to the standard output. Redirection can be used to collect the results in a file.

The calling sequence for the server is:

dela -r [-options]

and for the sender:

dela -t [-options] host

The host is the host where the server resides, and can be specified as a host name or a host internet address.

The allowed options for both client and server are:

-n## number of repetitions for the calculations performed on each pair (buffer length, number of buffers). The default is 1.

-c disables the data checksum. The default is data checksum computations and validations enabled.

-u mandatory for the UDP tests, not used for the other tests.

-p## the port number to send to or listen at. The default is IPPORT_USERRESERVED. This port is the first port reserved for non-privileged servers as defined on 4.3BSD systems.

-Q## max size of XTP input and output queues. The default is 16*1024 (16KB).

The main programs call the following common routines:

- init_time - initialize the time counters
- calc_time - calculate the time spent since init_time has been called

The following routines are also called. They are different from one protocol to another, and will be described in the section specific to each protocol:

- setup_rcv - setup for reception
- setup_transm - setup for transmission
- io_write - the transmission processing
- io_read - the reception processing

4.5.3 XTP Socket Interface Specific Routines

The following routines are specific to the XTP socket interface: setup_transm, setup_rcv, io_write, io_read.

The routine **setup_transm** performs the following major steps to set up the transmission:

It gets as input arguments the receiver host name or internet address, the receiver port number and protocol options such as the maximum size of the XTP output queue and the checksum option.

It creates a stream socket (SOCK_STREAM) with the XTP as underlying communication protocol (domain PF_XTP), binds the socket by specifying the local port as 0 (which means a free choice port), and sets the maximum size of the XTP output queue if this option has been requested, by using an XTP specific ioctl. It also disables the data checksum calculation if this option has been requested. Then it uses connect to initiate a connection to the receiver's socket, by specifying the Internet address and

port number of the receiver.

It returns as its output argument the file descriptor of the socket to be used for transmission.

The routine **setup_rcv** performs the following major steps to set up the reception:

It gets as input arguments its port number and the protocol options such as the maximum size of the XTP input queue and the checksum option.

It creates a stream socket (SOCK_STREAM) with the XTP as underlying communication protocol (domain PF_XTP), binds the socket by specifying the local port, uses listen to indicate it is ready to listen for incoming connection requests and accept, to accept the connection requested by the transmitter. Accept will return only when a connection is available or the system call is interrupted by a signal to the process. On receipt of a connection from the transmitter, accept will return a new descriptor along with a new socket to be used for data transfer. The routine also sets the maximum size of the XTP input queue and disables the checksum calculation, if these options have been requested.

The routine returns as its output arguments the file descriptor of the socket to be used for reception, and the host name of the client.

The routine **io_write** performs a send system call specifying the file descriptor obtained when setting up the transmission.

The routine **io_read** performs a read system call specifying the file descriptor obtained when setting up the reception. In the case of the delay program, this routine calls read several times until the number of bytes requested has been reached.

4.5.4 XTP Character Driver Interface Specific Routines

The receive part of the **throughput** program is different from the general model used. It queues the receive requests according to the pairs (buffer size, number of buffers) read from the same input file as the transmitter.

The **delay** program is also different from the general model, since the character driver interface does not allow bidirectional transfer on the same XTP device. Therefore an XTP device is used for transmission, with the ports (port, port+1), and another XTP device for reception with the ports (port+2, port+3), where port is the default port, or the port specified in the run command. Another difference is due to the fact that the current version of KRM does not support blocking listen functionality. As both the server and the client perform setup for reception and transmission, a mechanism for synchronizing these setup procedures has been implemented. As any available Inter Process Communication (IPC) facility which supports inter host communication could be used, the BSD sockets have been chosen. The scheme using two synchronization sockets, allows the synchronization between the client and server at initialization. Both processes perform the setup for the synchronization sockets. The server then performs the setup for reception and blocks on a read on the first synchronization socket. The client performs the setup for transmission, the setup for reception, then writes to the first synchronization socket, and then blocks on a read on the second synchronization socket. At this time, the server read unblocks, it will perform the setup for transmission and then write to the second synchronization socket, unblocking the client. After this, the transfer of data can be performed as in the general model. One more difference consists in the specification of the client host name when running the server. This scheme avoids a more complicated mechanism

for getting the client name.

The following routines are also specific to the XTP socket interface: setup_transm, setup_rcv, io_write, io_read.

The routine **setup_rcv** performs the following major steps to set up the reception:

It gets as input arguments its port number and the protocol options such as the maximum size of the XTP input queue and the checksum option.

It opens one of the devices associated with one of the character device driver (/dev/xtp00,..., /dev/xtp19), which is not busy, specifies the address to be used for filtering incoming FIRST packets by using an ioctl with the XTPIOCLISTEN_NW parameter, and sets the maximum size of XTP input queue and disables the data checksum calculations, if these options have been requested.

The routine returns as its output argument the file descriptor of the XTP device to be used for reception.

The routine **setup_transm** performs the following major steps to set up the transmission:

It gets as input arguments the port number and the protocol options such as the maximum size of the XTP output queue and the checksum option.

It opens one of the XTP devices associated with the character device driver (/dev/xtp00,..., /dev/xtp19), which is not busy, performs an ioctl call to set the maximum size of XTP output queue and to disable the checksum calculations, if these options have been requested, and initiates a connection (send a FIRST packet) by using an ioctl with the XTPIOCCONNECT_NW parameter.

The routine returns as its output argument the file descriptor of the XTP

device to be used for reception.

The routine **io_write** performs a write system call specifying the file descriptor obtained when setting up the transmission.

The routine **io_read** performs repeated read system calls specifying the file descriptor obtained when setting up the reception, until the number of bytes requested has been reached.

4.5.5 TCP Specific Routines

The following routines are specific to the TCP programs: setup_transm, setup_rcv, io_write, io_read.

The routine **setup_transm** performs the following major steps to setup the transmission:

It gets as input arguments the receiver host name or internet address, and the receiver port number.

It creates a stream socket (SOCK_STREAM) with TCP as the underlying communication protocol (domain AF_INET), and binds the socket by specifying the local port as 0 (which means a free choice port). Then it uses connect to initiate a connection to the receiver's socket, by specifying the Internet address and port number of the receiver.

It returns as its output argument the file descriptor of the socket to be used for transmission.

The routine **setup_rcv** performs the following major steps to set up the reception:

It gets as input argument its port number.

It creates a stream socket (SOCK_STREAM) with TCP as the underlying communication protocol (domain AF_INET), binds the socket by specifying the local port, uses listen to indicate it is ready to listen for incoming connection requests and accept, to accept the connection requested by the transmitter. Accept will return only when a connection is available or the system call is interrupted by a signal to the process. On receipt of a connection from the transmitter, accept will return a new descriptor along with a new socket to be used for data transfer.

The routine returns as its output arguments the file descriptor of the socket to be used for reception, and the host name of the client.

The routine **io_write** performs a send system call specifying the file descriptor obtained when setting up the transmission.

The routine **io_read** performs a read system call specifying the file descriptor obtained when setting up the reception. In the case of the delay program, this routine will call read several times until the number of bytes requested has been reached.

4.5.6 UDP Specific Routines

The transmit part of the **throughput** program is different from the general model used. It sends a "sentinel" size message to "start" and "end" the receiver, and in addition to the socket file descriptor, it also uses the socket structure when sending data.

The **delay** program is also different from the general model. Both the client and server, when sending data, send a "sentinel" size message to "start" and "end" the receiver, and in addition to the socket file descriptor, they also use the socket structure when sending data. Another difference consists in the specification of the

client host name when running the server. This scheme avoids a more complicated mechanism for getting the client name.

The following routines are also specific to the UDP programs setup_transm, setup_rcv, io_write, io_read.

The routine **setup_transm** used for throughput measurements performs the following major steps to set up the transmission:

It gets as input arguments the receiver host name or internet address and the receiver port number.

It creates a datagram socket (SOCK_DGRAM), binds the socket by specifying the local port as 0 (which means a free choice port).

It returns as its output arguments the file descriptor of the socket to be used for transmission, and the socket structure itself.

The routine **setup_transm** used for delay measurements is similar to that used for throughput but, because the socket is used for bidirectional transfer of data, the local port is not a free choice. The value port+1 is used instead. The value for the remote port is port.

The routine **setup_rcv** used for throughput measurements, performs the following major steps to set up the reception:

It gets as input argument its port number.

It creates a datagram socket (SOCK_DGRAM), binds the socket by specifying the local port.

The routine returns as its output argument the file descriptor of the socket to be used for reception.

The routine **setup_rcv** used for delay measurements is similar to that used for

throughput, but the client host name or internet address, is also specified as argument. Since the socket is used for bidirectional transfer of data, the required setup for transmission is also performed. The value port is used for the local remote socket, and the value port+1 for the local socket.

The routine returns the socket structure used for transmission, beside the file descriptor.

The routine **io_write** performs a sendto system call specifying the file descriptor and the socket structure, obtained when setting up the transmission.

The routine **io_read** performs one or more recvfrom system calls specifying the file descriptor obtained when setting up the reception. Due to the fact that UDP does not preserve message boundaries, the recvfrom system call may be called several times until the number of bytes requested has been reached.

4.5.7 Network Driver RAW Interface Specific Routines

The Network Interface Tap (NIT) is a facility specific to the SunOS, which provides modalities to directly access the link-level network at user level, not within the kernel. It consists of several STREAMS modules and drivers.

The NIT has been used for the throughput performance tests of the link-level network driver.

All the three components of NIT have been used. The nit_if module has been used for reading and writing data to the Ethernet driver. The nit_pf (filter) module has been used to discard all the data other than the "test type" packets. The nit_buf (buffering) module has been used to reduce the overhead resulting from repeated read system calls.

The receiver part of the throughput program performs the setup for reception,

and then loops reading.

The transmitter part performs the setup for transmission, prepares the STREAMS control and data and starts sending data according to the pairs (number of buffers, buffer size) read from an input file, using the putmsg system call.

The routine **setup_rcv** performs the following major steps to set up the reception:

It gets chunk size used for buffering, as argument.

It opens the NIT device, pushes the protocol filter, and arranges to receive only the ETHERTYPE_TEST type packets. It then pushes and configures the buffering module, and configures the nit device, binding it to the proper underlying interface (LANCE), "le0".

It returns as its output argument the file descriptor to be used for reception.

The routine **setup_transm** performs the following major steps to set up the transmission reception:

It gets the receiver's host name as argument.

It opens the NIT device, and configures it, binding it to the proper underlying interface (LANCE), "le0". It also prepares the Ethernet header for the packets to be sent, where source Ethernet address is the local host Ethernet address, the destination Ethernet address is the receiver's host Ethernet address and the Ethernet packet type is ETHERTYPE_TEST, defined in the include file "ethraw.h".

It returns as its output argument the file descriptor to be used for transmission.

Chapter 5

Results and Analysis of Performance Measurements

All the results represent only the data transfer itself, without the time spent to establish the connections, except for the UDP. This has been achieved by ignoring a first set of transfers.

All the graphs corresponding to the throughput measurements, plot the message size on the horizontal axis (in Bytes or KBytes) versus throughput on the vertical axis (in Mbps).

All the graphs corresponding to the delay measurements, plot the message size on the horizontal axis (in Bytes or KBytes) versus delays on the vertical axis (in Milliseconds).

All the graphs corresponding to the receiver's throughput versus offered load, plot the offered load (transmitter's throughput) on the horizontal axis (in Mbps) versus receiver's throughput on the vertical axis (in Mbps).

5.1 Measurements for the Sun-3 Configuration

5.1.1 Throughput Measurements for XTP

5.1.1.1 Socket Interface

For the XTP socket interface, measurements for three cases have been performed: the default, disable data checksum, and XTP input and output queues set to 32 KB instead of 16 KB.

The results are shown in Table 1 and Figures 4, 5, for the case of one virtual circuit.

The graph in Figure 4 plots the results for the XTP socket interface, in all three cases, for message size from 512 Bytes to 16 KB. The best throughput performance is for the case where the XTP input/output queues are 32 KB. The maximum throughput is 2.5 Mbps for the message size of 16 KB. That means that maximum 25% of the 10 Mbps raw bandwidth can be delivered to the user.

In the case of data checksum disabled, for the message size up to 4 KB, the performance is slightly better than for the default case, which means that the checksum processing is quite performant. An anomaly can be observed for message sizes 8 KB and 16 KB, where the performance is better in the default case, than in the data checksum disabled case.

The graph in Figure 5 plots the results for the XTP socket interface, in all three cases, for message size from 16 Bytes to 512 Bytes. Although the checksum disabled and XTP I/O queues set to 32 KB cases have a slightly better performance, all the three cases are similar, which confirms the fact that the checksum processing is performant and indicates that for message sizes between 16 B and 512 B it is not worth to increase the XTP input/output queues from 16 KB to 32 KB, mainly if we are in a configuration short of physical memory.

Message size	XTP socket default	XTP socket checksum disabled	XTP socket i/o que 32 KB	XTP char default	XTP char checksum disabled	XTP char i/o que 32 KB
	Mb / sec					
16 B	0.03	0.04	0.02	0.03	0.03	0.02
32 B	0.09	0.09	0.07	0.08	0.07	0.07
64 B	0.19	0.2	0.16	0.16	0.13	0.16
128 B	0.3	0.37	0.33	0.31	0.26	0.31
256 B	0.58	0.69	0.67	0.56	0.46	0.54
512 B	1.04	1.21	1.22	1.03	0.86	1.02
1 KB	1.56	1.56	1.95	1.56	1.34	1.76
2 KB	1.56	2.05	2.02	1.56	1.97	1.84
4 KB	2.07	2.09	2.26	2.07	2.08	2.13
8 KB	2.24	1.58	2.2	2.22	2.39	2.29
16 KB	2.07	1.57	2.5	2.07	1.57	2.39

TABLE 1

TROUGHPUT MEASUREMENTS
XTP SOCKET AND CHARACTER DRIVER INTERFACES
SUN-3, ONE VIRTUAL CIRCUIT

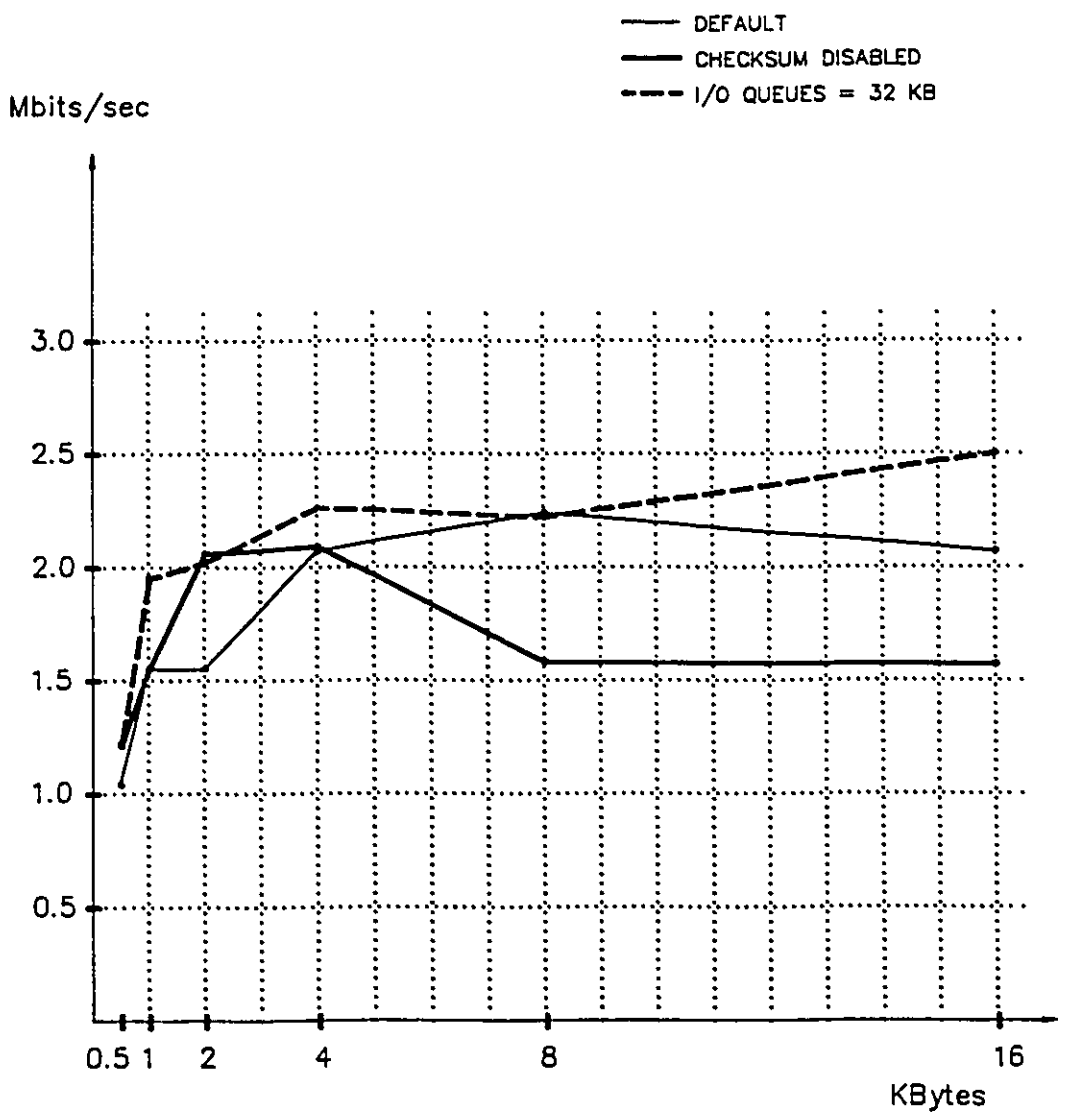

FIGURE 4

TROUGHPUT MEASUREMENTS
XTP SOCKET INTERFACE
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

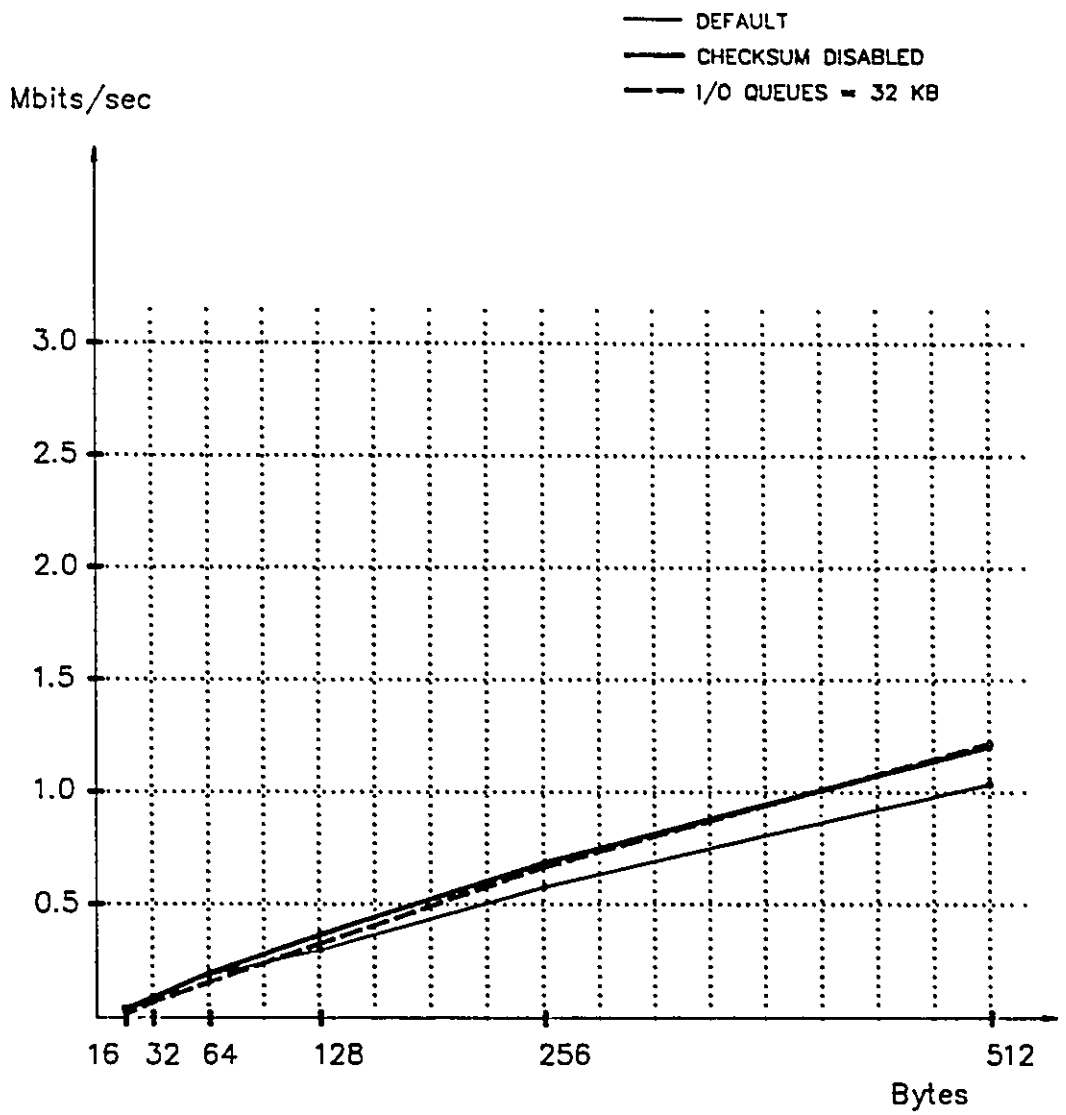

FIGURE 5

TROUGHPUT MEASUREMENTS
XTP SOCKET INTERFACE
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.1.1.2 Character Driver Interface

For the XTP character driver interface, measurements for the same three cases have been performed: the default, disable data checksum, and XTP input and output queues set to 32 KB instead of 16 KB. The results are shown in Table 1 and Figures 6, 7, for the case of one virtual circuit.

The graph in Figure 6 plots the results for the XTP character driver interface, in all three cases, for message size from 512 Bytes to 16 KB. The best throughput performance is in general for the case where the XTP input/output queues are 32 KB. The maximum throughput is 2.39 Mbps for the message size of 16 KB. That means that maximum 23.9 % of the 10 Mbps raw bandwidth can be delivered to the user. For message sizes 2 KB and 8 KB the performance is better for the case of data checksum disabled. The same anomaly as in the socket interface can be observed for message size 16 KB, where the performance is better in the default case, than in the data checksum disabled case.

The graph in Figure 7 plots the results for the XTP character driver interface, in all three cases, for message size from 16 Bytes to 512 Bytes. All three cases have a similar performance, although the default case has a slightly better performance for some message sizes.

A general comment applies to both interfaces. The throughput increases abruptly and linearly to about 2 Mbps for message sizes up to about 1 KB, and continues only with a slight increase up to 2.5 Mbps for message sizes up to 16 KB.

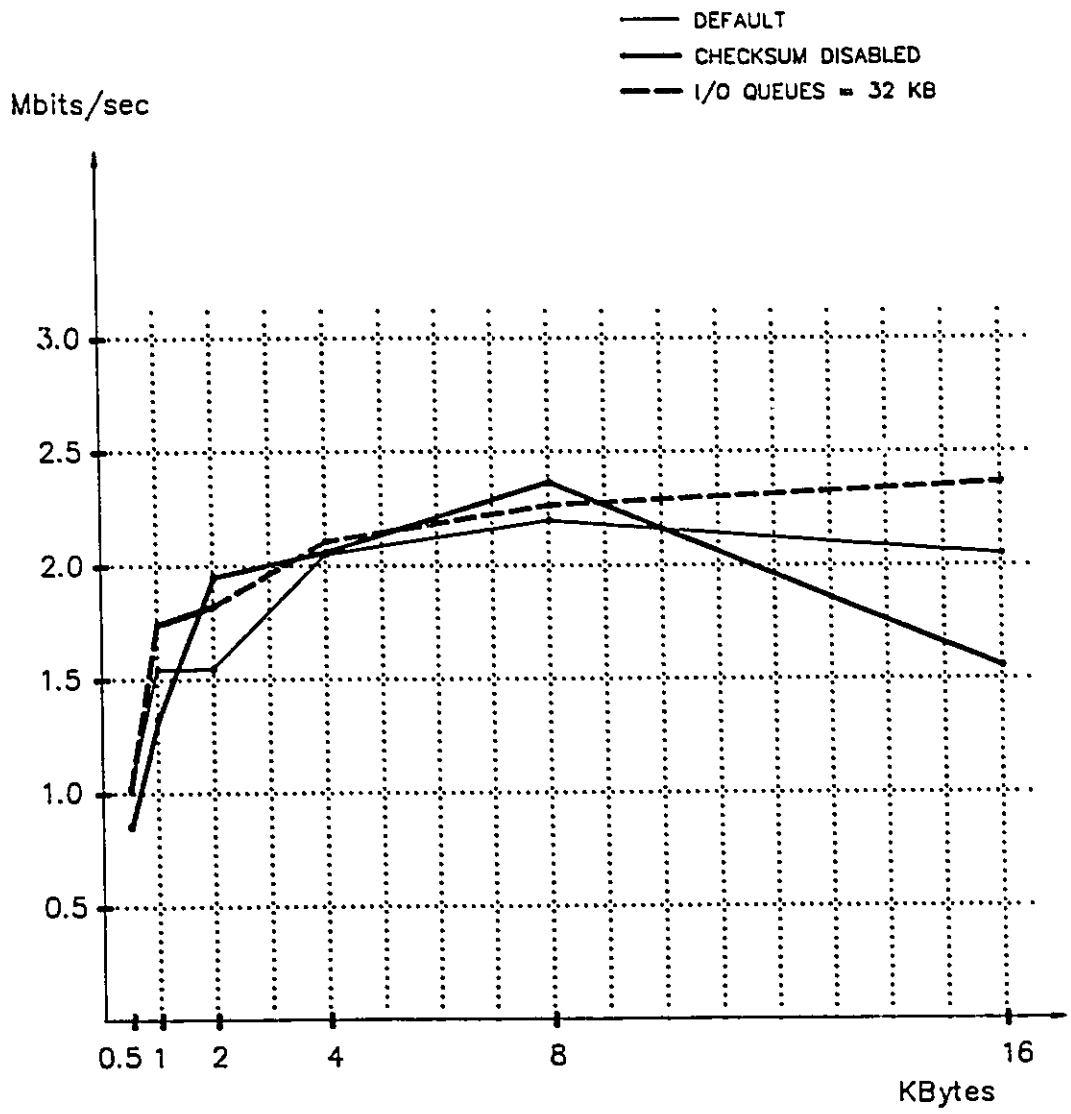

FIGURE 6

TROUGHPUT MEASUREMENTS
XTP CHARACTER DRIVER INTERFACE
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

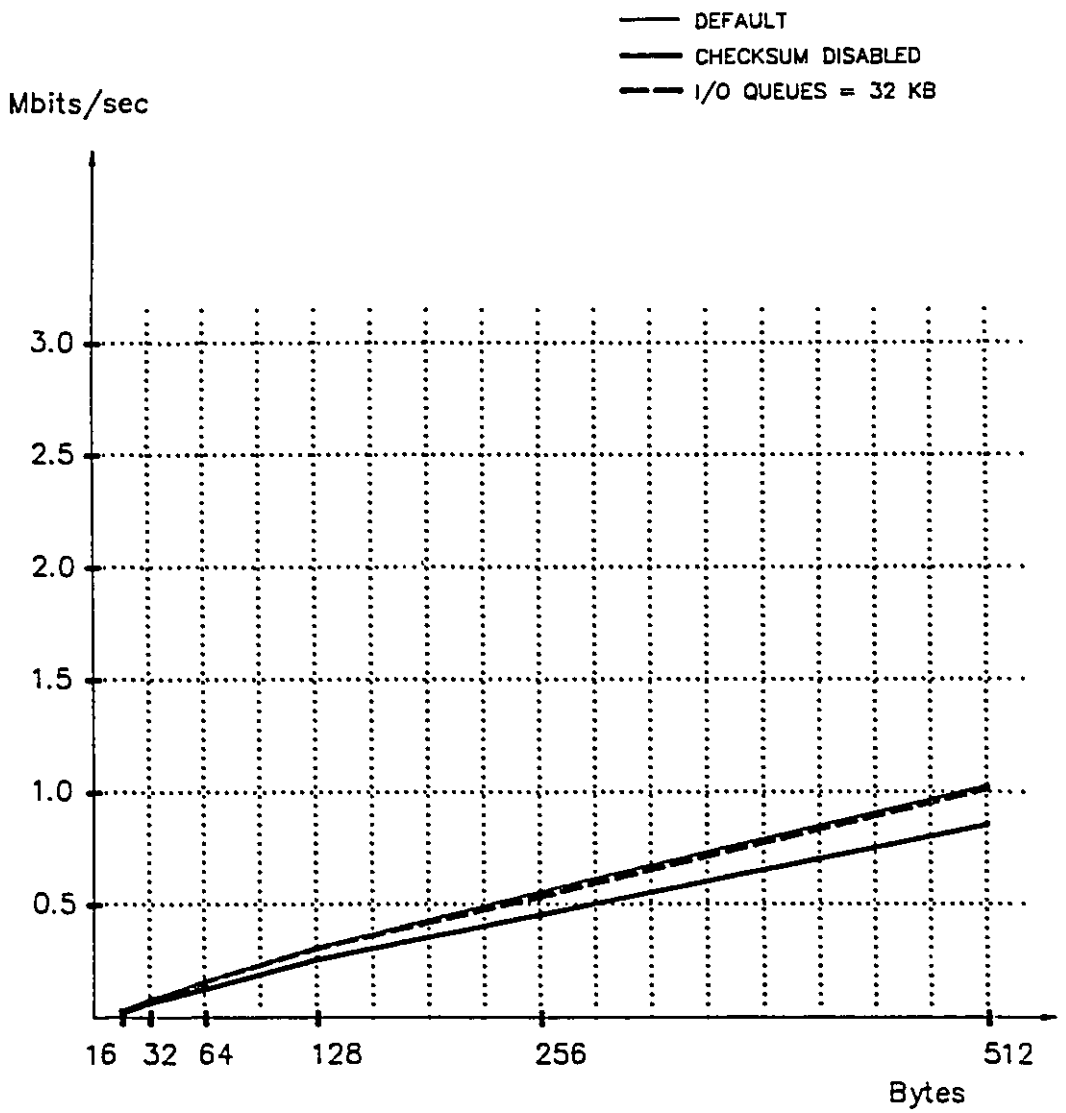

FIGURE 7

TROUGHPUT MEASUREMENTS
XTP CHARACTER DRIVER INTERFACE
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.1.2 Throughput Measurements for TCP

The throughput measurements for TCP, for one virtual circuit are shown in Table 2 and Figures 8 and 9. The maximum throughput obtained is 1.95 Mbps for message size 16 KB. That means that maximum 19.5 % of the 10 Mbps raw bandwidth can be delivered to the user.

5.1.3 Throughput Measurements for UDP

The throughput measurements for UDP, for one receiver and one sender are shown in Table 2 and Figures 8 and 9. The maximum throughput obtained is 4.94 Mbps for message size 8 KB. But in this case, at the receiver side 99.22% of the messages are lost. In order for the receiver to get most of the messages, the transmitter has to be slowed down. The subchapter 5.3 describes the measurements of the receiver's throughput versus the offered load (transmitter's throughput). The offered load of 4.94 Mbps has been obtained when the transmitter sends as fast as it can. When delays are added between every message, lower offered loads can be obtained. At an offered load of 2.07 Mbps, the receiver throughput is 2.16 Mbps and no messages are lost. That means that maximum 20.7% of the 10 Mbps raw bandwidth can be delivered to the user, without losing messages. Depending on the requirements of an application using UDP, the offered load could be increased, in order to obtain a better receiver throughput, and keep an acceptable percentage of messages lost.

5.1.4 Throughput Measurements for the RAW Network Driver

The throughput measurements for the RAW Network Driver are shown in

Table 2 and Figures 8 and 9.

The RAW Network Driver is implemented at the user level, as opposed to the other protocols in the study, which are implemented within the kernel. So that, although the protocol is very simple, without any error or flow control, the maximum throughput obtained is 2.16 Mbps for message size 1 KB. That means that maximum 21.6 % of the 10 Mbps raw bandwidth can be delivered to the user.

Message size	TCP	UDP	RAW	XTP socket i/o que 32 KB
	Mb / sec			
16 B	0.16	0.08	0.04	0.02
32 B	0.3	0.16	0.09	0.07
64 B	0.51	0.29	0.18	0.16
128 B	0.67	0.55	0.35	0.33
256 B	1.01	0.98	0.68	0.67
512 B	1.63	2.04	1.27	1.22
1 KB	1.75	3.49	2.16	1.95
2 KB	1.91	4.04	—	2.02
4 KB	1.92	4.71	—	2.26
8 KB	1.87	4.94	—	2.2
16 KB	1.95	—	—	2.5

TABLE 2

TROUGHPUT MEASUREMENTS
XTP,TCP,UDP,RAW
SUN-3, ONE VIRTUAL CIRCUIT

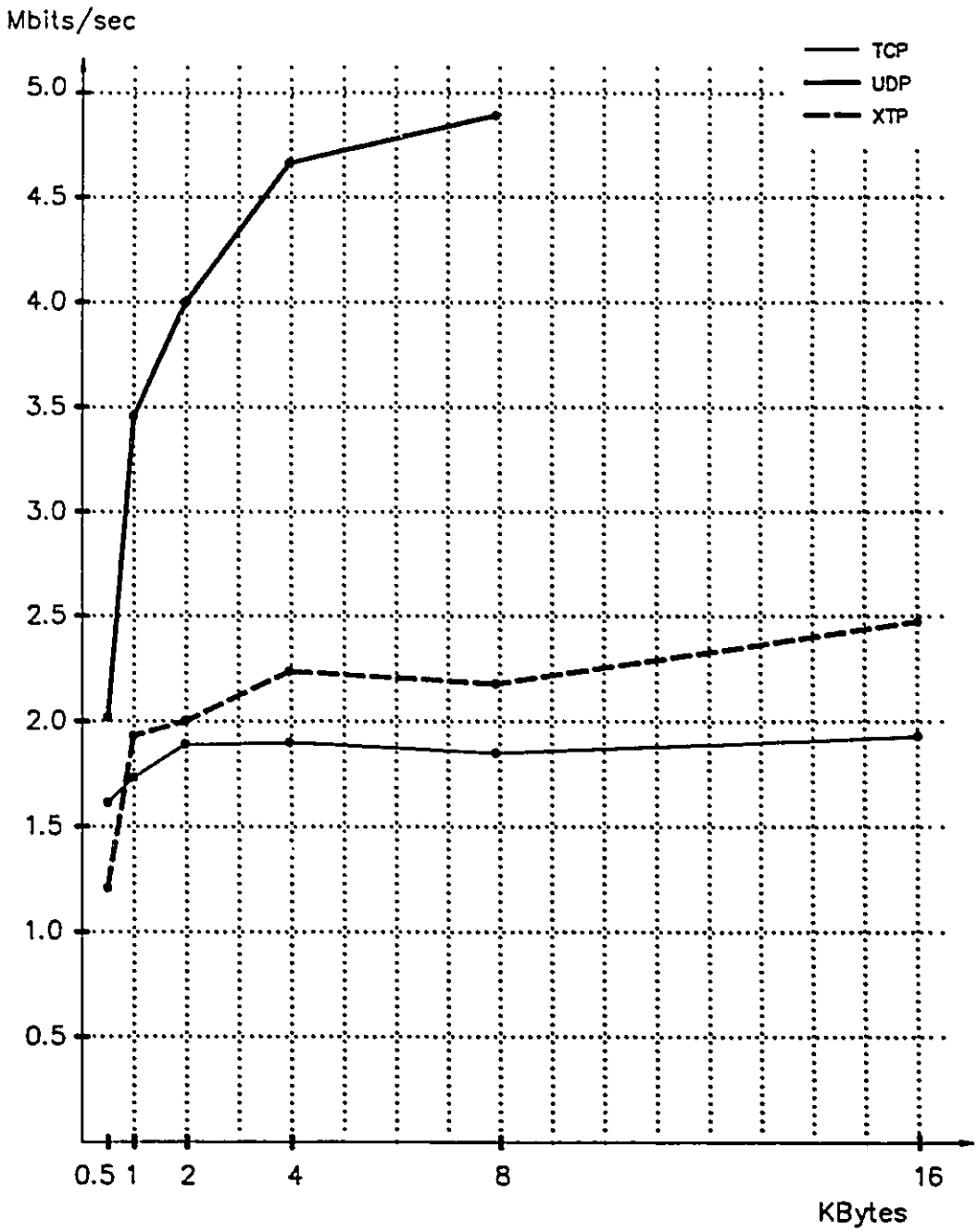

FIGURE 8
TROUGHPUT MEASUREMENTS
XTP,TCP,UDP
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

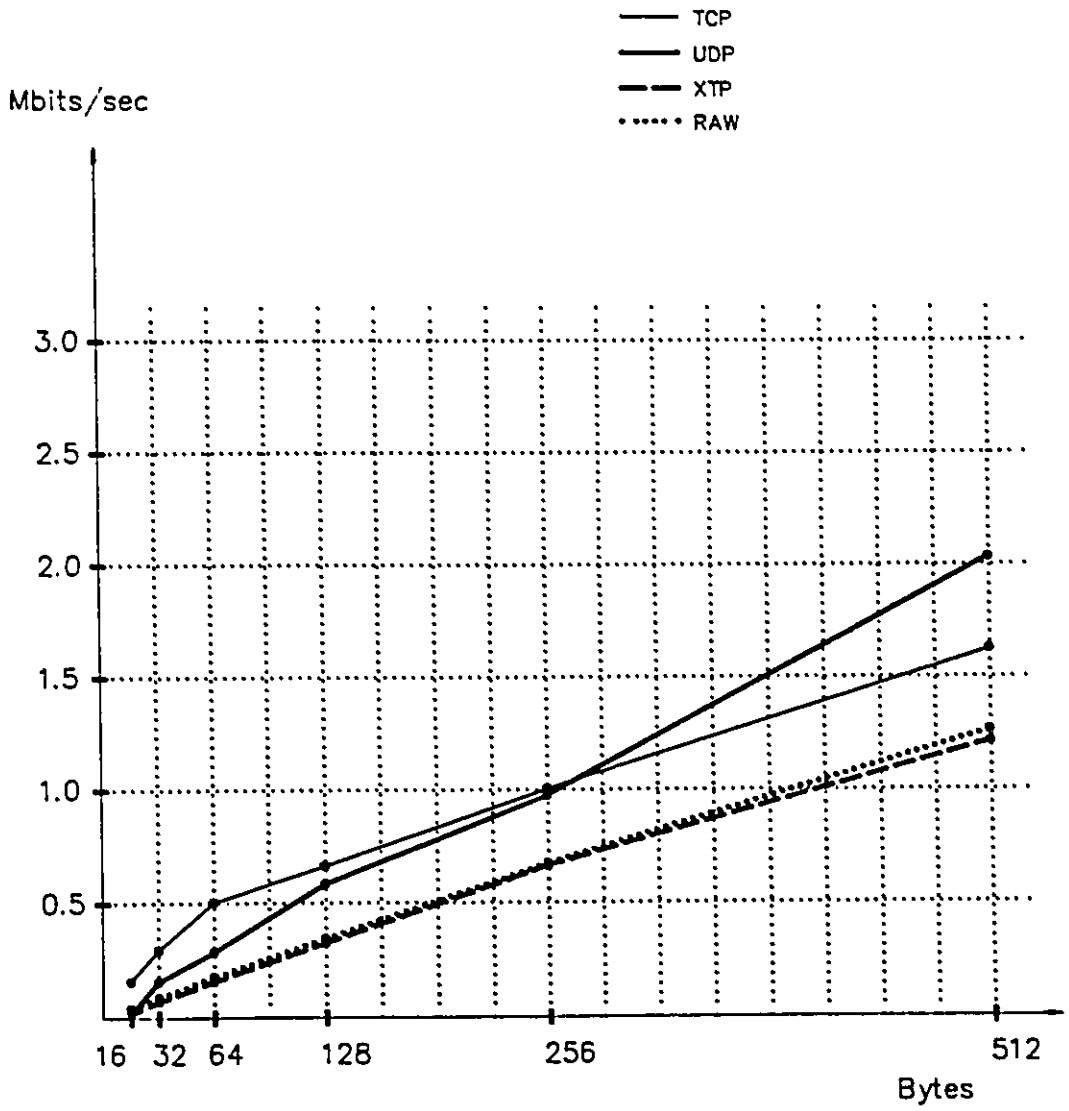

FIGURE 9

TROUGHPUT MEASUREMENTS
XTP,TCP,UDP,RAW
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.1.5 Throughput Comparisons

The performance difference between the two XTP interfaces, the socket and character driver, is not substantial, although the socket interface proves better for message sizes from about 256 B up. This is the result of the technique used by the character driver interface, which copies short data buffers and locks larger buffers down into memory.

For comparison with the other protocols, the XTP socket interface, the case where the XTP input/output queues are 32 KB, is used.

The Table 2 and Figures 8 and 9, show the results for XTP, TCP, UDP and the RAW Network Driver.

Although the UDP shows better performance, for message sizes from 256 B to 16 KB, this is the case where most of the messages are lost. In the case where a low percentage of messages are lost, UDP's performance is better than TCP, and slightly worse than XTP, for message size 8 KB.

The XTP performance is better than TCP, for message sizes from 1 KB to 16 KB, and worse for message sizes 16 B to 1 KB.

The RAW Network driver performance is very close to that of XTP for the message sizes measured (16 B - 1 KB).

The following list summarizes the results for the maximum throughput performance for all the protocols in study. From the maximum 10 Mbps of the Ethernet raw bandwidth the maximum percentage throughput delivered to the user, for each protocol is shown:

- 25% XTP sock for 16 KB message size,
- 23.9% XTP char for 16 KB message size,

- 21.6 % RAW Network Driver for 1 KB message size,
- 20.7% UDP for 8 KB message size (no message lost),
- 19.5 % TCP for 16 KB message size.

Therefore, XTP achieves the best throughput performance, for message sizes between 1 KB and 16 KB.

Although these results are for the XTP socket interface, the case where the XTP input/output queues are 32 KB, the default case also proves better for the message sizes from 4 KB to 16 KB.

Another set of tests have been performed to be able to compare the performance of XTP and TCP when transfers take place on two virtual circuits, in the same time. The results are shown in Table 3 and Figures 10, 11, 12, 13.

When two TCP virtual circuits are running in the same time, the maximum throughput percentage of the 10 Mbps raw bandwidth drops from 19.5% to 11.9%.

When two XTP virtual circuits are running in the same time, the maximum throughput percentage of the 10 Mbps raw bandwidth drops from 25% to 10.5%.

When one TCP virtual circuits is transferring data in the same time with an XTP virtual circuit, the maximum throughput percentage of the 10 Mbps raw bandwidth drops from 19.5% to 2.3% for TCP, and from 25% to 15.9% for XTP.

When the two virtual circuits transfer the same data, to different nodes, a multicast facility could be used, instead, to provide faster service. Such a facility is available for XTP only.

Message size	TCP 1 circuit	TCP 2 circuits	XTP socket i/o que 32 KB 1 circuit	XTP socket i/o que 32 KB 2 circuits	TCP with XTP	XTP socket with TCP
	Mb / sec					
16 B	0.16	0.08	0.02	0.02	0.07	0.03
32 B	0.3	0.15	0.07	0.07	0.14	0.06
64 B	0.51	0.25	0.16	0.1	0.23	0.10
128 B	0.67	0.33	0.33	0.18	0.30	0.22
256 B	1.01	0.51	0.67	0.33	0.45	0.48
512 B	1.63	0.94	1.22	0.58	0.82	0.84
1 KB	1.75	1.05	1.95	0.94	0.97	1.48
2 KB	1.91	1.10	2.02	0.96	1.04	1.38
4 KB	1.92	1.15	2.26	1.1	0.95	1.64
8 KB	1.87	1.18	2.2	1.09	0.47	1.60
16 KB	1.95	1.19	2.5	1.05	0.23	1.59

TABLE 3

TROUGHPUT MEASUREMENTS
XTP,TCP
SUN-3 TWO VIRTUAL CIRCUITS
VS. ONE VIRTUAL CIRCUIT

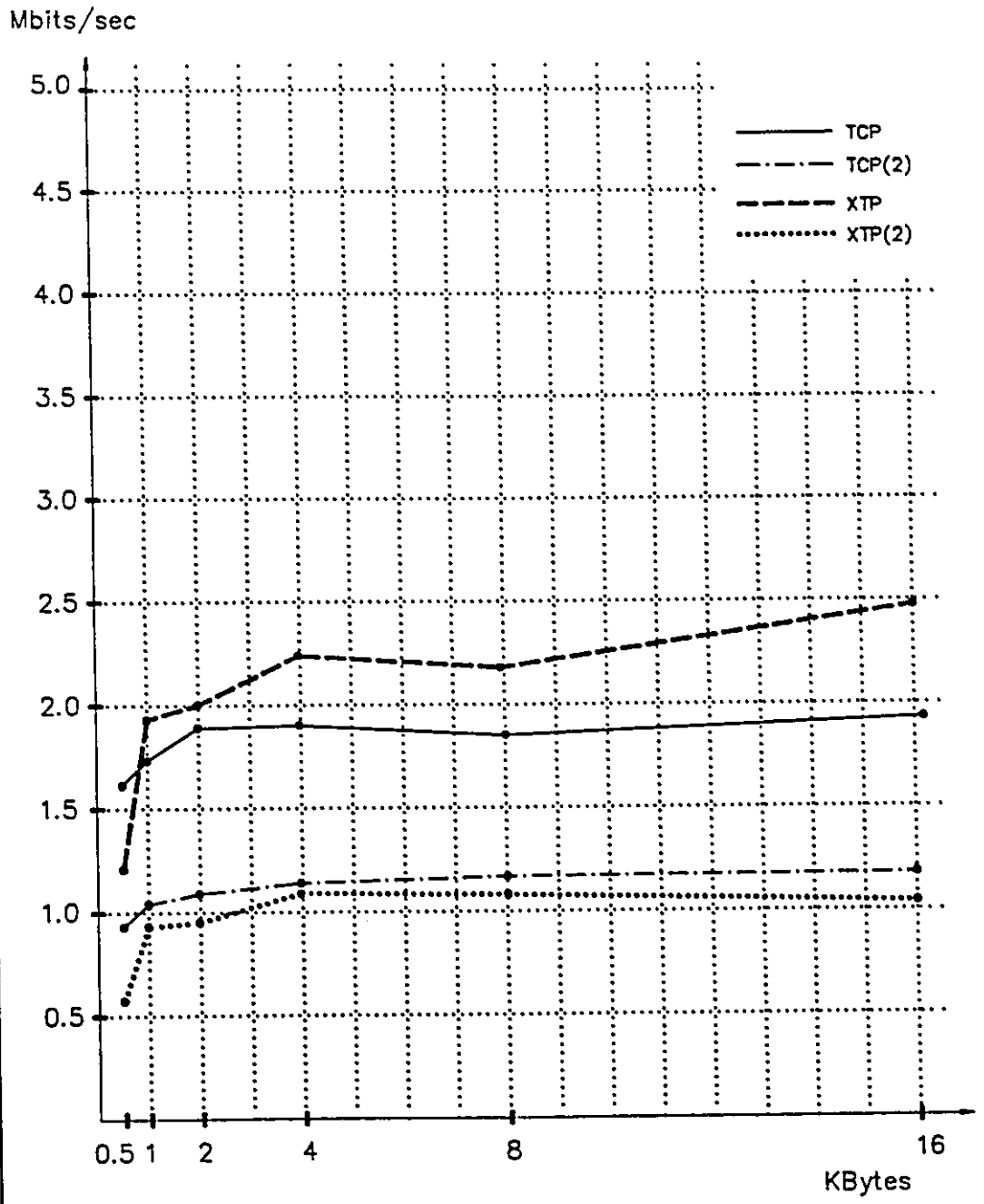

FIGURE 10

TROUGHPUT MEASUREMENTS
XTP,TCP
SUN-3, ONE VIRTUAL CIRCUIT
VS.TWO VIRTUAL CIRCUITS
MESSAGE SIZE : 512 B - 16 KB

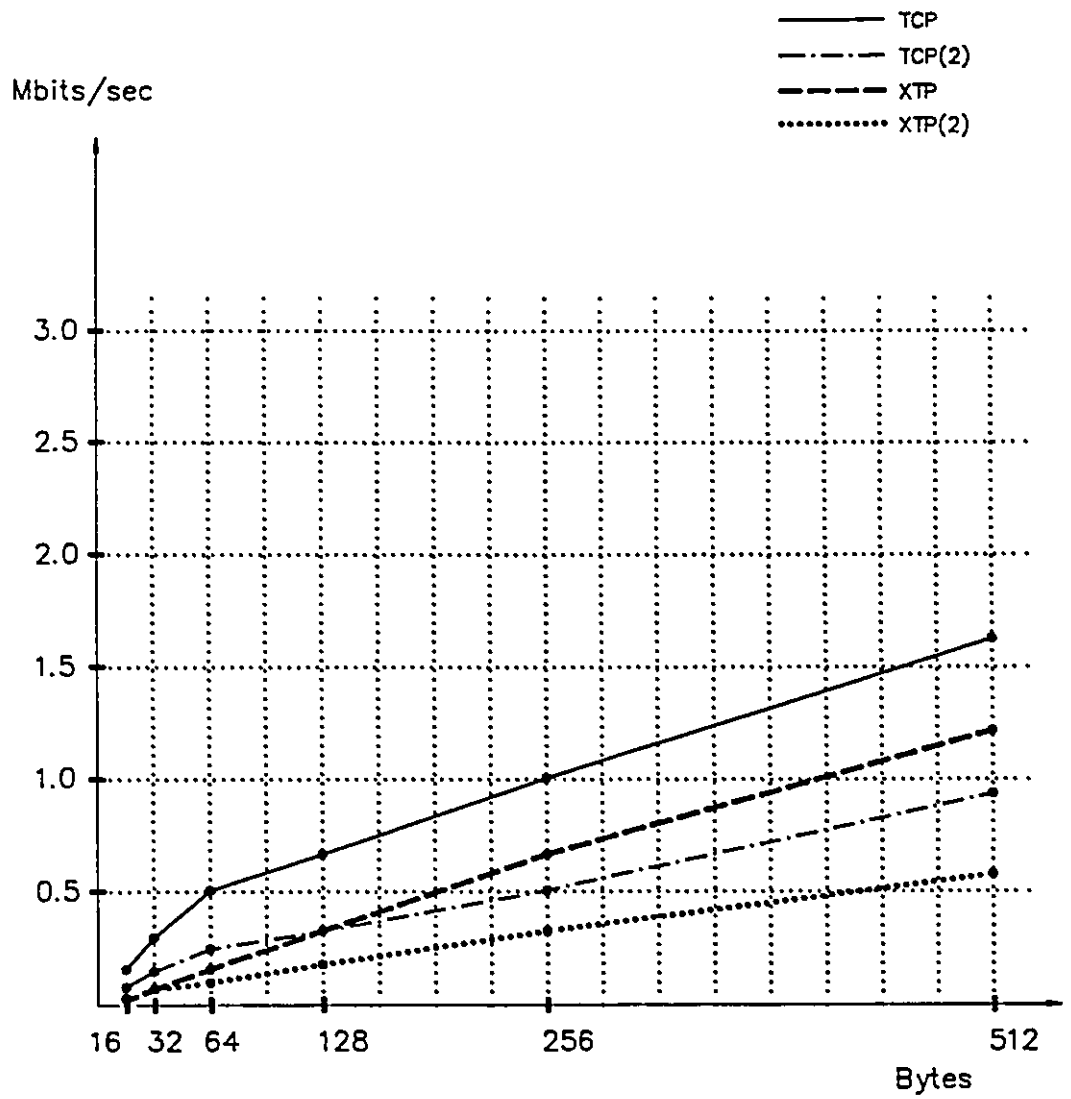

FIGURE 11

TROUGHPUT MEASUREMENTS
XTP,TCP
SUN-3, ONE VIRTUAL CIRCUIT
VS.TWO VIRTUAL CIRCUITS
MESSAGE SIZE : 16 B - 512 B

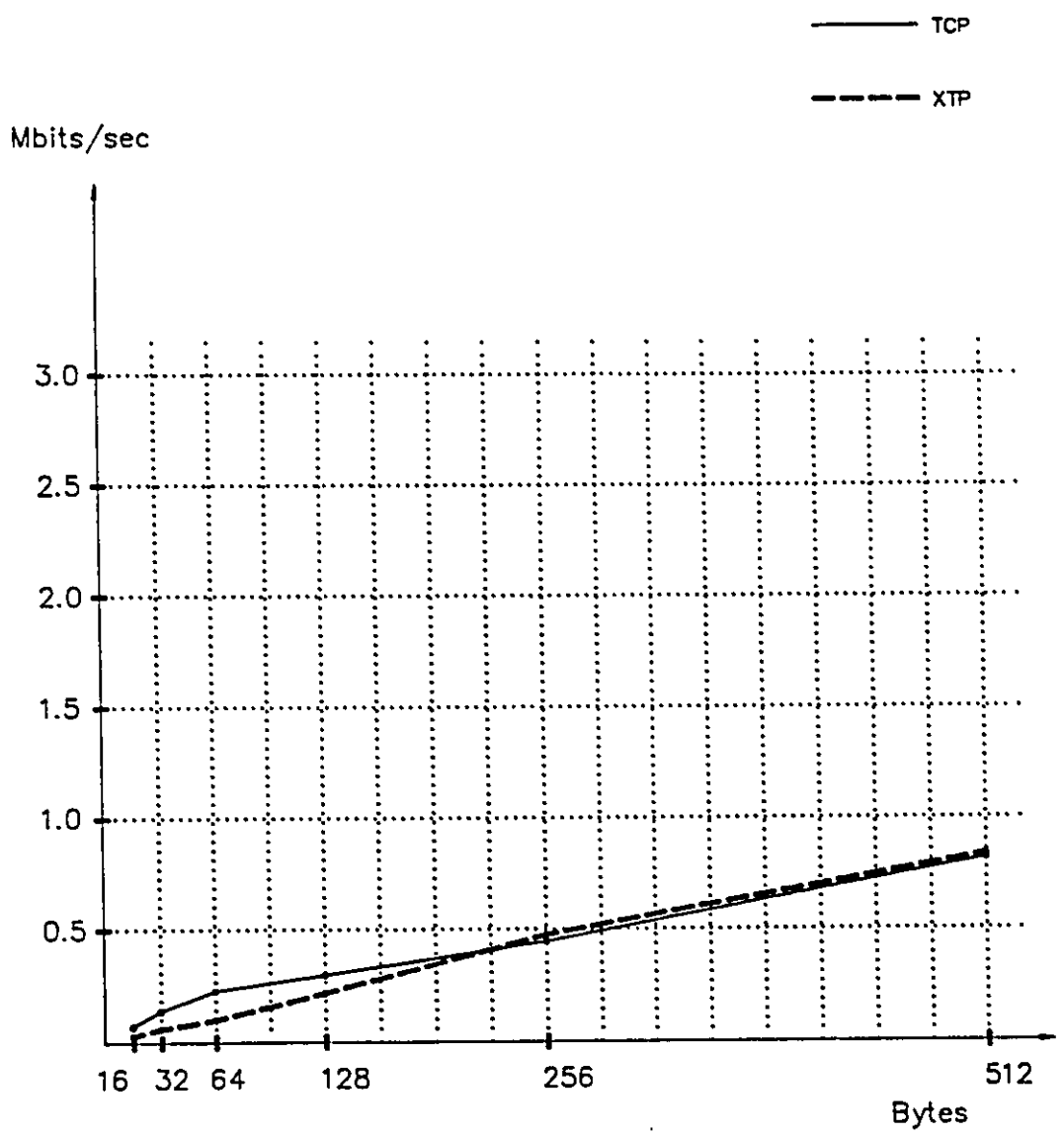

FIGURE 12

TROUGHPUT MEASUREMENTS
XTP,TCP RUNNING IN THE SAME TIME
SUN-3
MESSAGE SIZE : 16 B - 512 B

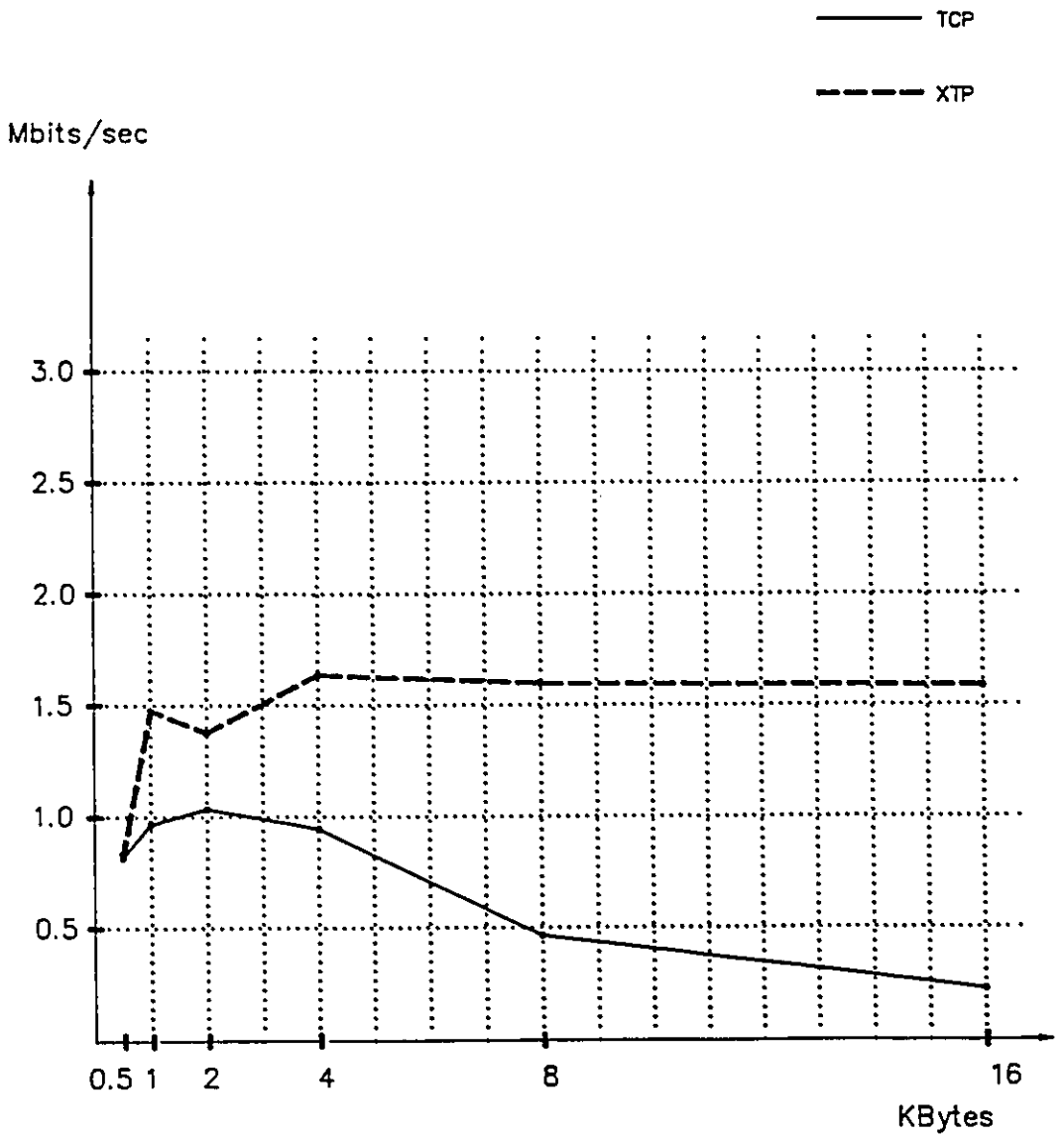

FIGURE 13

TROUGHPUT MEASUREMENTS
XTP,TCP RUNNING IN THE SAME TIME
SUN-3
MESSAGE SIZE: 512B-16KB

5.1.6 Delay Measurements for XTP

5.1.6.1 Socket Interface

For the XTP socket interface, measurements for three cases have been performed: the default, disable data checksum, and XTP input and output queues set to 32 KB instead of 16 KB.

The results are shown in Table 4 and Figures 14, 15, for the case of one virtual circuit.

The graph in Figure 14 plots the results for the XTP socket interface, in all three cases, for message size from 512 B to 16 KB. The best delay performance is in the case where the XTP input/output queues are 32 KB. The minimum delay is about 106 msec for the message size of 16 KB. In the case of data checksum disabled, for the message size up to 8 KB, the performance is slightly better than for the default case, which means that the checksum processing is quite performant. An anomaly can be observed for message sizes 16 KB, where the performance is better in the default case, than in the data checksum disabled case.

The graph in Figure 15 plots the results for the XTP socket interface, in all three cases, for message size from 16 B to 512 B. Although the checksum disabled case has a slightly better performance, all the three cases are similar. For message sizes between 16 B and 128 B the performance for the XTP I/O queues set to 32 KB case is worse.

Message size	XTP socket default	XTP socket checksum disabled	XTP socket i/o que 32 KB	XTP char default	XTP char checksum disabled	XTP char i/o que 32 KB
	Milliseconds					
16 B	7.19	6.92	8.69	7.36	7.09	10.31
32 B	5.7	5.45	6.49	5.86	5.6	7.42
64 B	5.59	5.14	5.59	5.76	5.26	6.12
128 B	5.81	5.72	5.88	6.03	5.55	6.18
256 B	6.54	6.58	6.57	6.7	6.14	6.54
512 B	7.5	7.02	7.42	7.62	6.74	7.11
1 KB	9.99	10.34	9.32	10.01	9.98	8.56
2 KB	20	18.14	18.27	20	15.89	16.01
4 KB	31.34	29.98	29.70	30.83	29.99	27.14
8 KB	55.54	46.08	55.56	55.73	47.08	52.83
16 KB	115.85	139.81	106.66	113.06	126.06	105.72

TABLE 4

DELAY MEASUREMENTS
XTP SOCKET AND CHARACTER DRIVER INTERFACES
SUN-3, ONE VIRTUAL CIRCUIT

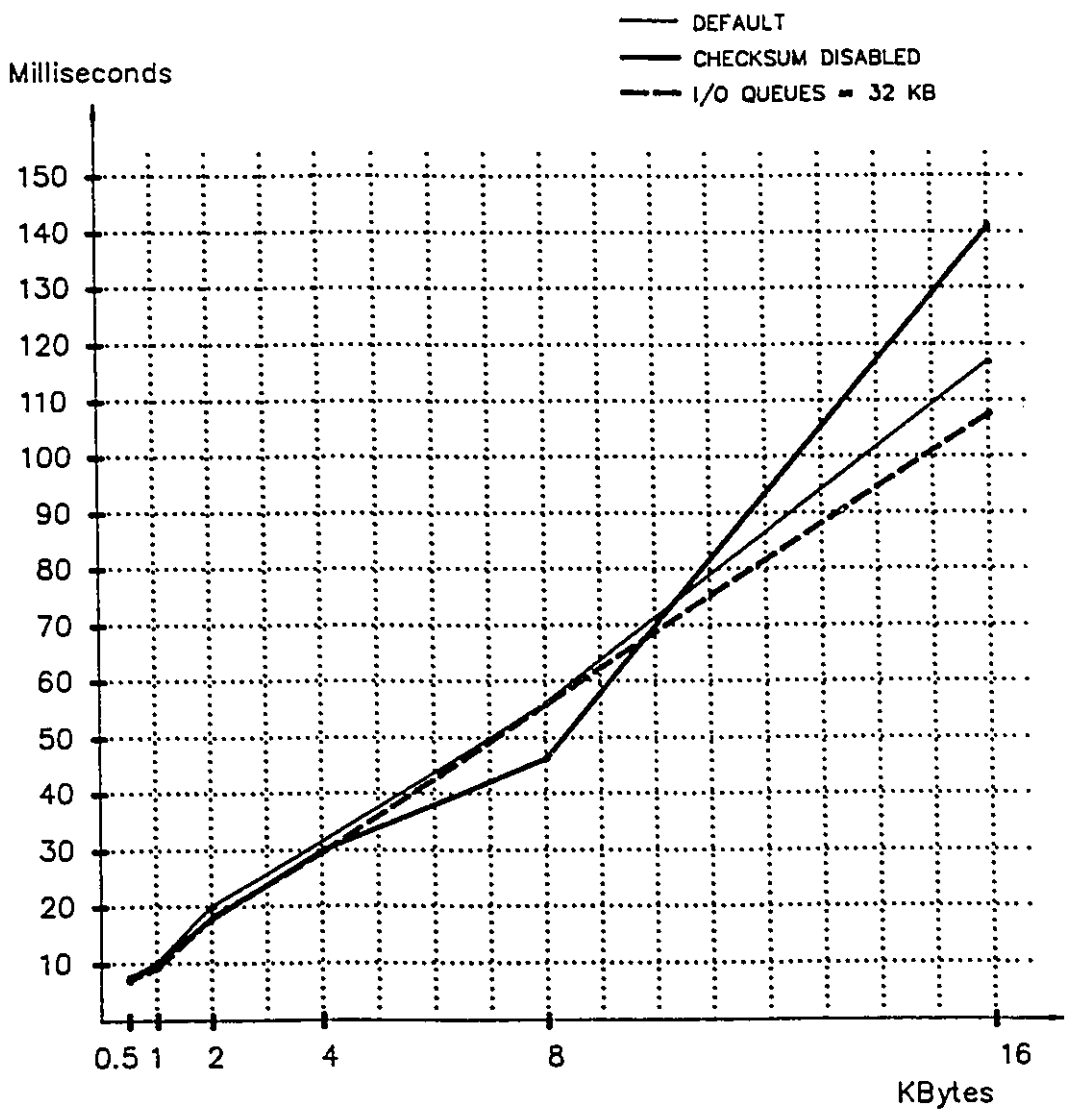

FIGURE 14

DELAY MEASUREMENTS
XTP SOCKET INTERFACE
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

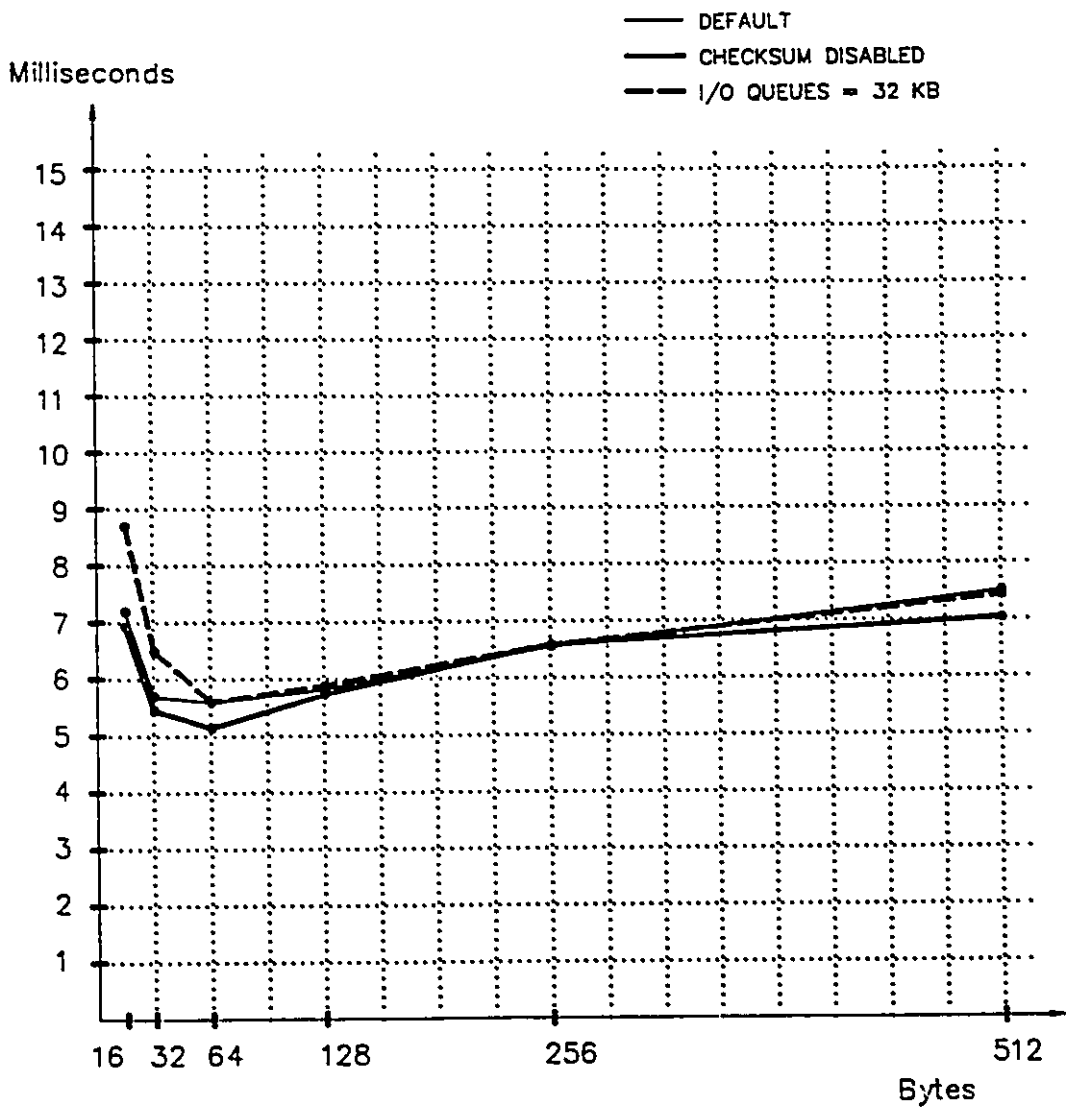

FIGURE 15

DELAY MEASUREMENTS
XTP SOCKET INTERFACE
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.1.6.2 Character Driver Interface

For the XTP character driver interface, measurements for the same three cases have been performed: the default, disable data checksum, and XTP input and output queues set to 32 KB instead of 16 KB. The results are shown in Table 4 and Figures 16, 17, for the case of one virtual circuit.

The same comments for the socket interface apply here.

A general comment applies to both interfaces. The delay stays low between 5 to 9 msec for message sizes between 16 B and 512 B and then increases abruptly and linearly to about 140 msec for message sizes from 512 B to 16 KB.

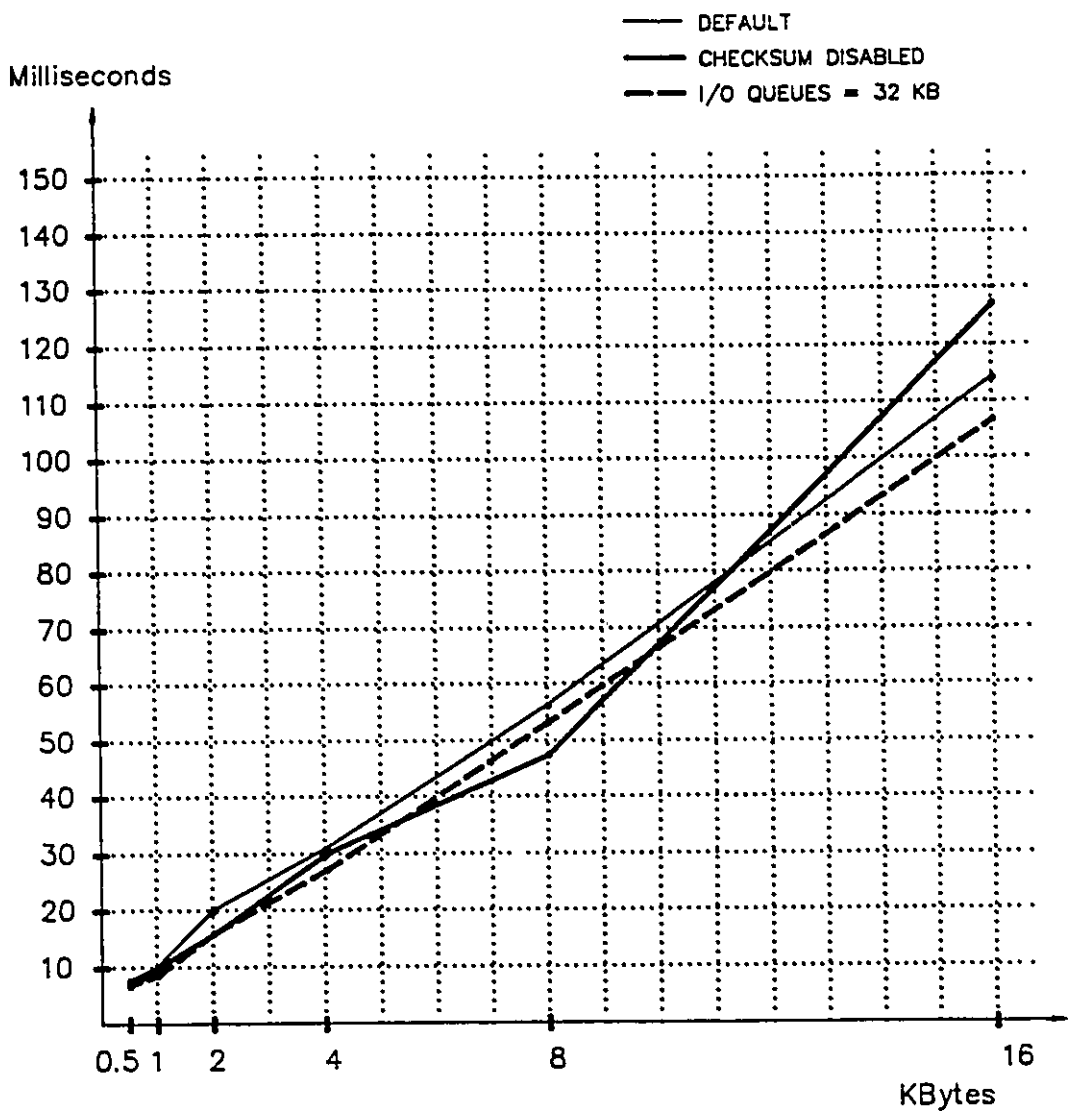

FIGURE 16

DELAY MEASUREMENTS
XTP CHARACTER DRIVER INTERFACE
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

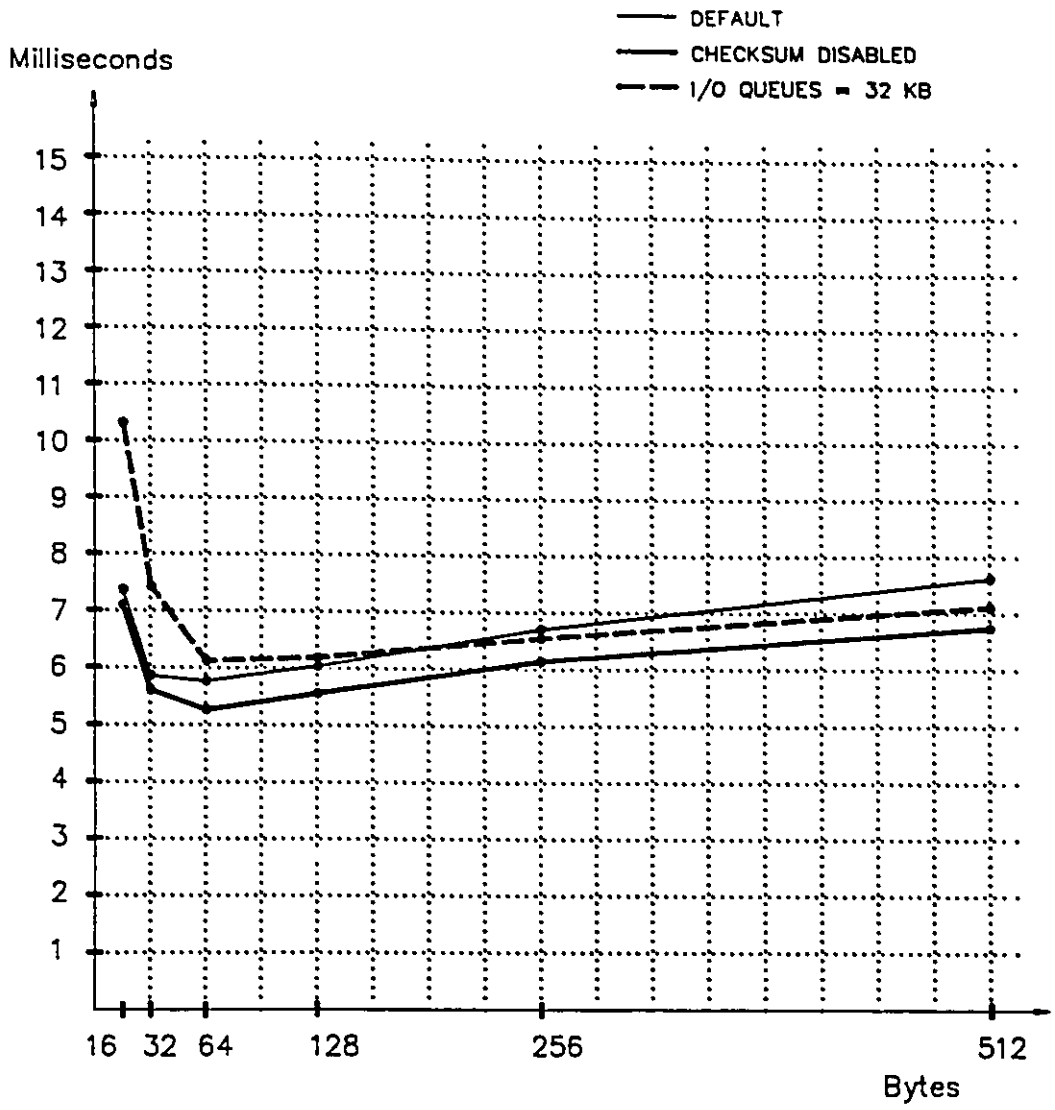

FIGURE 17

DELAY MEASUREMENTS
XTP CHARACTER DRIVER INTERFACE
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.1.7 Delay Measurements for TCP

The delay measurements for TCP, for one virtual circuit are shown in Table 5 and Figures 18 and 19. The delay obtained for message size 16 KB is about 120 msec.

5.1.8 Delay Measurements for UDP

The delay measurements for UDP, for one receiver and one sender are shown in Table 5 and Figures 18 and 19. The delay obtained for message size 8 KB is 52.16 msec.

It is worth mentioning that no messages are lost, as opposed to the throughput measurements, because for the delay experiments the time to send a message and receive the same message back is measured.

Message size	TCP	UDP	XTP socket i/o que 32 KB
	Milliseconds		
16 B	1.48	not measured	8.69
32 B	1.63	not measured	6.49
64 B	1.95	not measured	5.59
128 B	2.99	not measured	5.88
256 B	4.05	not measured	6.57
512 B	4.7	9.45	7.42
1 KB	8.39	10.94	9.32
2 KB	15.79	17.82	18.27
4 KB	30.73	28.20	29.70
8 KB	61.75	52.16	55.56
16 KB	119.94	—	106.66

TABLE 5

DELAY MEASUREMENTS
XTP,TCP,UDP
SUN-3, ONE VIRTUAL CIRCUIT

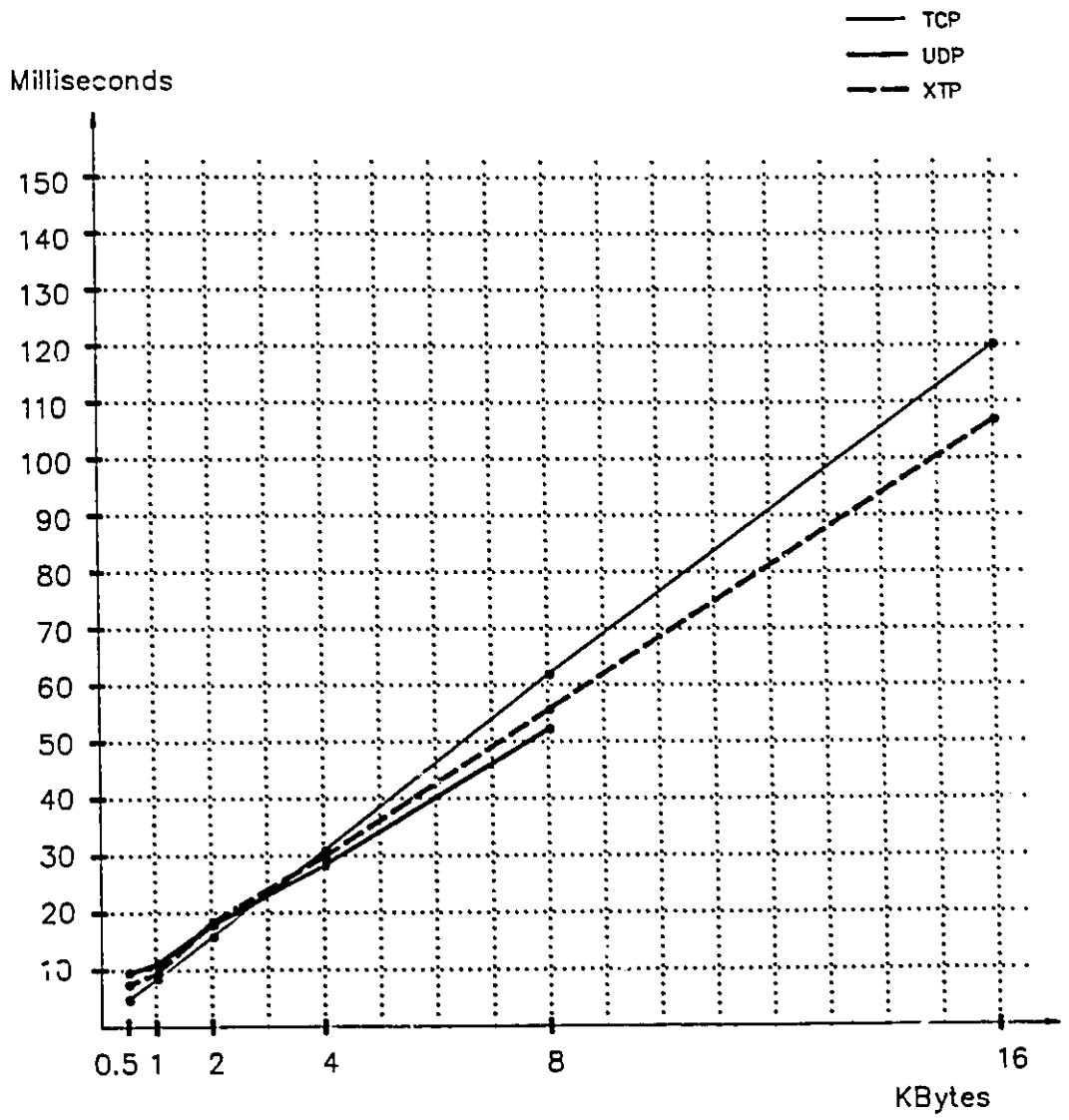

FIGURE 18

DELAY MEASUREMENTS
XTP,TCP,UDP
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

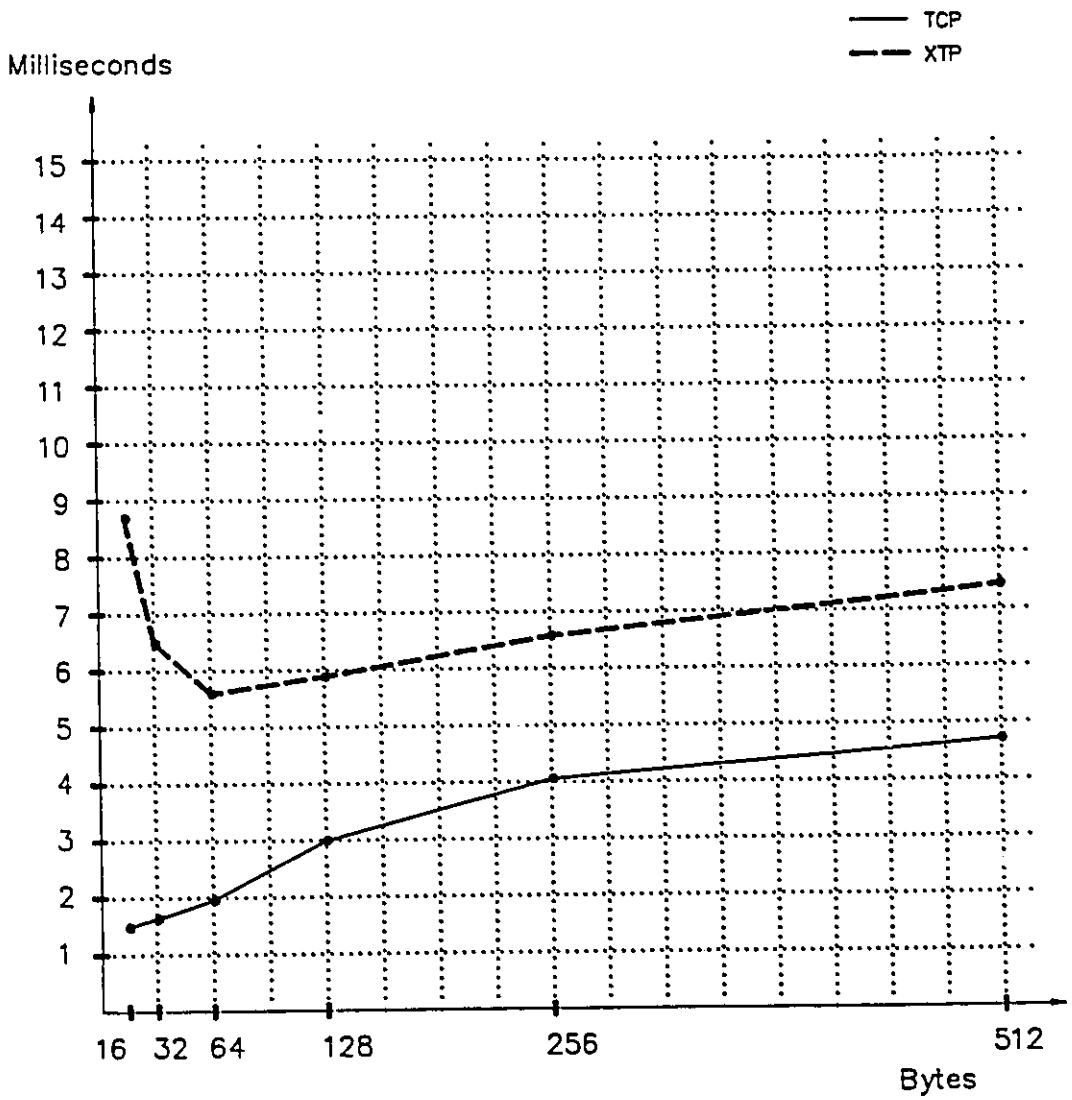

FIGURE 19

DELAY MEASUREMENTS
XTP,TCP
SUN-3, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.1.9 Delay Comparisons

The performance of the two XTP interfaces, the socket and character driver, is very similar.

For comparison with the other protocols, the XTP socket interface, the case where the XTP input/output queues are 32 KB, is used.

The Table 5 and Figures 18 and 19, show the results for XTP, TCP, UDP.

The UDP shows better performance, than XTP or TCP, for message sizes 4 KB and 8 KB.

The XTP performance is better than TCP, for message sizes from 4 KB to 16 KB, and worse for message sizes 16 B to 4 KB.

Although these results are for the XTP socket interface, the case where the XTP input/output queues are 32 KB, the default case also proves better for the message sizes from 8 KB to 16 KB.

Another set of tests have been performed to be able to compare the performance of XTP and TCP when transfers take place on two virtual circuits, in the same time. The results are shown in Table 6 and Figures 20, 21, 22, 23.

When two TCP virtual circuits are running at the same time, the delay for a message of 16 KB, increases by about 70%.

When two XTP virtual circuits are running at the same time, the delay for a message of 16 KB, increases by about 103%.

When one TCP virtual circuits is transferring data in the same time with an XTP virtual circuit, the delay increases 110% for TCP and 30% for XTP.

Message size	TCP 1 circuit	TCP 2 circuits	XTP socket i/o que 32 KB 1 circuit	XTP socket i/o que 32 KB 2 circuits	TCP with XTP	XTP socket with TCP
	Milliseconds					
16 B	1.48	3.01	8.69	not measured	4.15	8.06
32 B	1.63	3.31	6.49	not measured	4.85	7.14
64 B	1.95	3.95	5.59	11.3	5.13	8.00
128 B	2.99	6.12	5.88	12.15	8.59	8.41
256 B	4.05	8.15	6.57	13.06	10.31	10.49
512 B	4.7	9.16	7.42	14.08	10.13	15.04
1 KB	8.39	16.08	9.32	15.59	13.76	16.86
2 KB	15.79	30.02	18.27	31.43	29.37	30.15
4 KB	30.73	53.98	29.70	52.15	51.85	47.98
8 KB	61.75	106.42	55.56	101.80	106.84	76.61
16 KB	119.94	203.87	106.66	217.19	252.09	139.22

TABLE 6

DELAY MEASUREMENTS
XTP,TCP
SUN-3 TWO VIRTUAL CIRCUITS
VS. ONE VIRTUAL CIRCUIT

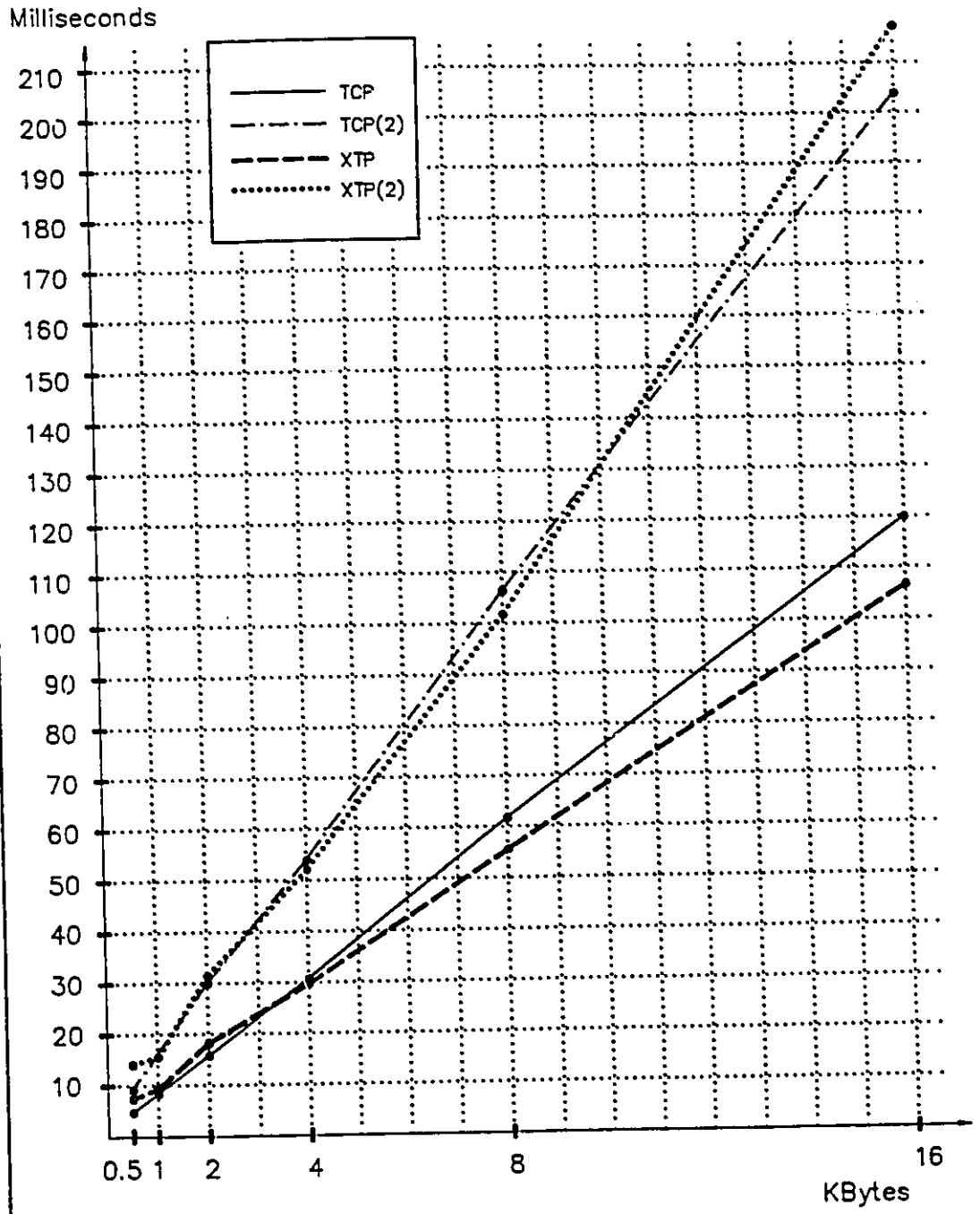

FIGURE 20

DELAY MEASUREMENTS
XTP,TCP
SUN-3, ONE VIRTUAL CIRCUIT
VS.TWO VIRTUAL CIRCUITS
MESSAGE SIZE : 512 B - 16 KB

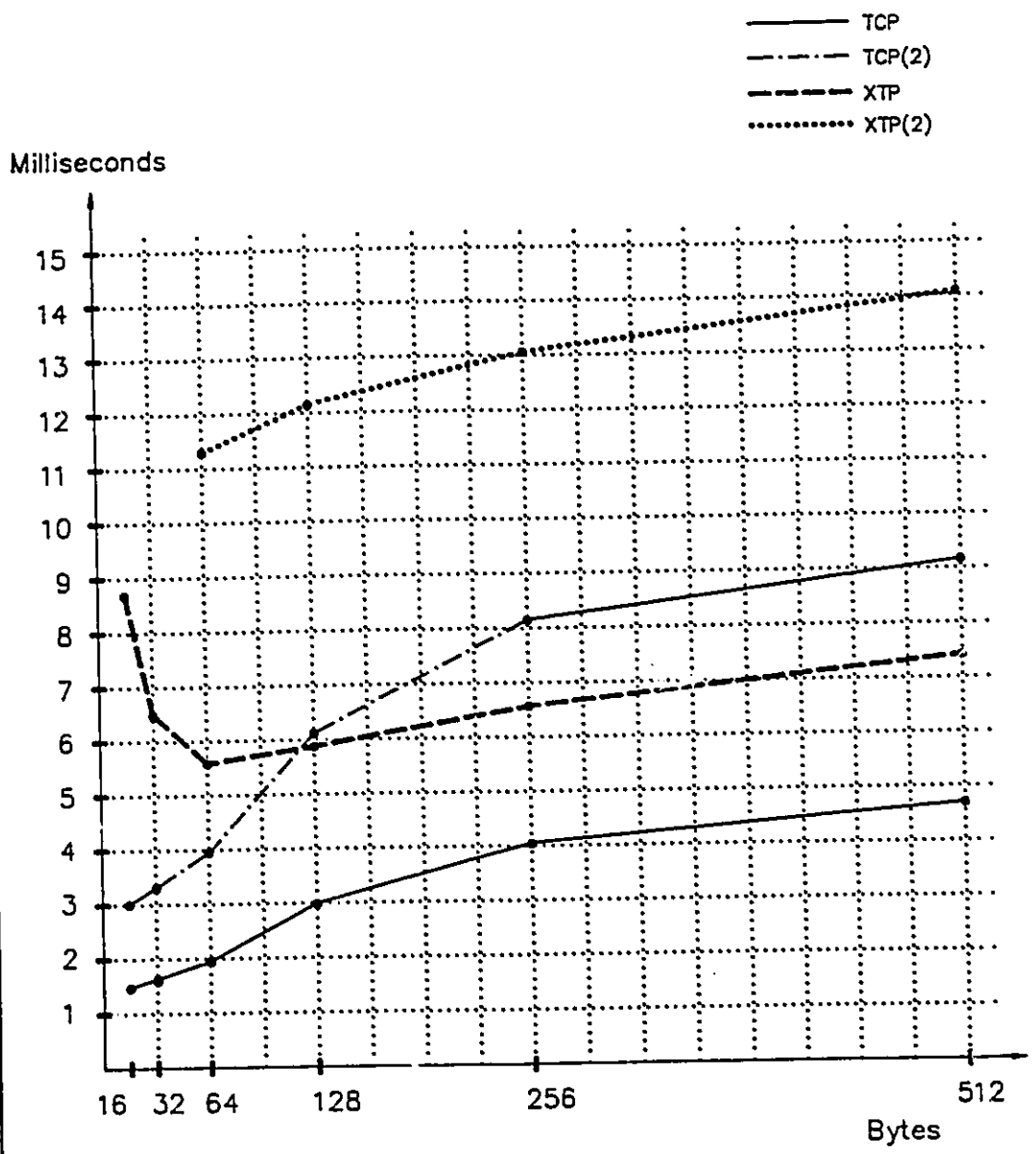

FIGURE 21

DELAY MEASUREMENTS
XTP,TCP
SUN-3, ONE VIRTUAL CIRCUIT
VS.TWO VIRTUAL CIRCUITS
MESSAGE SIZE : 16 B - 512 B

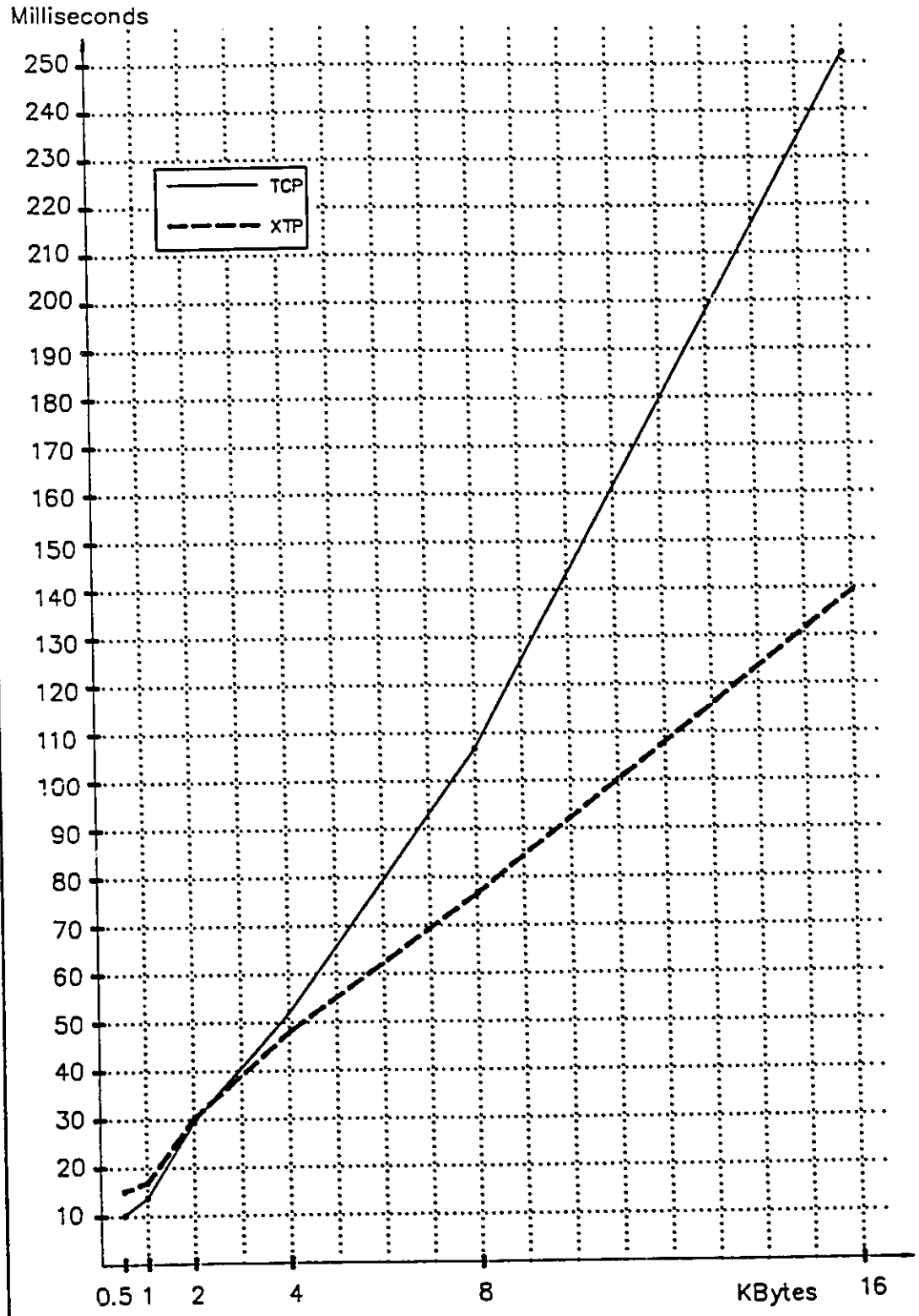

FIGURE 22

DELAY MEASUREMENTS XTP,TCP RUNNING IN THE SAME TIME
SUN-3 MESSAGE SIZE: 512B-16KB

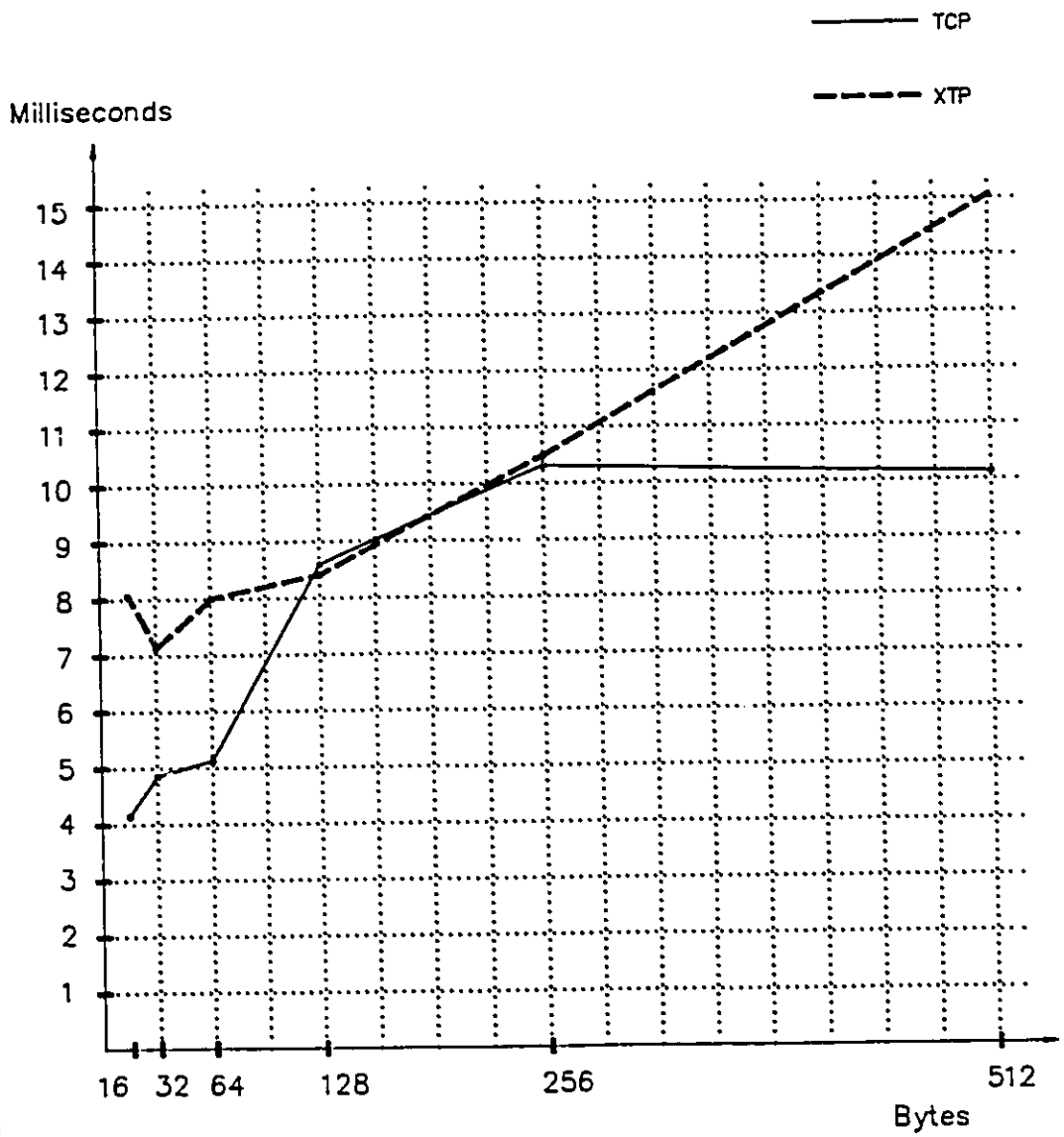

FIGURE 23

DELAY MEASUREMENTS
XTP,TCP RUNNING IN THE SAME TIME
SUN-3
MESSAGE SIZE : 16 B - 512 B

5.2 Measurements for the Sun-4 Configuration

5.2.1 Throughput Measurements for XTP

5.2.1.1 Socket Interface

For the XTP socket interface, measurements for three cases have been performed: the default, disable data checksum, and XTP input and output queues set to 32 KB instead of 16 KB.

The results are shown in Table 7 and Figures 24, 25, for the case of one virtual circuit.

The graph in Figure 24 plots the results for the XTP socket interface, in all three cases, for message size from 512 B to 16 KB. The best throughput performance is for the case where the XTP input/output queues are 32 KB. The maximum throughput is 6.79 Mbps for the message size of 16 KB. That means that maximum 67.9% of the 10 Mbps raw bandwidth can be delivered to the user.

In the case of data checksum disabled, the performance is very similar to the default case, which means that the checksum processing is quite performant. It is worthwhile mentioning that the anomaly observed for the Sun-3 configuration does not appear for this configuration.

The graph in Figure 25 plots the results for the XTP socket interface, in all three cases, for message size from 16 B to 512 B. Although the checksum disabled and XTP I/O queues set to 32 KB cases have a slightly better performance, all the three cases are similar, which confirms the fact that the checksum processing is performant and indicates that for message sizes between 16 B and 512 B it is not worth to

increase the XTP input/output queues from 16 KB to 32 KB, mainly if we are in a configuration short of physical memory.

Message size	XTP socket default	XTP socket checksum disabled	XTP socket i/o que 32 KB	XTP char default	XTP char checksum disabled	XTP char i/o que 32 KB
	Mb / sec					
16 B	0.12	0.13	0.08	0.12	0.12	0.08
32 B	0.04	0.34	0.25	0.31	0.32	0.24
64 B	0.69	0.73	0.61	0.65	0.69	0.60
128 B	1.11	1.22	1.11	1.11	1.11	1.08
256 B	2.00	2.00	2.17	2.00	2.00	2.00
512 B	2.95	2.94	3.35	2.94	2.94	3.34
1 KB	3.84	3.84	5.51	3.84	3.84	5.49
2 KB	3.84	5.51	5.55	3.86	5.50	5.53
4 KB	5.53	5.52	5.60	5.5	5.53	5.57
8 KB	5.55	5.54	6.74	5.56	5.54	6.74
16 KB	5.53	5.55	6.79	5.51	5.54	6.69

TABLE 7

TROUGHPUT MEASUREMENTS
XTP SOCKET AND CHARACTER DRIVER INTERFACES
SUN-4, ONE VIRTUAL CIRCUIT

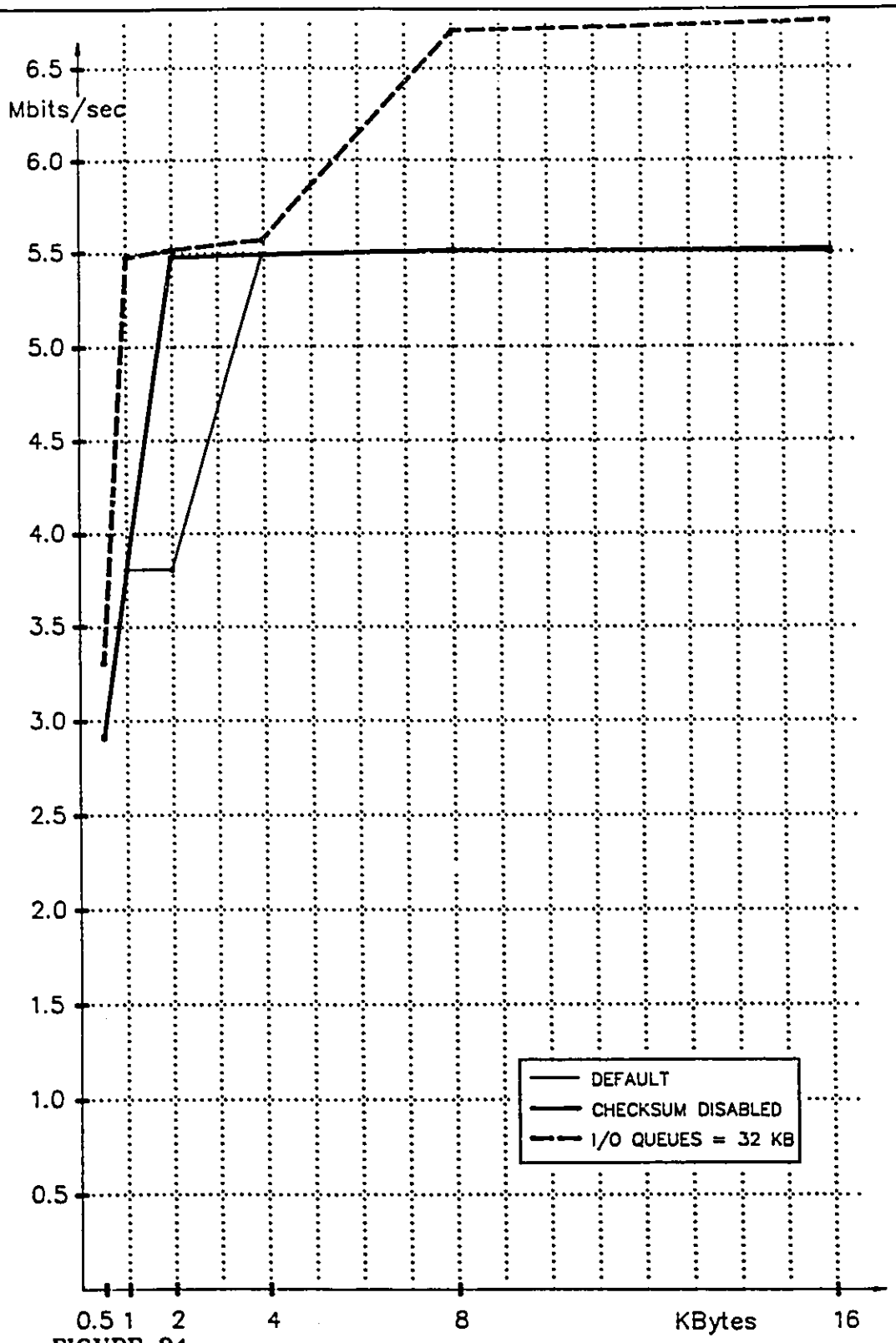

FIGURE 24

TROUGHPUT MEASUREMENTS SUN-4, ONE VIRTUAL CIRCUIT

XTP SOCKET INTERFACE MESSAGE SIZE : 512 B - 16 KB

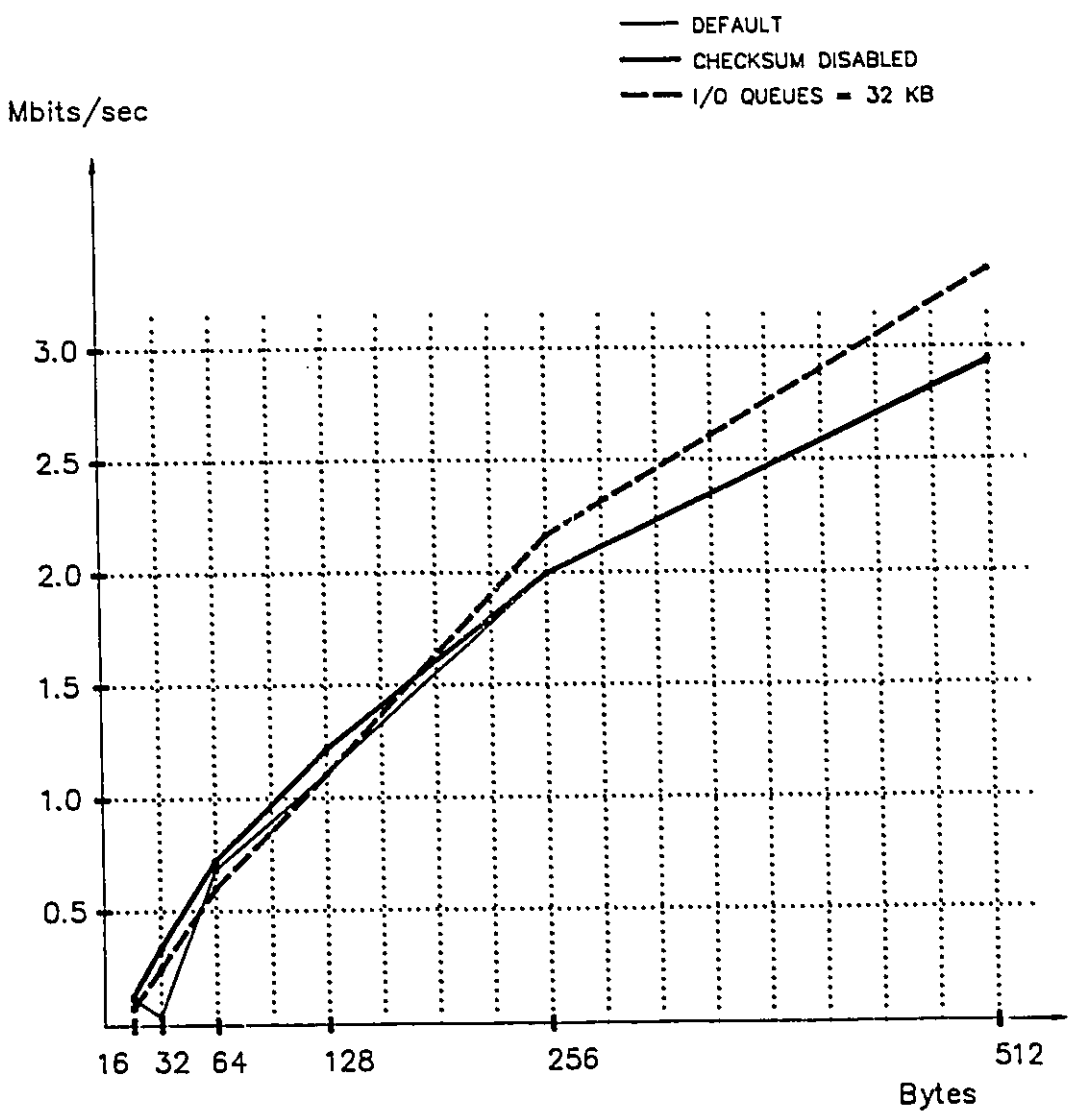

FIGURE 25

TROUGHPUT MEASUREMENTS
XTP SOCKET INTERFACE
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.2.1.2 Character Driver Interface

For the XTP character driver interface, measurements for the same three cases have been performed: the default, disable data checksum, and XTP input and output queues set to 32 KB instead of 16 KB. The results are shown in Table 7 and Figures 26, 27, for the case of one virtual circuit.

The same comments as for the socket interface apply to the character interface.

A general comment applies to both interfaces. The throughput increases abruptly and linearly to about 5.5 Mbps for message sizes up to about 1 KB, and continues only with a slighter increase up to 6.79 Mbps for message sizes up to 16 KB.

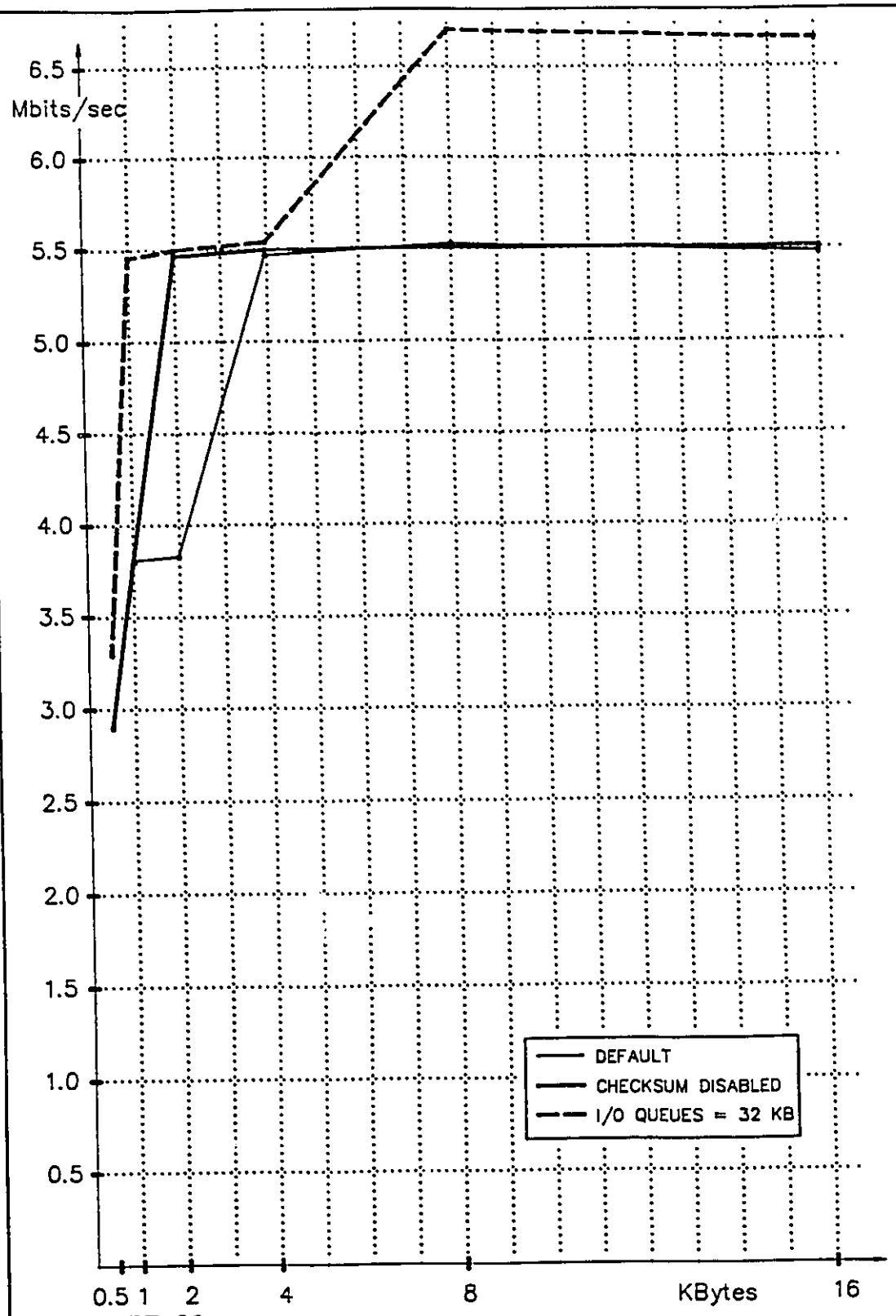

FIGURE 26
TROUGHPUT MEASUREMENTS SUN-4, ONE VIRTUAL CIRCUIT
XTP CHARACTER DRIVER INTERFACE MESSAGE SIZE: 512B-16KB

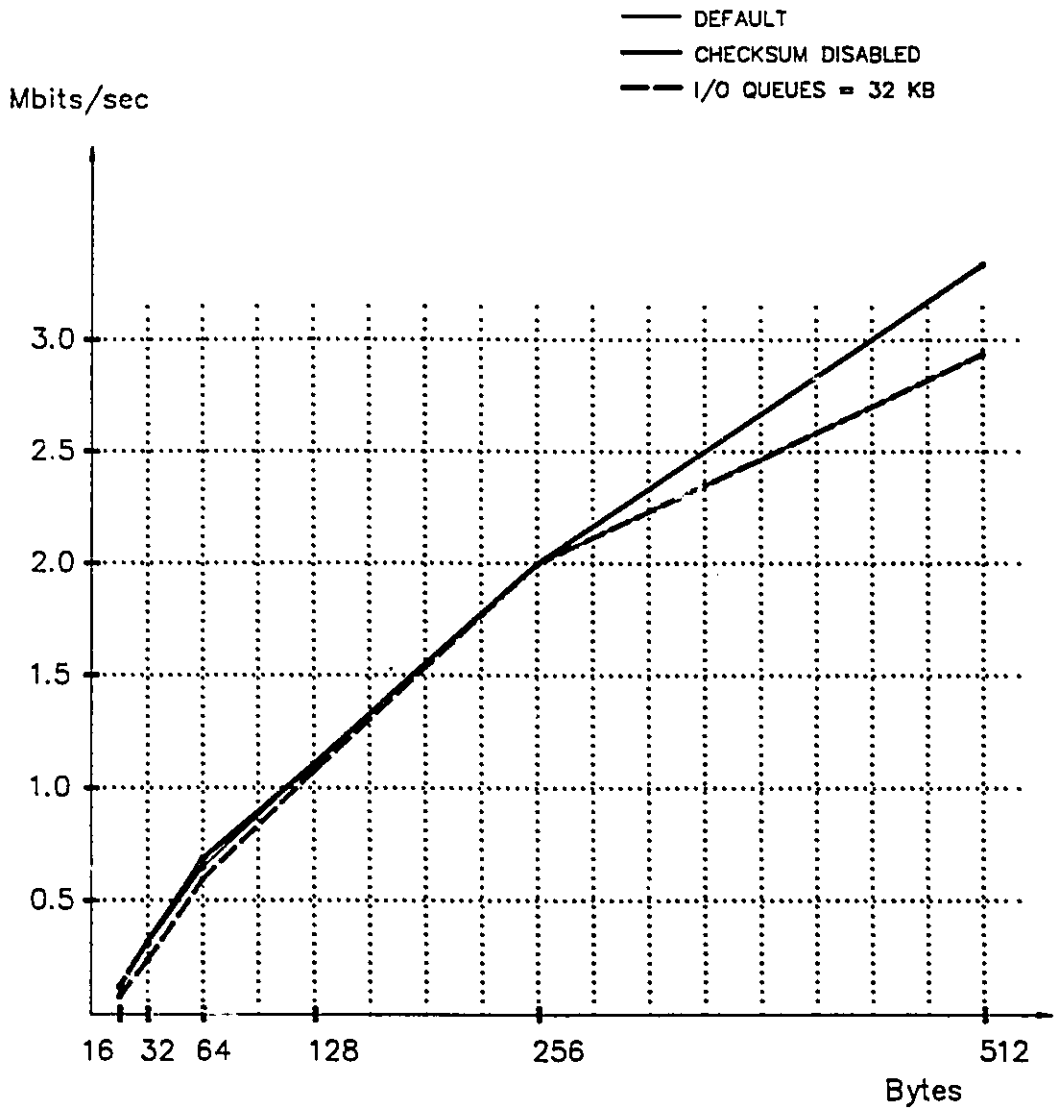

FIGURE 27

TROUGHPUT MEASUREMENTS
XTP CHARACTER DRIVER INTERFACE
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.2.2 Throughput Measurements for TCP

The throughput measurements for TCP, for one virtual circuit are shown in Table 8 and Figures 28 and 29. The maximum throughput obtained is 4.81 Mbps for message size 1 KB. That means that maximum 48.1% of the 10 Mbps raw bandwidth can be delivered to the user.

For the message size 16 KB, the throughput is 4.49 Mbps.

5.2.3 Throughput Measurements for UDP

The throughput measurements for UDP, for one receiver and one sender are shown in Table 8 and Figures 28 and 29. The maximum throughput obtained is 16.92 Mbps for message size 8 KB. But in this case, at the receiver side 96.88% of the messages are lost. In order for the receiver to get most of the messages, the transmitter has to be slowed down. The subchapter 5.3 describes the measurements of the receiver's throughput versus the offered load (transmitter's throughput). The offered load of 16.92 Mbps has been obtained when the transmitter sends as fast as it can. When delays are added between every message, lower offered loads can be obtained. At an offered load of 4.92 Mbps, the receiver throughput is 4.96 Mbps and no messages are lost. That means that maximum 49.6% of the 10 Mbps raw bandwidth can be delivered to the user, without losing messages. Depending on the requirements of an application using UDP, the offered load could be increased, in order to obtain a better receiver throughput, and keep an acceptable percentage of messages lost.

5.2.4 Throughput Measurements for the RAW Network Driver

The throughput measurements for the RAW Network Driver are shown in Table 8 and Figures 28 and 29.

The maximum throughput obtained is 8.55 Mbps for message size 1 KB. That means that maximum 85.5% of the 10 Mbps raw bandwidth can be delivered to the user. In this case not all the messages will be received, since the RAW Network Driver does not implement any error or flow control, and the interface returns to the user before making sure that the message has been delivered.

Message size	TCP	UDP	RAW	XTP socket I/o que 32 KB
	Mb / sec			
16 B	0.58	0.28	0.25	0.08
32 B	1.00	0.56	0.43	0.25
64 B	1.78	1.03	0.85	0.61
128 B	2.52	1.90	1.80	1.11
256 B	3.51	3.32	3.18	2.17
512 B	4.41	6.07	4.88	3.35
1 KB	4.81	9.85	5.55	5.51
2 KB	4.72	12.72	—	5.55
4 KB	4.68	15.40	—	5.60
8 KB	4.48	16.92	—	6.74
16 KB	4.49	—	—	6.79

TABLE 8

TROUGHPUT MEASUREMENTS
XTP,TCP,UDP,RAW
SUN 4, ONE VIRTUAL CIRCUIT

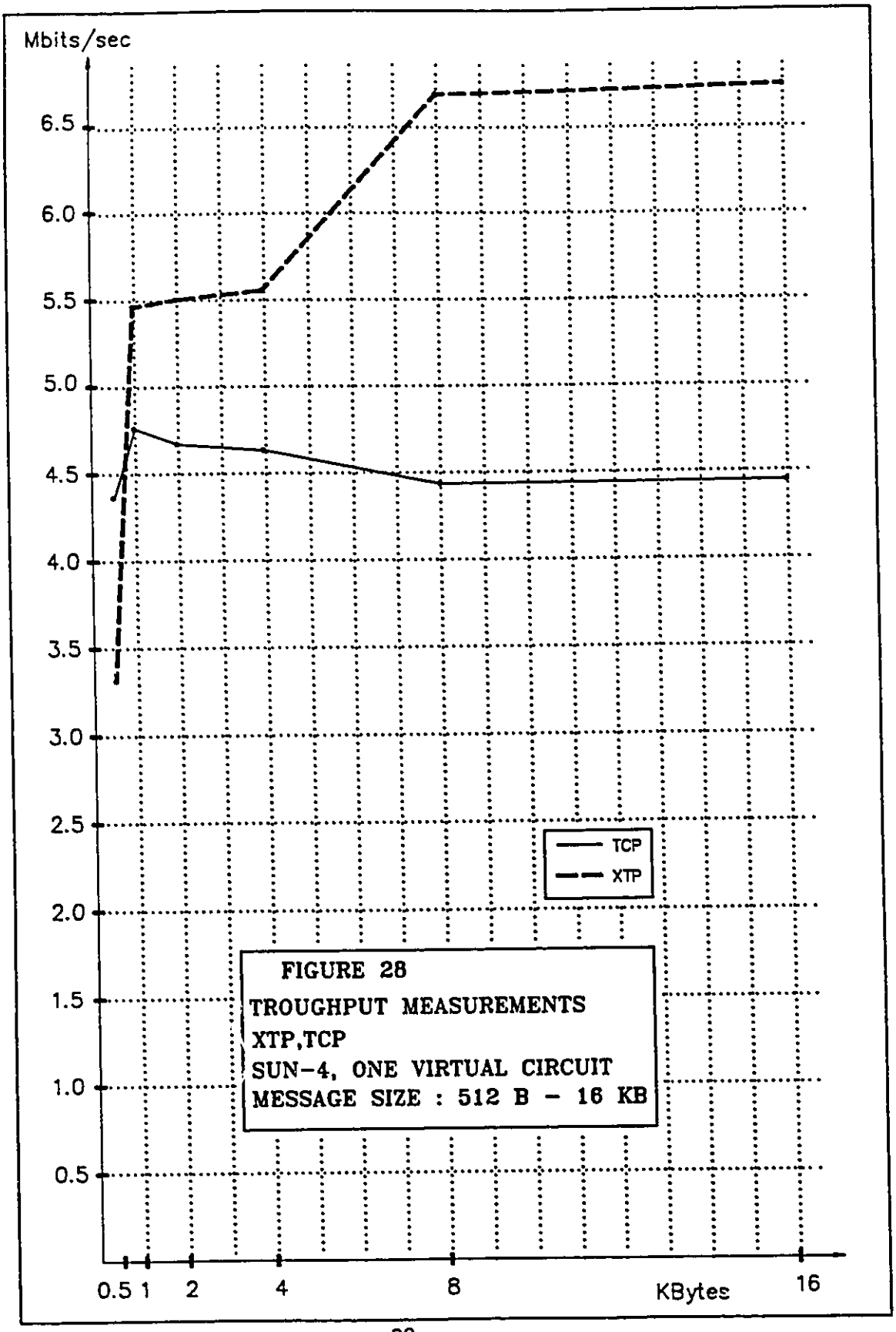

FIGURE 28
TROUGHPUT MEASUREMENTS
XTP,TCP
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

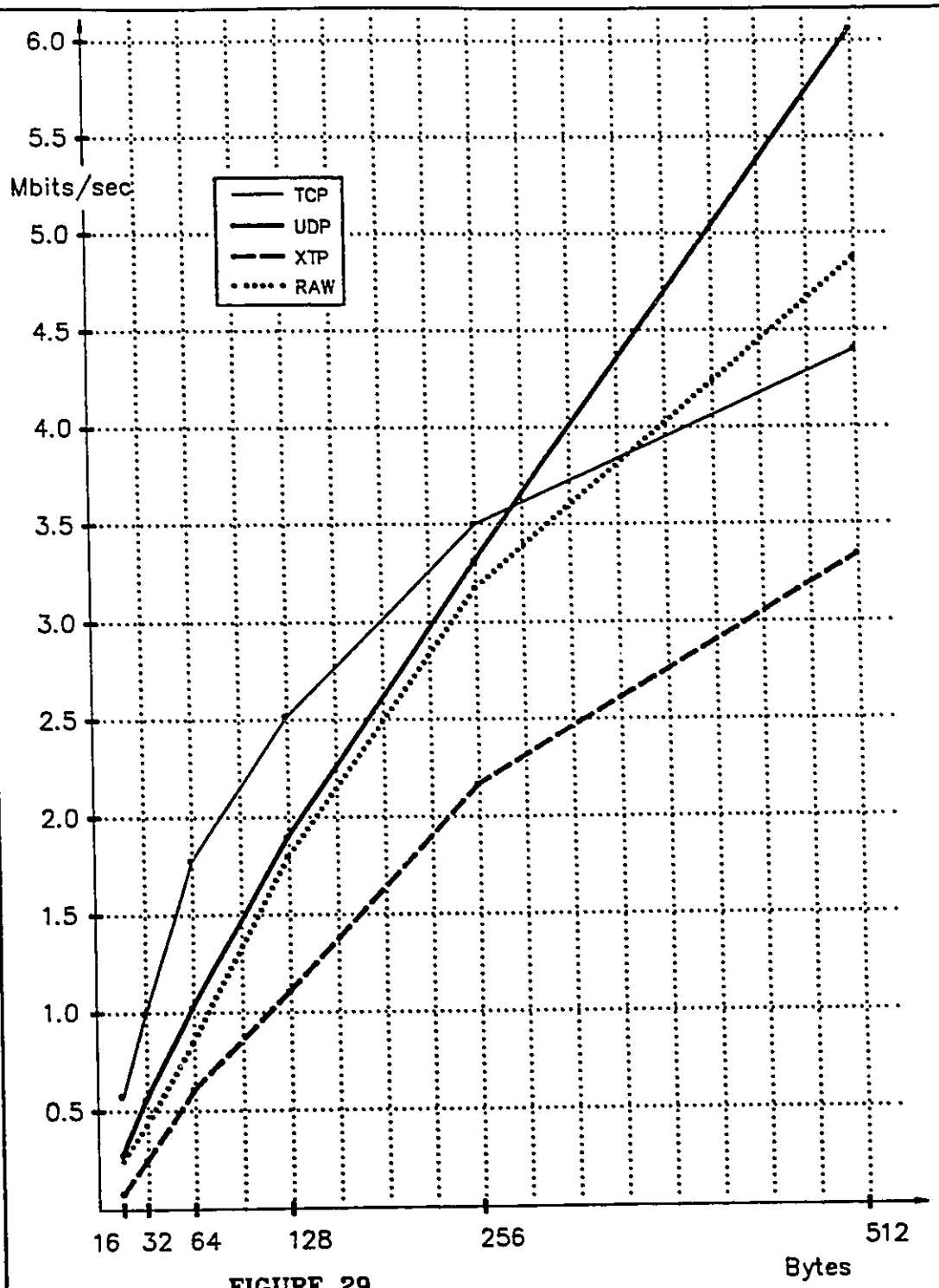

FIGURE 29

TROUGHPUT MEASUREMENTS
XTP,TCP,UDP,RAW
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.2.5 Throughput Comparisons

The performance of two XTP interfaces, the socket and character driver, is very similar, for all message sizes. The technique used by the character driver interface is different for short data buffers (copy) from larger buffers (lock down into memory). However, in the case of the sun-4 configuration, this does not generate any significant performance difference.

For comparison with the other protocols, the XTP socket interface, the case where the XTP input/output queues are 32 KB, is used.

The Table 8 and Figures 28 and 29, show the results for XTP, TCP, UDP and the RAW Network Driver.

Although the UDP shows better performance, for message sizes from 256 B to 16 KB, this is the case where most of the messages are lost. In the case where a low percentage of messages are lost, UDP's performance is better than TCP, and worse than XTP, for message size 8 KB.

The XTP performance is better than TCP, for message sizes from 1 KB to 16 KB, and worse for message sizes 16 B to 1 KB.

The RAW Network driver performance is better than that of XTP and TCP for the message sizes (512 B - 1 KB), and worse than TCP and better than XTP for message sizes (16 B - 512 B).

The following list summarizes the results for the maximum throughput performance for all the protocols in the study. From the maximum 10 Mbps of the Ethernet raw bandwidth the maximum percentage throughput delivered to the user, for each protocol is shown:

- 85.5% RAW Network Driver for 1 KB message size (unreliable transfer),

- 67.90% XTP socket interface for 16 KB message size,
- 66.90% XTP character interface for 16 KB message size,
- 49.6% UDP for 8 KB message size (no messages lost),
- 48.1% TCP for 16 KB message size.

Therefore, XTP achieves the best throughput performance, for message sizes between 1 KB and 16 KB, for reliable transfers.

Although these results are for the XTP socket interface, the case where the XTP input/output queues are 32 KB, the default case also proves better for the message sizes from 4 KB to 16 KB.

Another set of tests have been performed to be able to compare the performance of XTP and TCP when transfers take place on two virtual circuits, in the same time. The results are shown in Table 9 and Figures 30, 31, 32, 33.

When two TCP virtual circuits are running in the same time, the maximum throughput percentage of the 10 Mbps raw bandwidth drops from 44.9% to 27.1% (for message size 16 KB).

When two XTP virtual circuits are running in the same time, the maximum throughput percentage of the 10 Mbps raw bandwidth drops from 67.9% to 28.8%.

When one TCP virtual circuits is transferring data in the same time with an XTP virtual circuit, the maximum throughput percentage of the 10 Mbps raw bandwidth drops from 44.9% to 20.5% for TCP, and from 67.9% to 56.6% for XTP.

Message size	TCP 1 circuit	TCP 2 circuits	XTP socket i/o que 32 KB 1 circuit	XTP socket i/o que 32 KB 2 circuits	TCP with XTP	XTP socket with TCP
	Mb / sec					
16 B	0.58	0.29	0.08	0.08	0.28	0.08
32 B	1.00	0.53	0.25	0.25	0.50	0.21
64 B	1.78	0.89	0.61	0.40	0.86	0.40
128 B	2.52	1.24	1.11	0.65	1.13	0.64
256 B	3.51	1.77	2.17	1.23	1.62	1.09
512 B	4.41	2.59	3.35	2.27	2.99	2.16
1 KB	4.81	2.49	5.51	3.44	3.28	4.43
2 KB	4.72	2.76	5.55	3.42	3.41	5.26
4 KB	4.68	2.77	5.60	3.46	3.26	5.45
8 KB	4.48	2.72	6.74	3.36	2.94	5.52
16 KB	4.49	2.71	6.79	2.88	2.05	5.66

TABLE 9

TROUGHPUT MEASUREMENTS
XTP,TCP
SUN-4 TWO VIRTUAL CIRCUITS
VS. ONE VIRTUAL CIRCUIT

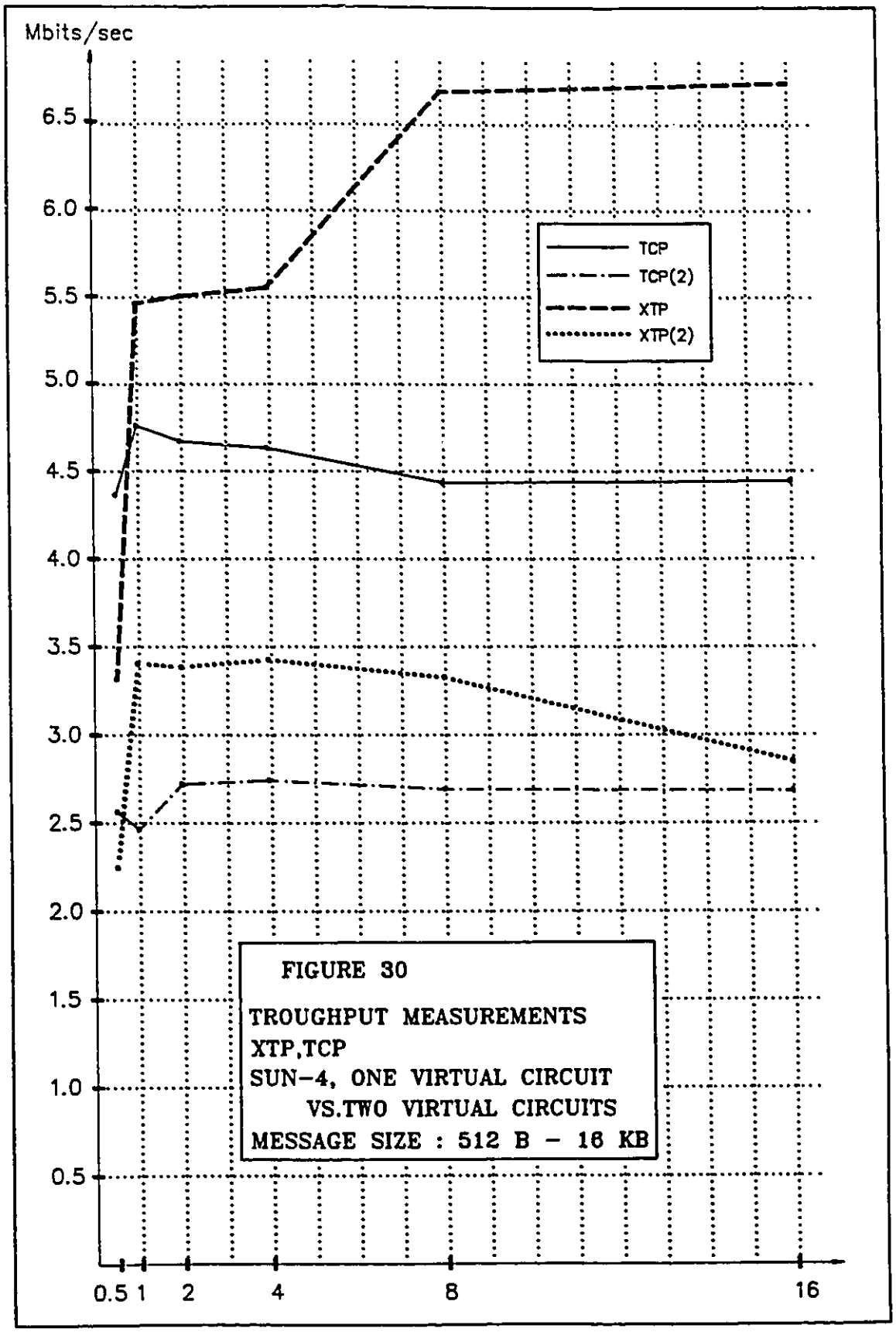

FIGURE 30

TROUGHPUT MEASUREMENTS
XTP,TCP
SUN-4, ONE VIRTUAL CIRCUIT
VS.TWO VIRTUAL CIRCUITS
MESSAGE SIZE : 512 B - 16 KB

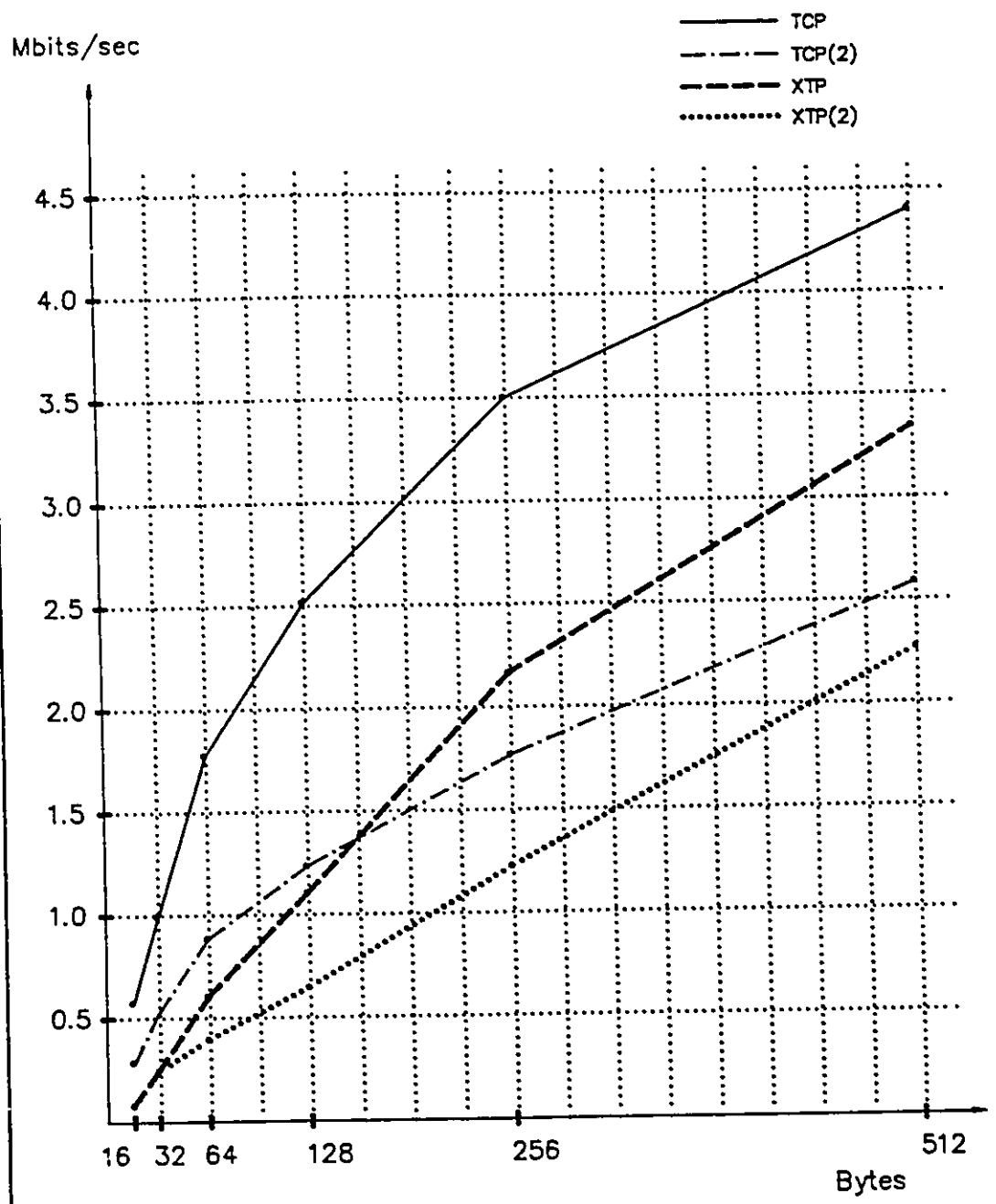

FIGURE 31

TROUGHPUT MEASUREMENTS
XTP,TCP
SUN-4, ONE VIRTUAL CIRCUIT
VS.TWO VIRTUAL CIRCUITS
MESSAGE SIZE : 16 B - 512 B

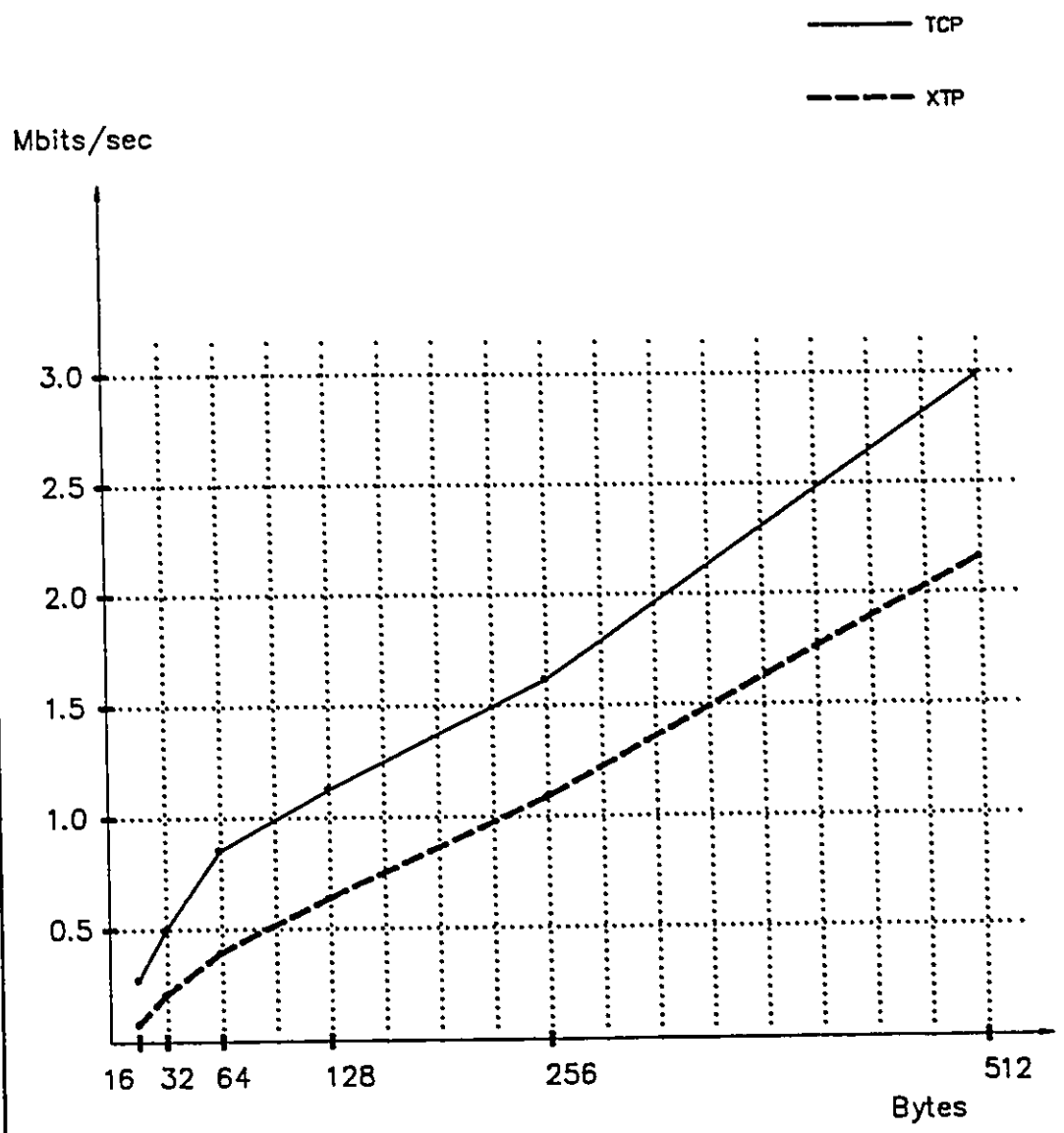

FIGURE 32

TROUGHPUT MEASUREMENTS
XTP,TCP RUNNING IN THE SAME TIME
SUN-4
MESSAGE SIZE : 16 B - 512 B

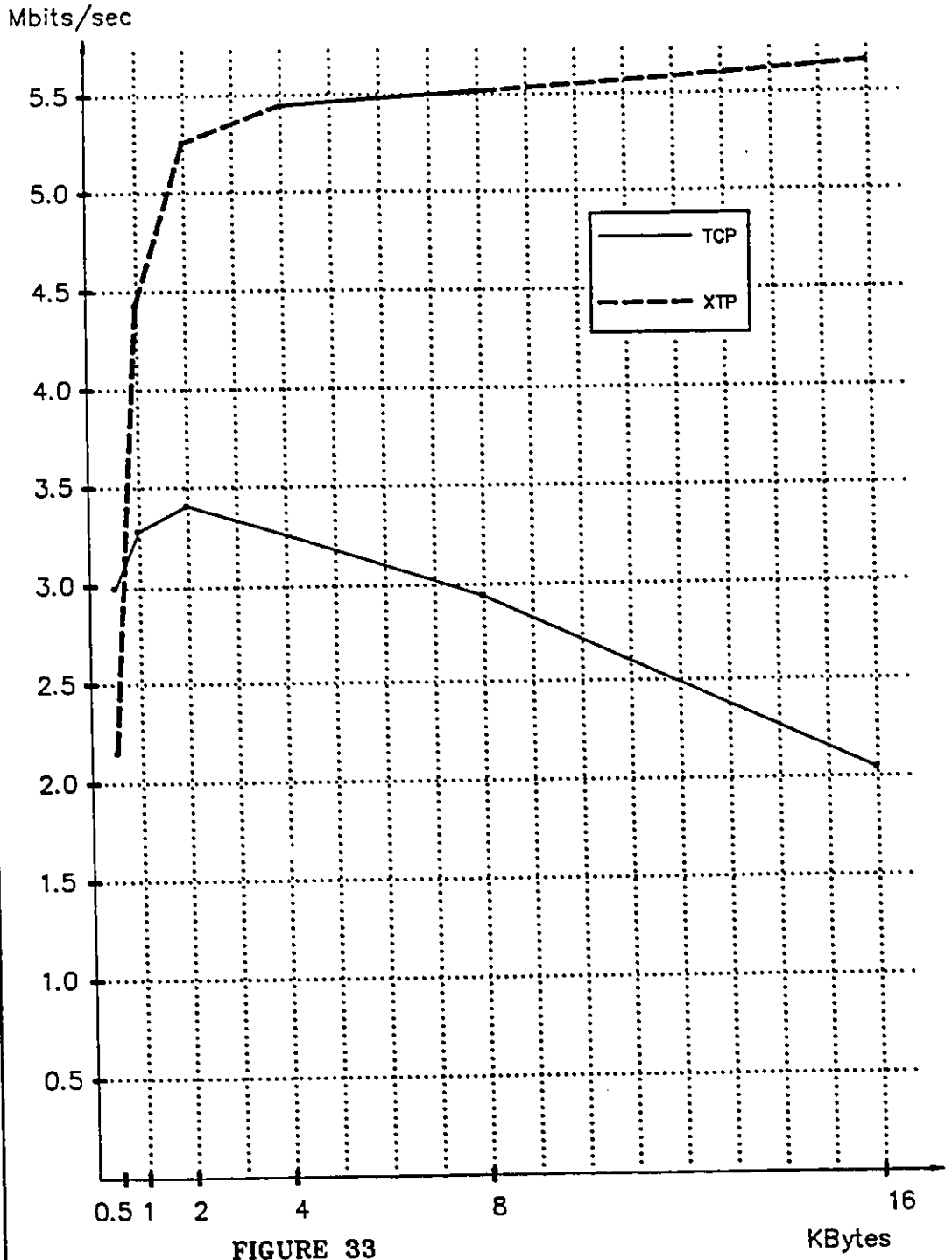

FIGURE 33

TROUGHPUT MEASUREMENTS
XTP,TCP RUNNING IN THE SAME TIME
SUN-4
MESSAGE SIZE: 512B-16KB

5.2.6 Delay Measurements for XTP

5.2.6.1 Socket Interface

For the XTP socket interface, measurements for three cases have been performed: the default, disable data checksum, and XTP input and output queues set to 32 KB instead of 16 KB.

The results are shown in Table 10 and Figures 34, 35, for the case of one virtual circuit.

The graph in Figure 34 plots the results for the XTP socket interface, in all three cases, for message size from 512 B to 16 KB. Although the best delay performance is in the case where the checksum is disabled, all three case have very similar performance. The minimum delay for the message size of 16 KB is about 47.19 msec.

The graph in Figure 35 plots the results for the XTP socket interface, in all three cases, for message size from 16 B to 512 B. Although the checksum disabled case has a slightly better performance, all the three cases are similar.

Message size	XTP socket default	XTP socket checksum disabled	XTP socket i/o que 32 KB	XTP char default	XTP char checksum disabled	XTP char i/o que 32 KB
	Milliseconds					
16 B	1.76	1.37	1.65	1.82	1.75	2.78
32 B	1.31	1.29	1.53	1.36	1.31	1.83
64 B	1.33	1.21	1.38	1.37	1.27	1.45
128 B	1.51	1.40	1.47	1.52	1.41	1.53
256 B	1.72	1.59	1.63	1.75	1.61	1.77
512 B	2.24	2.24	1.94	2.25	2.24	1.99
1 KB	3.24	3.24	3.69	3.26	3.97	2.97
2 KB	6.83	5.26	6.83	6.48	6.71	5.45
4 KB	11.23	10.49	13.10	10.52	13.58	13.42
8 KB	21.01	22.16	26.71	22.49	20.96	26.35
16 KB	49.55	47.19	51.31	42.20	42.06	46.87

TABLE 10

DELAY MEASUREMENTS
XTP SOCKET AND CHARACTER DRIVER INTERFACES
SUN 4, ONE VIRTUAL CIRCUIT

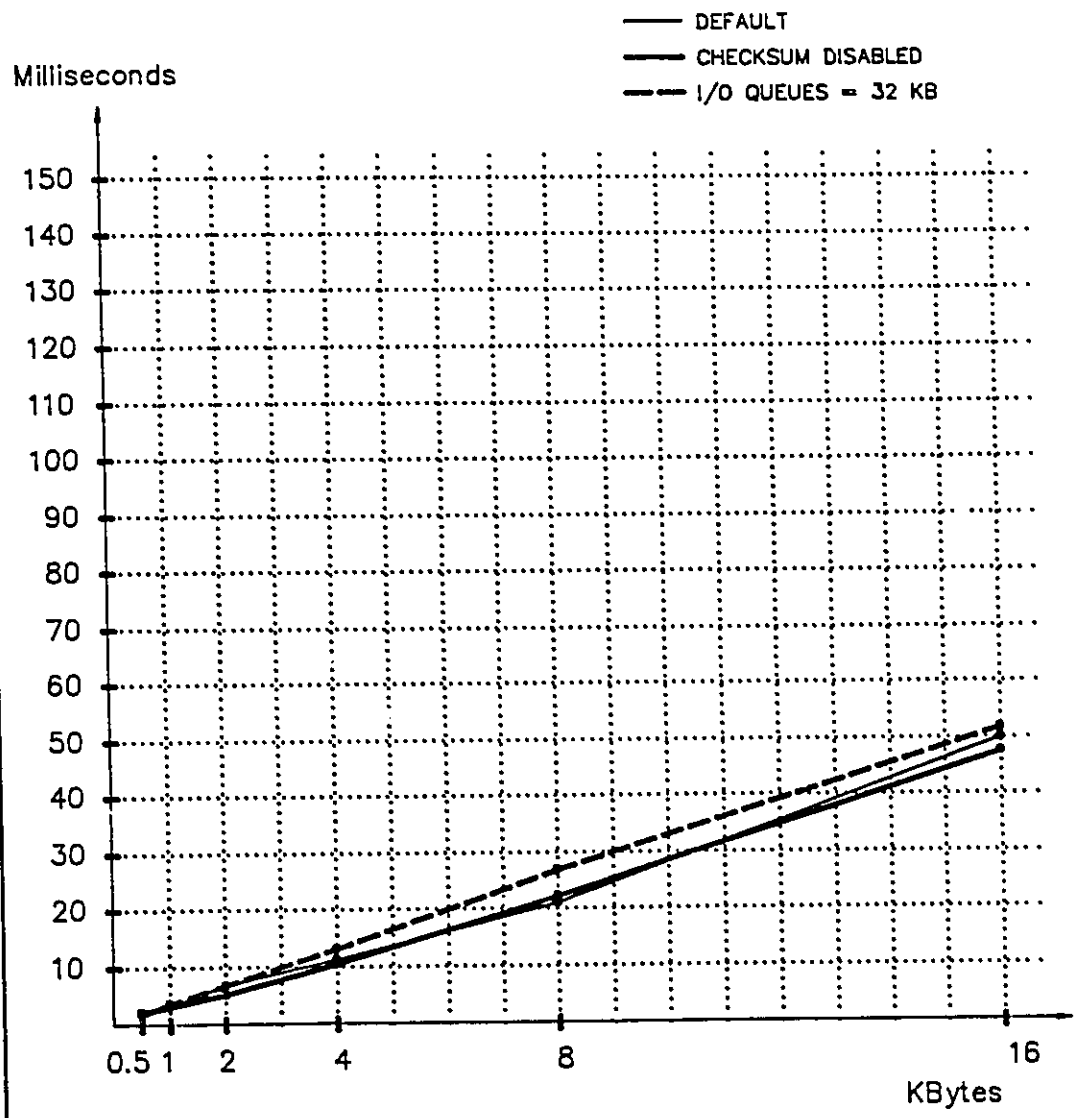

FIGURE 34

DELAY MEASUREMENTS
XTP SOCKET INTERFACE
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

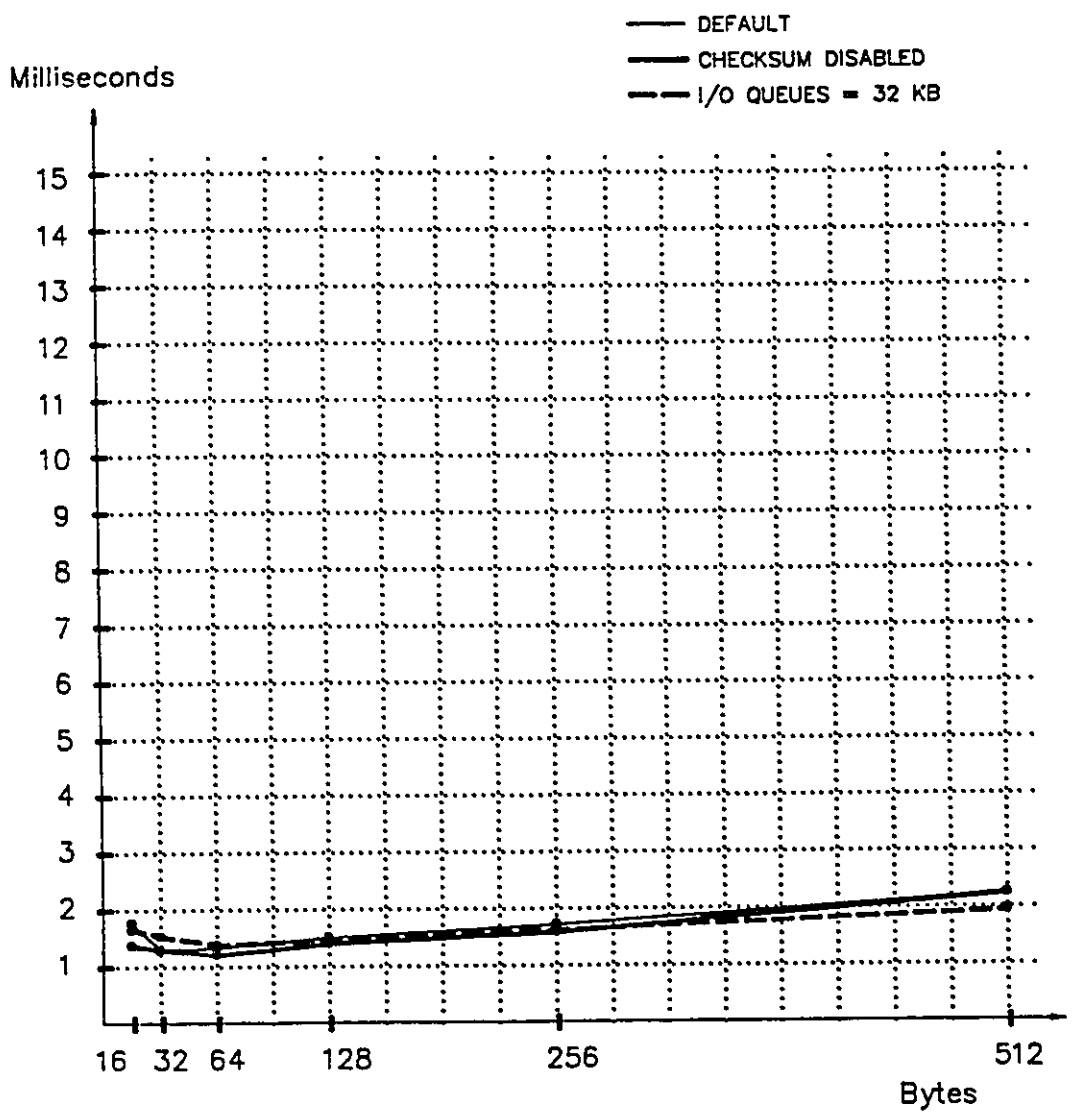

FIGURE 35

DELAY MEASUREMENTS
XTP SOCKET INTERFACE
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.2.6.2 Character Driver Interface

For the XTP character driver interface, measurements for the same three cases have been performed: the default, disable data checksum, and XTP input and output queues set to 32 KB instead of 16 KB. The results are shown in Table 10 and Figures 36, 37, for the case of one virtual circuit.

The same comments as for the socket interface apply here.

A general comment applies to both interfaces. The delay stays low between 1 to 3 msec for message sizes between 16 B and 512 B and then increases abruptly and linearly to about 50 msec for message sizes from 512 B to 16 KB.

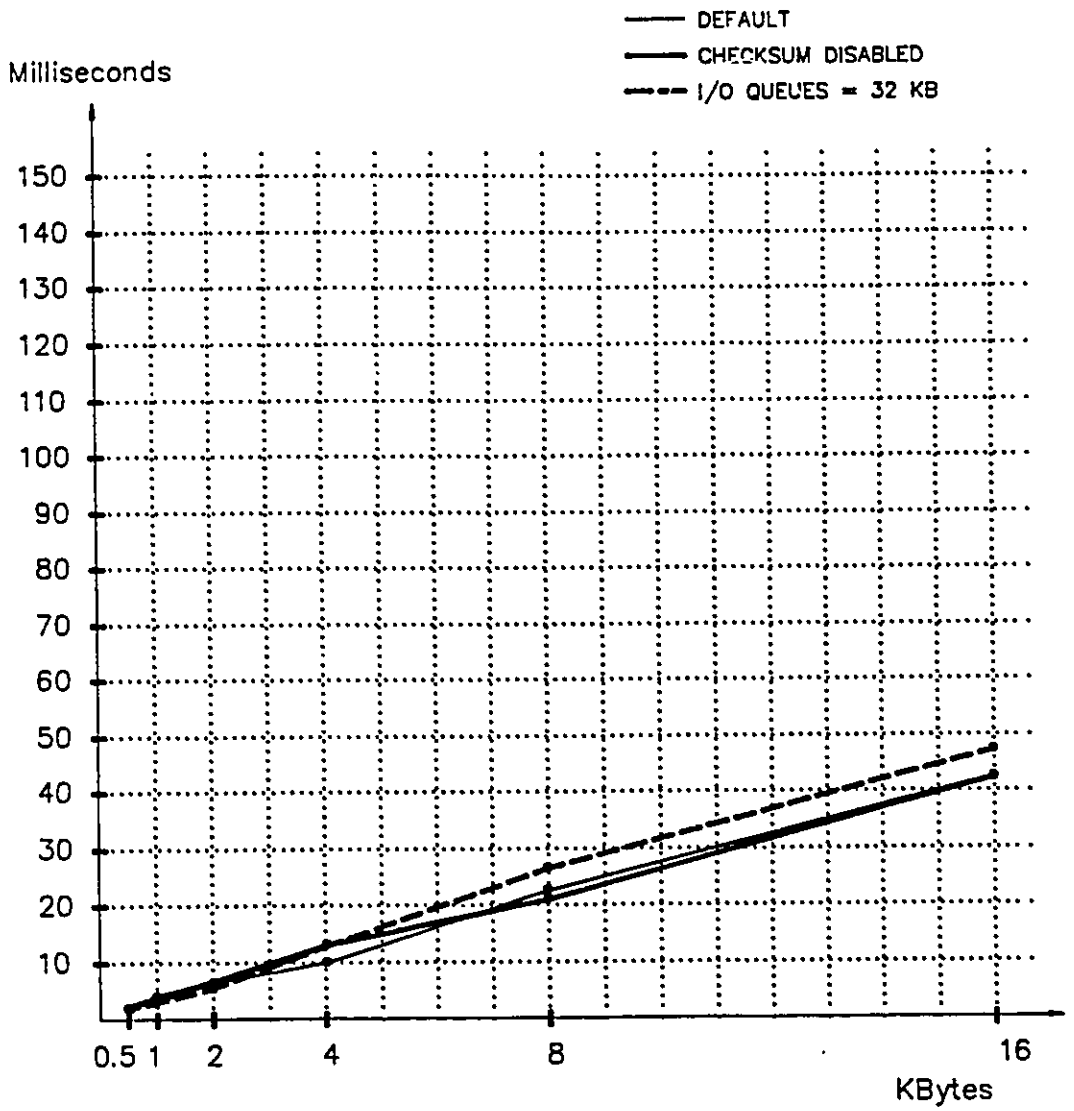

FIGURE 36

DELAY MEASUREMENTS
XTP CHARACTER DRIVER INTERFACE
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

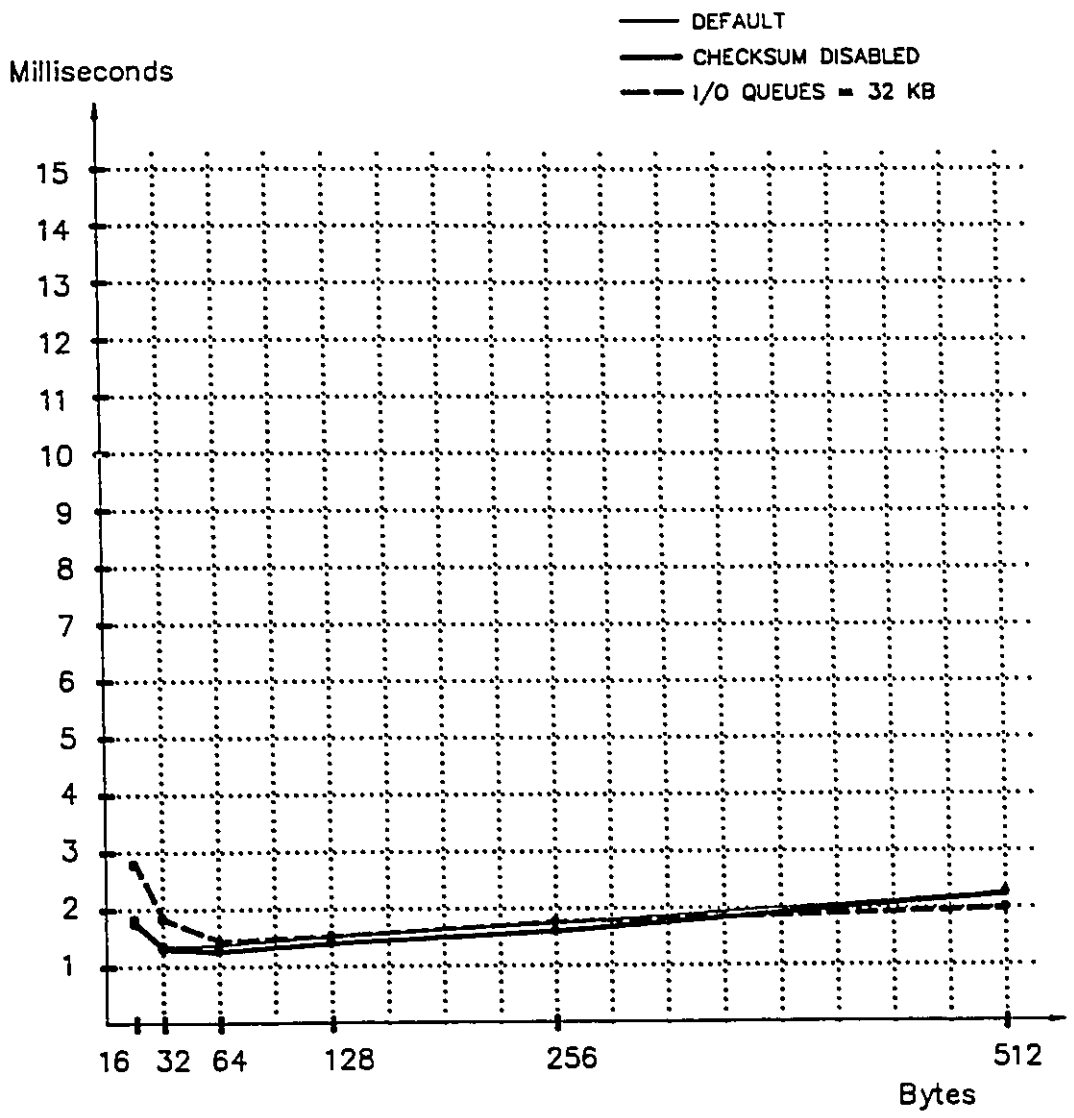

FIGURE 37

DELAY MEASUREMENTS
XTP CHARACTER DRIVER INTERFACE
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.2.7 Delay Measurements for TCP

The delay measurements for TCP, for one virtual circuit are shown in Table 11 and Figures 38 and 39.

The delay obtained for message size 16 KB is about 48.2 msec.

5.2.8 Delay Measurements for UDP

The delay measurements for UDP, for one receiver and one sender are shown in Table 11 and Figures 38 and 39. The delay obtained for message size 8 KB is 21.23 msec.

It is worth mentioning that no messages are lost, as opposed to the throughput measurements, because for the delay experiments the time to send a message and receive the same message back is measured.

Message size	TCP	UDP	XTP socket i/o que 32 KB
	Milliseconds		
16 B	0.38	0.84	1.65
32 B	0.42	1.66	1.53
64 B	0.48	1.82	1.38
128 B	0.64	2.01	1.47
256 B	1.01	2.32	1.63
512 B	1.65	2.87	1.94
1 KB	3.11	4.13	3.69
2 KB	6.14	6.61	6.83
4 KB	12.20	11.50	13.10
8 KB	24.03	21.23	26.71
16 KB	48.20	—	51.31

TABLE 11

DELAY MEASUREMENTS
XTP,TCP,UDP
SUN 4, ONE VIRTUAL CIRCUIT

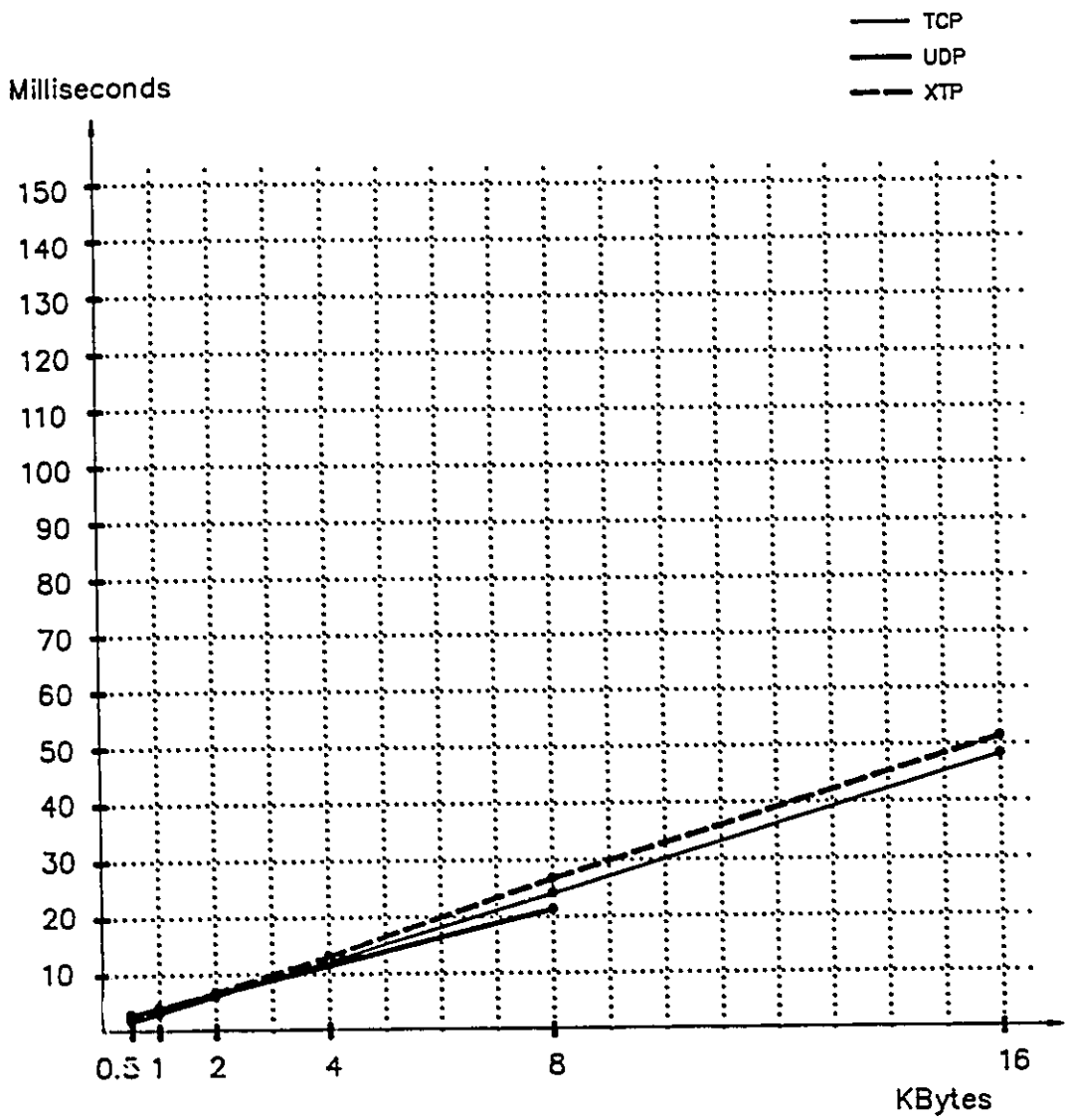

FIGURE 38

DELAY MEASUREMENTS
XTP,TCP,UDP
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 512 B - 16 KB

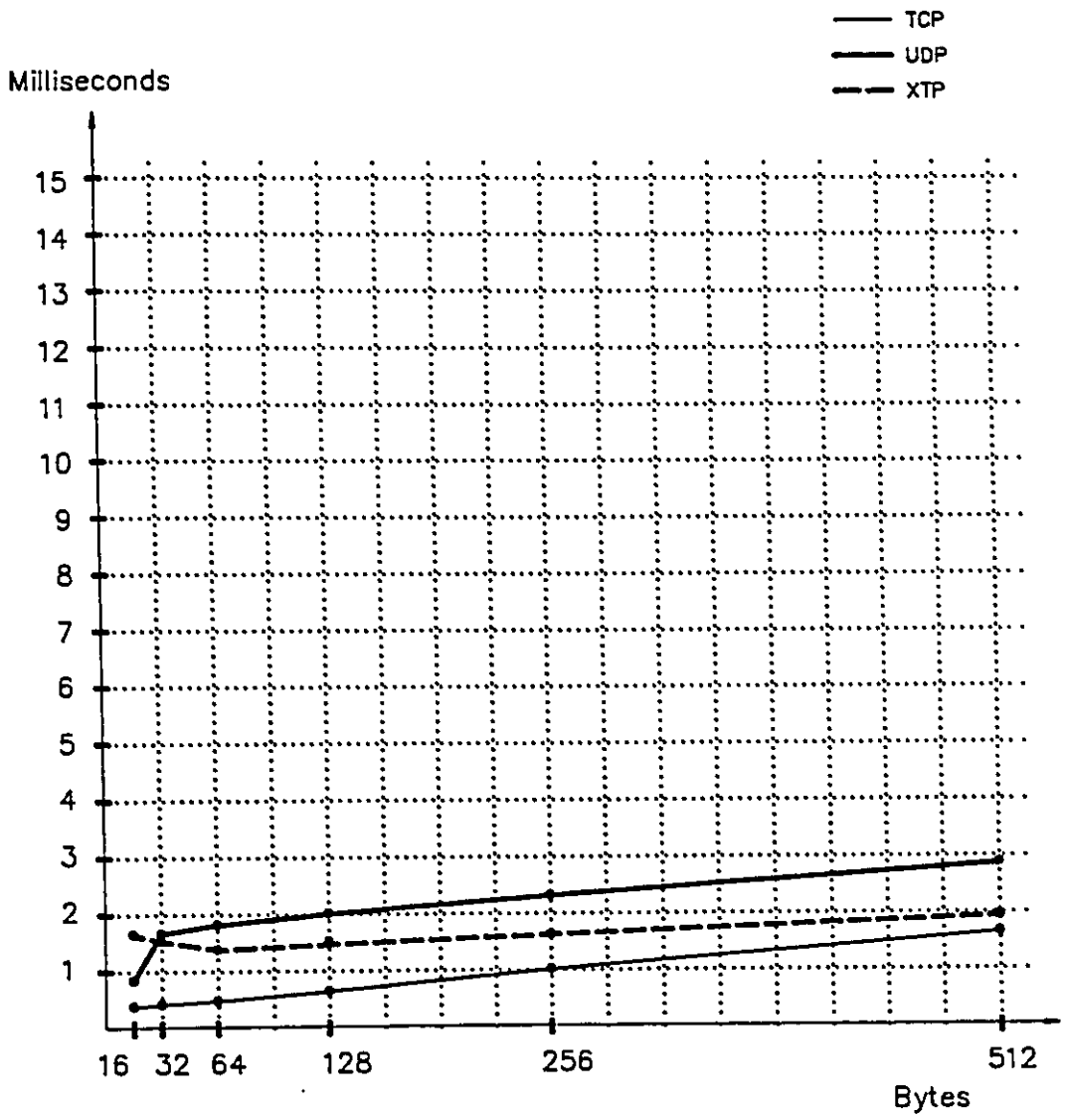

FIGURE 39

DELAY MEASUREMENTS
XTP,TCP
SUN-4, ONE VIRTUAL CIRCUIT
MESSAGE SIZE : 16 B - 512 B

5.2.9 Delay Comparisons

The performance of the two XTP interfaces, the socket and character driver, is very similar.

For comparison with the other protocols, the XTP socket interface, the case where the XTP input/output queues are 32 KB, is used.

Table 11 and Figures 38 and 39, show the results for XTP, TCP, and UDP.

The UDP shows better performance, than XTP or TCP, for message sizes 4 KB to 8 KB.

The TCP performance is slightly better than that of XTP.

Another set of tests have been performed to be able to compare the performance of XTP and TCP when transfers take place on two virtual circuits, at the same time. The results are shown in Table 12 and Figures 40, 41, 42, 43.

When two TCP virtual circuits are running in the same time, the delay for a message of 16 KB, increases by about 54.32%.

When two XTP virtual circuits are running in the same time, the delay for a message of 16 KB, increases by about 49.11%.

When one TCP virtual circuit is transferring data in the same time with an XTP virtual circuit, the delay increases 101.2% for TCP and 1.92% for XTP.

Message size	TCP 1 circuit	TCP 2 circuits	XTP socket i/o que 32 KB 1 circuit	XTP socket i/o que 32 KB 2 circuits	TCP with XTP	XTP socket with TCP
	Milliseconds					
16 B	0.38	0.77	1.65	not measured	not measured	not measured
32 B	0.42	0.86	1.53	not measured	not measured	not measured
64 B	0.48	0.99	1.38	2.39	1.12	2.10
128 B	0.64	1.32	1.47	2.69	1.45	2.51
256 B	1.01	1.91	1.63	2.88	1.87	3.19
512 B	1.65	2.67	1.94	3.38	2.57	3.57
1 KB	3.11	4.94	3.69	4.64	4.85	4.19
2 KB	6.14	9.62	6.83	9.45	9.46	7.18
4 KB	12.20	18.85	13.10	17.89	23.05	13.34
8 KB	24.03	37.56	26.71	37.66	52.36	33.15
16 KB	48.20	74.38	51.31	76.51	96.98	52.31

TABLE 12

DELAY MEASUREMENTS
XTP,TCP
SUN-4 TWO VIRTUAL CIRCUITS
VS. ONE VIRTUAL CIRCUIT

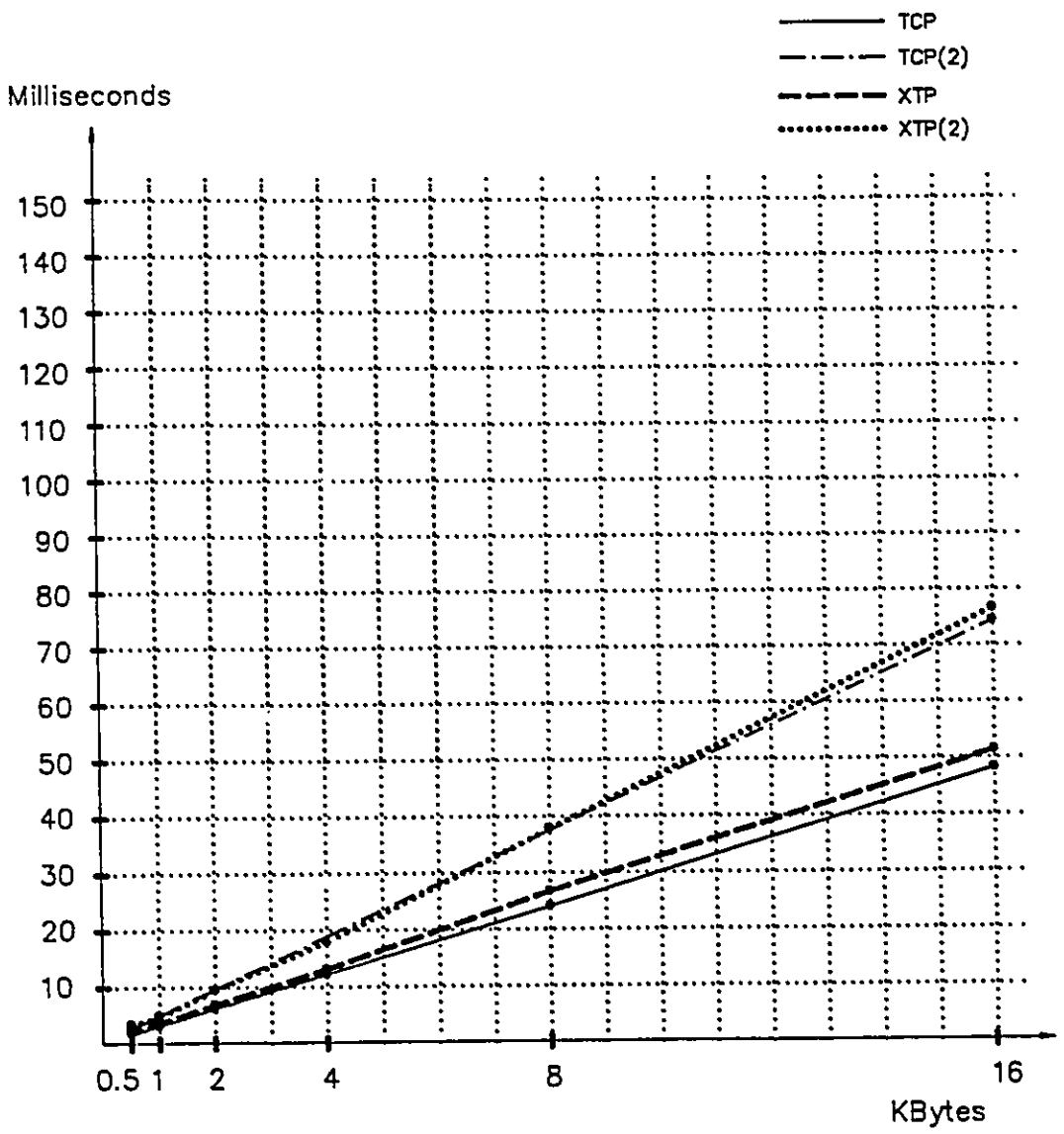

FIGURE 40

DELAY MEASUREMENTS
XTP,TCP
SUN-4, ONE VIRTUAL CIRCUIT
VS.TWO VIRTUAL CIRCUITS
MESSAGE SIZE : 512 B - 16 KB

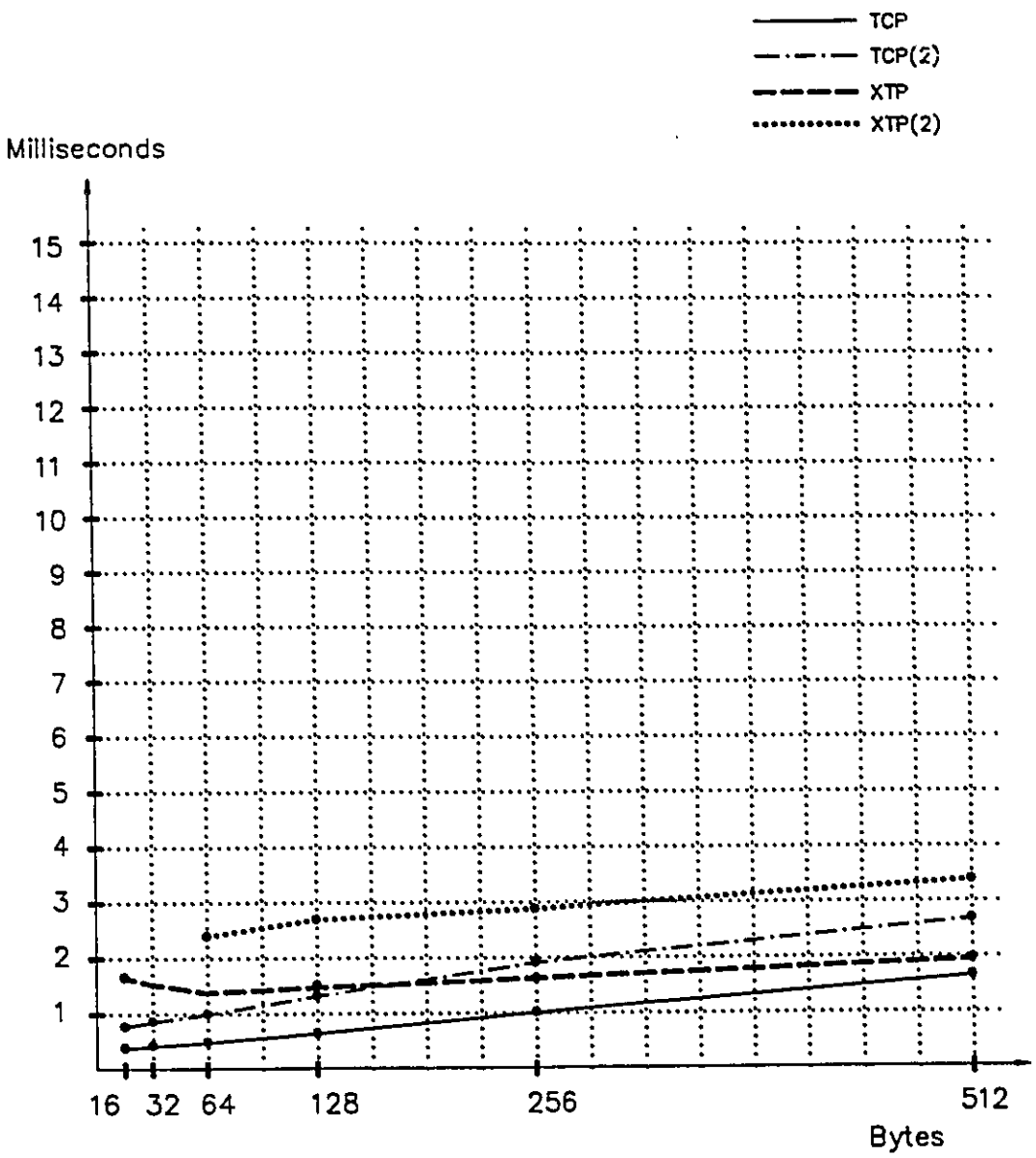

FIGURE 41

DELAY MEASUREMENTS
XTP,TCP
SUN-4, ONE VIRTUAL CIRCUIT
VS.TWO VIRTUAL CIRCUITS
MESSAGE SIZE : 16 B - 512 B

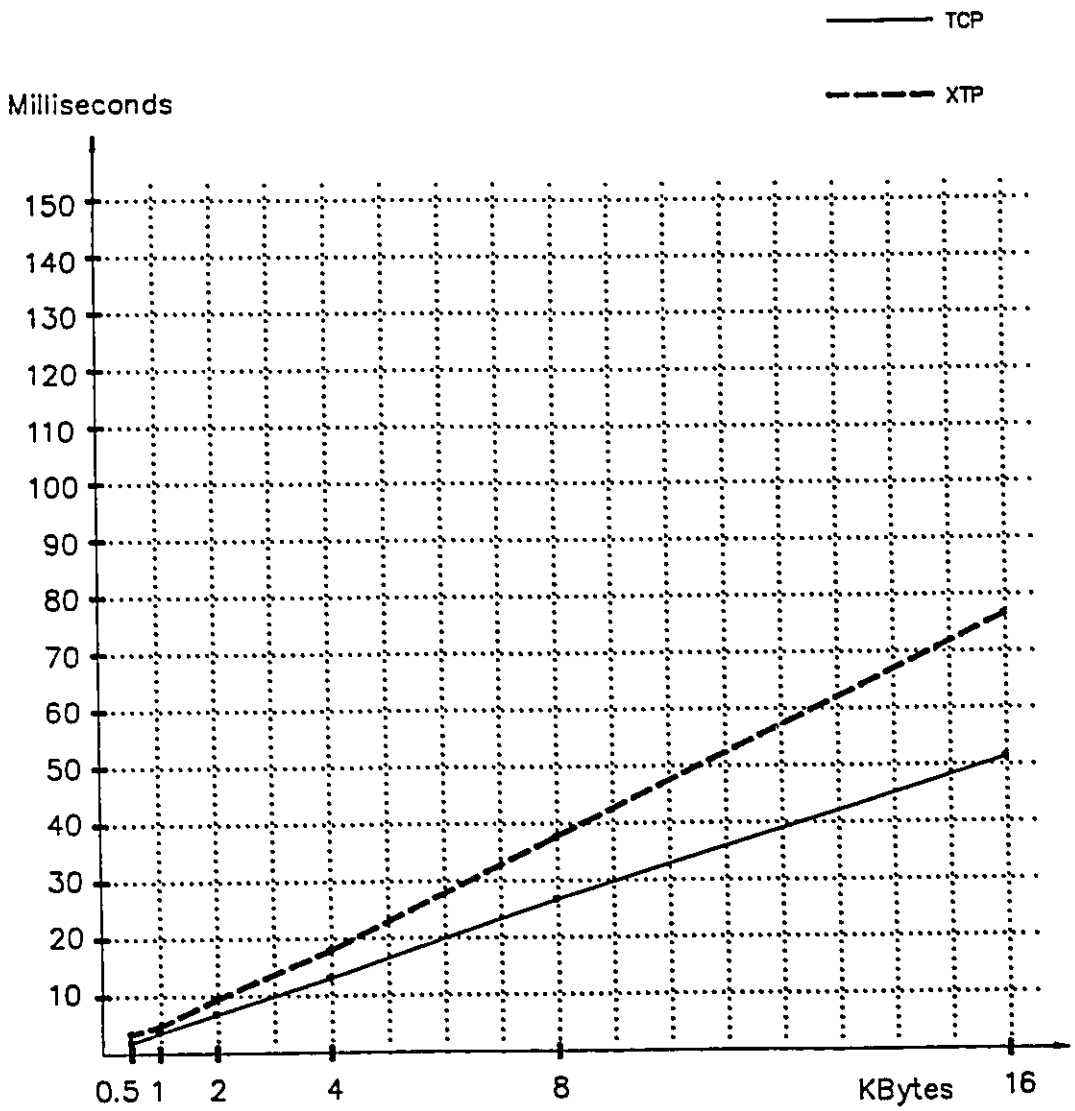

FIGURE 42

DELAY MEASUREMENTS
XTP,TCP RUNNING IN THE SAME TIME
SUN-4
MESSAGE SIZE: 512B-16KB

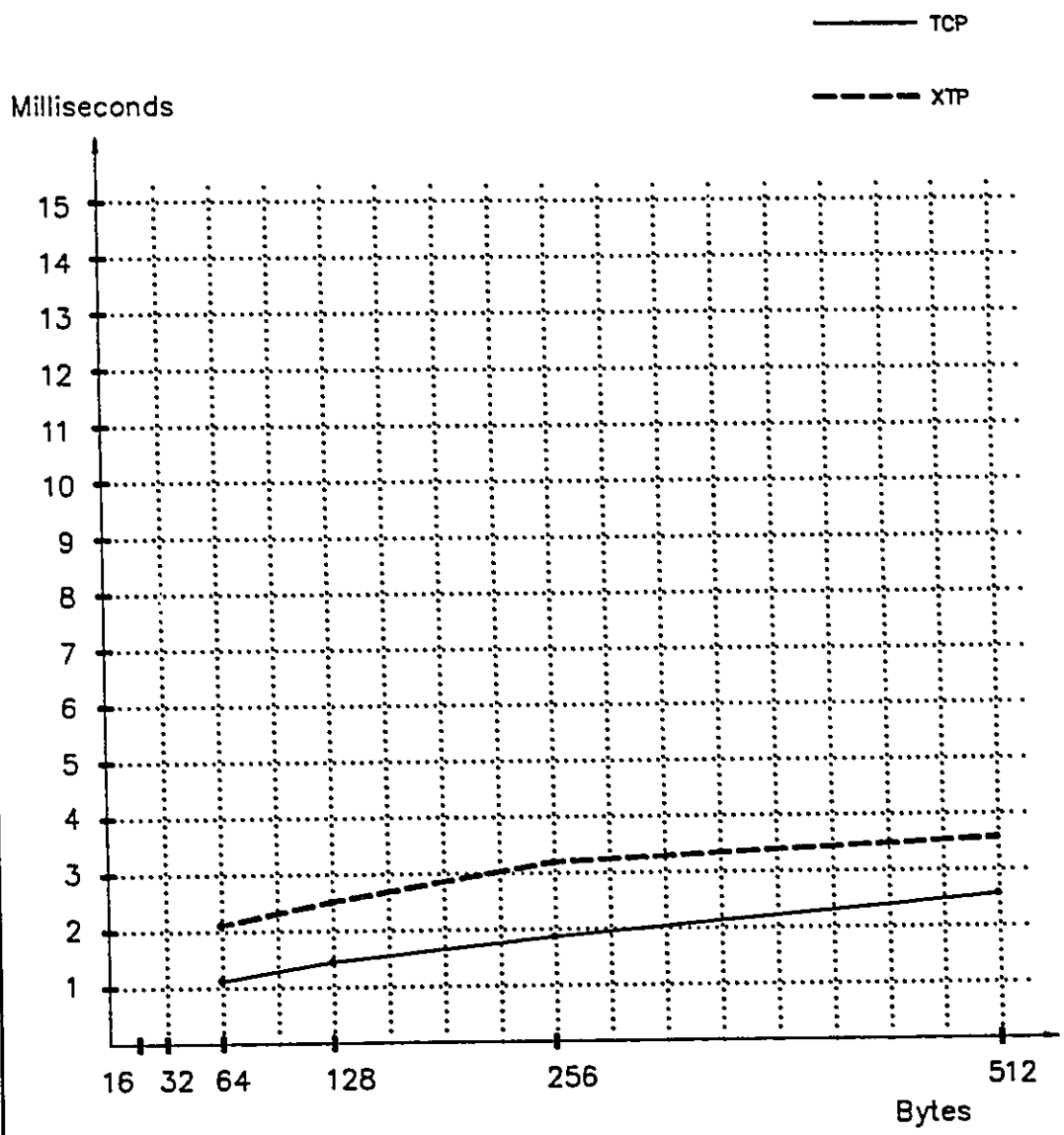

FIGURE 43

DELAY MEASUREMENTS
XTP,TCP RUNNING IN THE SAME TIME
SUN-4
MESSAGE SIZE : 16 B - 512 B

5.3 UDP Receiver Throughput versus Offered Load

The User Datagram Protocol (UDP) provides unreliable data transfer. When a transfer request is made by an application, the interface to UDP returns before the data is delivered.

If the application makes requests too fast, most of the messages may be lost. In order for the receiver to get most of the messages, the transmitter has to be slowed down.

This subchapter describes the measurements of the receiver's throughput versus the offered load (transmitter's throughput).

The results for message size 8 KB in the Sun-3 configuration are shown in Table 13 and Figure 44.

The results for message size 8 KB in the Sun-4 configuration are shown in Table 14 and Figure 45.

As the offered load decreases, the receiver throughput increases and less and less messages are lost. After a certain point, when the offered load continues to decrease, the receiver's throughput will start to decrease too, both measures being approximately equal.

SUN-3		
OFFERED LOAD	RECEIVER THROUGHPUT	MESSAGES LOST
Mb / sec	Mb / sec	%
1.83	1.90	1.17
1.97	2.00	0
2.07	2.16	0
2.31	2.50	0.39
2.69	1.30	54.30
2.93	0.04	98.83
3.59	0.03	99.22
4.04	0.03	99.22

TABLE 13

TROUGHPUT MEASUREMENTS
RECEIVER TROUGHPUT VS. OFFERED LOAD
UDP
SUN-3 CONFIGURATIONS
MESSAGE SIZE 8 KB

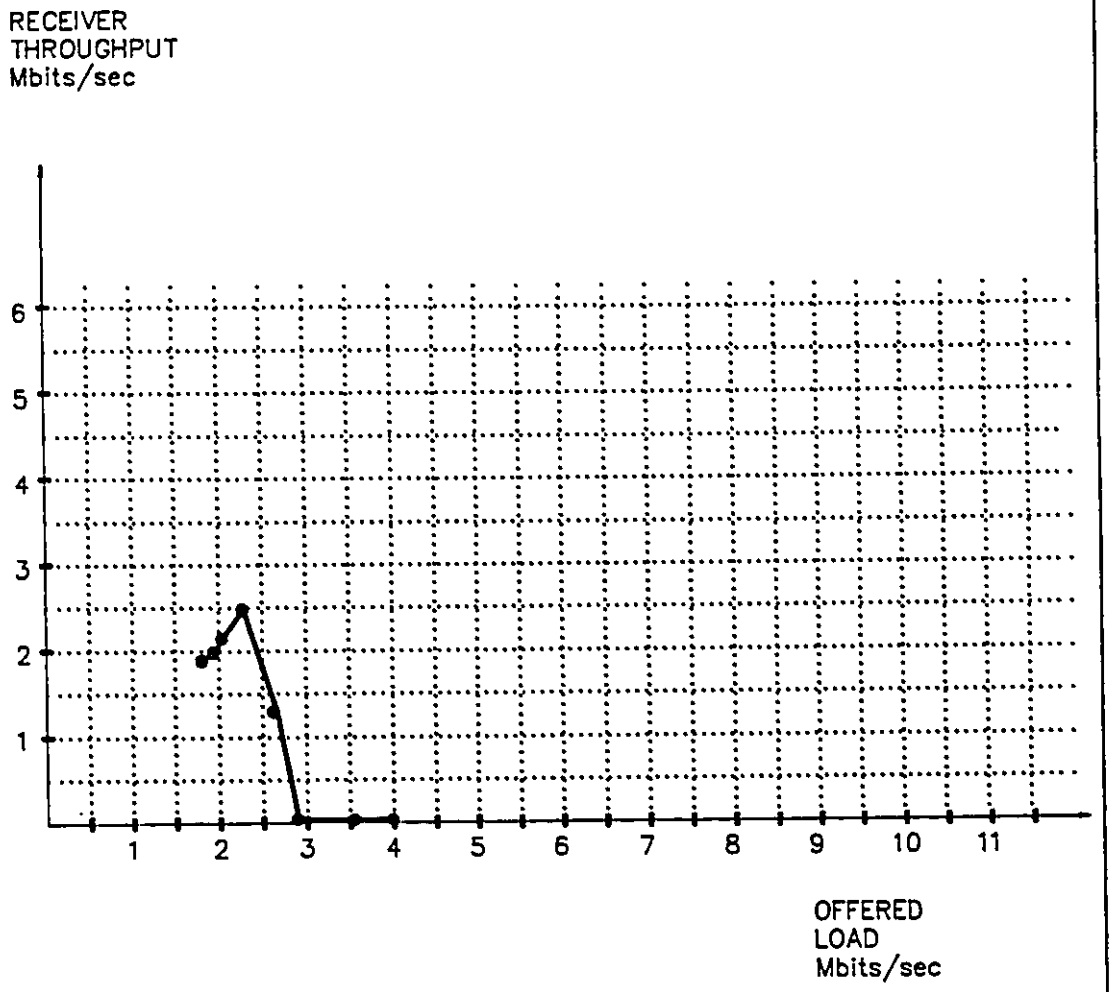

FIGURE 44

TROUGHPUT MEASUREMENTS
RECEIVER TROUGHPUT VS. OFFERED LOAD
UDP
SUN-3 CONFIGURATIONS
MESSAGE SIZE 8 KB

SUN-4		
OFFERED LOAD	RECEIVER THROUGHPUT	MESSAGES LOST
Mb / sec	Mb / sec	%
0.99	1.00	0
1.47	1.48	0
2.75	2.77	0
4.92	4.96	0
6.64	5.34	20.31
8.59	0.97	95.31
10.38	0.61	97.27
11.76	0.67	96.88

TABLE 14

TROUGHPUT MEASUREMENTS
RECEIVER TROUGHPUT VS. OFFERED LOAD
UDP
SUN-4 CONFIGURATIONS
MESSAGE SIZE 8 KB

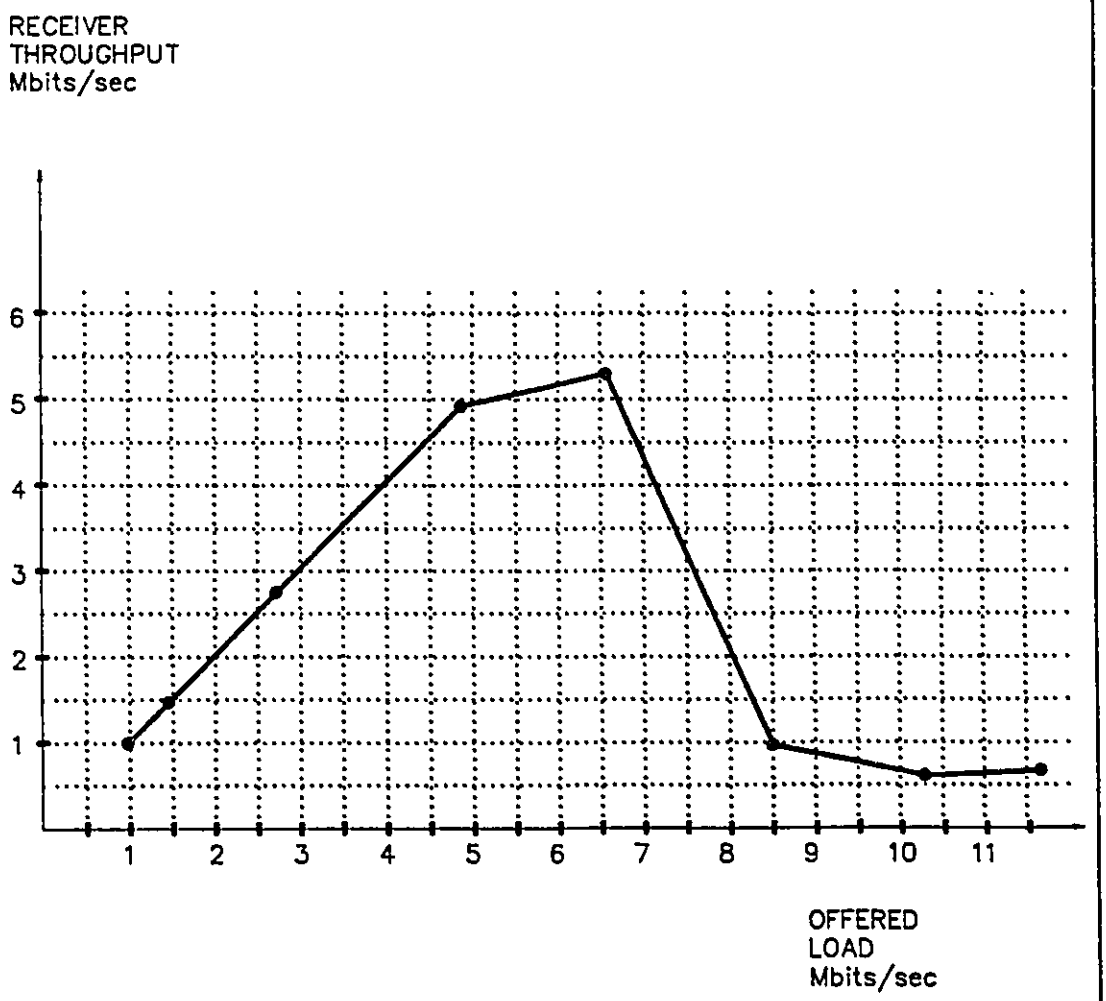

FIGURE 45

TROUGHPUT MEASUREMENTS
RECEIVER TROUGHPUT VS. OFFERED LOAD
UDP
SUN-4 CONFIGURATIONS
MESSAGE SIZE 8 KB

Chapter 6

Conclusions

6.1 XTP Performance

The goal of this project, was to evaluate the KRM 1.7 - the first software implementation of the XTP 3.6, performance wise, comparing it with other existing protocols. The measurements included throughput and end-to-end delay in an Ethernet network.

The choices made in implementing a protocol have a dramatic effect on performance [5].

There are several factors that can affect protocol processing times and thus affect throughput and end-to-end delay. These factors can be divided into three groups. The first group consists of the hardware environment, including the speed of the processor and memory, the network packet size. The second group consists of protocol implementation details, such as timer management, buffer management, connection state management, transfer of data from user, division of the protocol processing into processes. The third group consists in user requirements such as the user message size.

The project investigated the KRM performance in two hardware configurations, in an Ethernet network, with user message size from 16 B to 16 KB. The KRM itself is implemented within the kernel.

The KRM 1.7 did not have performance as one of its design objectives. There are at least two potential modifications that could improve its performance significantly. One is that the KRM interrupt routine could take advantage of specific

configuration representations of input data. The second is the optimization for specific machines and operating systems.

But even in the current state the KRM 1.7 provided superior throughput and delay performance, in both hardware configurations, especially for the larger message size.

The XTP has as an objective its hardware implementation, which eventually will provide even better performance.

6.2 Future Work

This project did not cover performance evaluation of XTP's multicast facility. This is one of the open areas of investigation.

Others would be the search for performance improvements of the current KRM, the evaluation in a higher speed network such as FDDI, and finally the comparison with another "new generation" transport protocol.

Acronyms

ANSI = American National Standards Institute

API = Application Programming Interface

FDDI = Fiber Distributed Data Interface

HSTP = High Speed Transport Protocol

IEEE = Institute of Electrical and Electronics Engineers

IP = Internet Protocol

ISO = International Organization for Standardization

KRM = Kernel Reference Model

LAN = Local Area Network

LANCE = Local Area Network Controller for Ethernet

LLC = Logical Link Control

LSAP = Link Layer Service Access Point

MAC = Media Access Control

Mbps = Megabits per second

MIPS = Million Instructions Per Second

NIT = Network Interface Tap (SUNOS 4.1.1)

OSI = Open Systems Interconnection

RISC = Reduced Instruction Set Computing

SPARC = Scalable Processor ARChitecture

TCP = Transmission Control Protocol

TP4 = ISO Transport Protocol Class 4

TPDU = Transport Service Data Unit

TSDU = Transport Protocol Data Unit

UDP = User Datagram Protocol

XTP = Xpress Transfer Protocol

VLSI = Very Large Scale Integration

References

[1] Atwood J.William, Chen, Jason X.G.: Performance of the Xpress Transfer Protocol in an Ethernet Environment

[2] Cheriton, D: VMTP: A Transport Protocol for the Next Generation of Communications Systems, IEEE Computer Communications Review, Vol.16, No.3, 1986

[3] Cheriton, D: VMTP as the Transport Layer for High-Performance Distributed Systems, IEEE Computer Communications Magazine, June 1989

[4] Hartrick, Timothy W.: Performance Analysis of a Software Implementation of the Xpress Transfer Protocol, Master of Science (Computer Science) Thesis, University of Virginia, august 1991

[5] Heatley, Sharon, Stokesberry, Dan: Analysis of Transport Measurements Over a Local Area Network. In IEEE Communications Magazine, June 1989

[6] Protocol Engines Inc. Document 92-106: Kernel Reference Model (KRM) User's Guide, Version 1.7, May, 1992

[7] Protocol Engines Inc. 92-10: XTP Protocol Definition Revision 3.6, 11 January 1992

[8] Sanders, Robert M.: The Xpress Transfer Protocol - A Tutorial, Department of Computer Science, University of Virginia

[9] Stallings, William: Data and Computer Communications, Macmillan Publishing Company

[10] SunOS 4.1 System and Network Administration

[11] SunOS 4.1 Network Interface Tap (NIT) Manual Pages

[12] Tanenbaum, A.S.: Computer Networks, Prentice-Hall Inc., New York, 1981