SCS&E Report 9402

HTPNET: A New Transport Protocol for High-speed Networks

Toong Shoon Chan and Ian Gorton

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
THE UNIVERSITY OF NEW SOUTH WALES

Abstract

The quantum increase in transmission speed of optical fibre networks has created a bottleneck in transport protocol processing at host systems. In this paper we present a new transport protocol system, HTPNET, that is designed to overcome this protocol processing bottleneck. HTPNET is based on a highly parallel architecture and is designed to exploit the evolving characteristics of high-speed networks. HTPNET uses an out-of-band signalling system based upon transmitter-paced periodic exchanges of state information between end systems. This mechanism exhibits several attractive properties which have been demonstrated to perform efficiently in a high-speed environment with high-bandwidth-delay-product. A prototype implementation of HTPNET has been constructed from a network of T800 transputers. The results obtained from this implementation are presented: these demonstrate the advantages of exploiting a parallel architecture for protocol processing.

E-mail:
chants@spectrum.cs.unsw.oz.au iango@spectrum.cs.unsw.oz.au

1. INTRODUCTION

The advent of fibre optics technology has enabled the development of high-speed networks that are able to provide far higher communication bandwidth, with extremely low error rates. Although the processing capacity of communication processors has risen steadily, they are not able to keep pace with the additional burden of greatly improved transmission media speeds. This mismatch in speed has created a bottleneck in the communication protocol processing. Consequently, this has resulted in applications that are unable to exploit the full potential of the transmission media offered by the high-speed networks.

In early networks, for example X.25 which operates at low speed (< 64 kb/s) with high error rates (approximately 10^{-6}), the network was designed to support inter-system communication with the end-user fully aware of the high delay introduced in the underlying network. This delay was the result of the high overheads of early protocols, that were designed to be robust in the face of adverse conditions. High protocol overheads were necessary for the proper control of the transmission of data, so that better utilisation of expensive bandwidth could be achieved. Consequently, the protocol architecture of an X.25 WAN is designed to have error recovery and flow control functions in multiple protocol layers, introducing overlap in the functions [2]. However, given the wider application mix and high-speed communication environment of future high-speed broadband networks, duplication of functionality is highly undesirable. In broadband networks like B-ISDN, protocol functions inside the network have been streamlined and are limited only to switching, routing and relaying functions. Streamlining of protocols facilitates high speed relaying of data, and is amenable to a custom-built hardware implementation [3].

As a consequence, critical functions for providing reliable data transport have been shifted to the host systems. For existing protocols such as TCP/IP, the high overheads associated with protocol processing creates a massive imbalance between the processing capacity of the protocols and the transmission speed of the underlying network. This mismatch in speed reduces data transfer rates between hosts and degrades the overall performance of the underlying network [1].

In addition, the evolution of lightwave networks has altered network characteristics, bringing about a higher bandwidth-delay-product and lower error rates. These changed characteristics will have a major impact on the design of future transport protocols. Consequently, the advent of broadband networks has raised the challenge of designing alternative transport protocol systems that are able to provide high-speed protocol processing, without compromising on functionality, to provide reliable data transport.

This paper introduces a new transport protocol processing system, HTPNET (**H**igh-speed **T**ransport **P**rotocol for **Net**works), which is designed for high-speed data communications over high-speed networks. HTPNET is designed not only to efficiently support parallel

architectures, but also takes into consideration the communication network characteristics, namely increased bandwidth-delay-product and lower error rates. HTPNET is currently implemented on a network of five T800 transputers. Experiments have revealed promising packets transfer rate of 12000 to 13000 packets/s. With a packet size of 2 Kbytes, a data transfer rate of more than 200 Mb/s could be achieved.

2. DESIGN ISSUES FOR FUTURE TRANSPORT PROTOCOL

With the changing characteristics of high-speed networks and a wider application mix, there is a necessity to re-examine end-to-end protocol requirements. The formulation of an effective and efficient transport protocol system is particularly essential in the design of high-speed networks. A suitable transport protocol must provide reliable transport services to higher layers while attaining maximum bandwidth utilisation.

2.1 Implementation Issues

One of the major drawbacks identified in existing protocols is the extreme reliance on the host CPU to perform protocol processing [4,5]. This results in heavy usage of timers, interrupts and context switching, thus causing severe overheads at the host to support protocol processing. For a high-speed network operating at 1Gb/s, with a packet size of 1 Kbytes, a processing time of 8 microseconds for each packet is required. Given that a typical protocol packet would require approximately 250 instructions [6], then the processing speed required merely to support network traffic is about 30 MIPS (Million Instructions Per Second).

Therefore, in order to meet the high speed requirements of protocol processing, we believe additional processor support must be provided to off-load much of the network overheads from the host system. This approach of separating protocol processing from the main host operating system has been used in the design of VMTP network adapter board[7]. Their analysis has shown that 6% of the delay in packet processing is due to network delay, 12% is due to processing on the network adapter board and 88% is due to host processing. Although it is possible to design dedicated hardware for a specific protocol, as demonstrated in XTP[5], the drawback is the difficulty in adapting to changing requirements. Another approach that meets cost constraints and yet remains adaptable to changing requirements is the application of parallel processing techniques using commercially available microprocessors such as transputers[4]. A network of transputers can be constructed to distribute the protocol processing workload. Importantly, the ease with which transputer systems can be reconfigured to improve performance makes this solution highly adaptable to changes in processing requirements.

2.2 Architecture Issues

In first generation networks, efficient bandwidth utilisation was an important factor in the design of transport protocols. Furthermore, the processing speed at host systems was sufficient to match the transmission speed of the underlying network. This gave rise to

protocols that use in-band signalling [1] for the exchange of information between peer transport entities. With in-band signalling, the control and data functions are combined together as a single entity. Thus, the actual body of data is encapsulated with all the control information necessary for the reliable transport of data. At the receiving end, each packet is parsed to determine the control information that is present, so as to invoke the necessary actions to ensure correct operation. As a consequence, the processing of control information for each received packet is inherently sequential, and this leads to increased processing time for the embedded data portion.

On the contrary, out-of-band signalling aims to transmit the control information and data as separate packets. Therefore, at the receiving end, a simple hardware circuit can be built to identify different packets, which can then be routed to the corresponding processing units. The major advantage of out-of-band signalling is the ability to support parallel architectures efficiently[1]. In addition, out-of-band signalling is adaptable to changing needs. New services can be added without incurring any impact on the data transport mechanism. For example, to introduce congestion detection at the end systems, a special control packet can be used to detect the condition of the network without having any impact on the actual data transfer. The drawback of out-of-band signalling is the increased bandwidth needed for transmission of individual control packets. However, with a quantum increase in available bandwidth in high-speed communication systems and the ability to support efficient parallel architectures, the out-of-band signalling technique is an attractive proposition for incorporation in future transport protocol designs.

2.3 New Design Issues

Due to the ease of implementation and minimal buffering, the *go-back-n* technique has been employed by most existing protocols for error recovery in terrestrial networks. In this method, the receiver only buffers those packets that are error-free and in-sequence. If one of the packets in sequence is lost, then the subsequent packets that are still in transit have to be retransmitted. In high-speed wide area networks, the bandwidth-delay-product[1] can work out to be very large, requiring a very large window size at each end of the connection to achieve high throughput. For example, for a network operating at rates of 100 Mb/s, with a propagation delay of 30 milliseconds and a packet size of 1 Kbytes, the bandwidth-delay-product is 1000 packets. Thus, if the go-back-n mechanism is used, this could mean retransmitting the entire window of 1000 packets due to error in one packet. These retransmitted packets, however do not necessary arrive at the receiver error-free, and possible retransmission may be necessary. This results in multiplicative packet loss in the underlying network. Therefore, in view of the high bandwidth-delay-product for high-speed communication system, the go-back-n error recovery strategy needs to be re-examined if optimal throughput is to be attained. In HTPNET, a technique known as *selective retransmission* has been devised: selective retransmission is explained in section 3.

---

[1]Bandwidth-delay-product is defined as the product of the bandwidth (BW) and the round-trip-delay (RTD), ie. w=BW * RTD, where w is the amount of unacknowledged data the transmitter needs to send so to keep the pipeline full and maximise throughput.

Mechanisms for detecting packet loss in communication networks have traditionally utilised *timers*. Experience with the TCP retransmission timer has clearly shown that timers have limitations in offering optimal performance[8], particularly for high-speed communications. One of the major problems with using timers is the difficulty in obtaining an accurate estimate of the round-trip-delay (RTD) due to lack of information about network topology and dynamics. If the timer value is too short, then the timer will time-out too soon, resulting in unnecessary retransmission and duplicates. On the other hand, if the timer value is set too high, it will result in detection delay of packet loss. Therefore, to obtain high performance, extreme reliance on accurate knowledge of the RTD to detect packet loss should be avoided.

Another drawback of timer based mechanisms is the overhead of managing timers. For a network operating at 100 Mb/s, with packet size of 1K bytes and a RTD of 30 milliseconds, the transmitter will need to manage about 1000 timers at a particular instance of time. This considerable amount of computational overheads would place unacceptable overheads on the transmitter processor[6]. In HTPNET, a new mechanism for detecting loss of packets has been incorporated. This mechanism has the advantage of requiring only a coarse estimate of RTD, with low computational overheads of managing a single timer.

Early transport protocols like the TCP and TP4 employ a sliding window mechanism for the end-to-end flow control. The objective of this control is to avoid overrunning buffers at the receiver. This is done by some kind of feedback mechanism, to ensure that the transmitter is synchronised with the receiver's ability to keep in pace. However, this mechanism does not take into account the condition when the transmitter packet rate exceeds the processing rate at intermediate network nodes. Such continuous over-admission of packets results in buffer overflow at the nodes, thus causing data loss. Consequently, severe congestion may occur due to excessive retransmission by host systems affected by the overrun node(s). If the situation persists, this will eventually lead to a congestion collapse[9]. To avoid such congestion, HTPNET is designed with a rate control mechanism which dynamically adapts the rate at which packets are transmitted, based on the situation of the underlying network.

3. HTPNET OVERVIEW

The design of HTPNET aims to decouple as far as possible protocol processing from the main operating system using an additional multiprocessor-based subsystem. In addition, the structure of the HTPNET protocol is based on several finite state machines which are designed to exploit a parallel implementation and achieve high throughput. This is in contrast to the implementations of conventional protocols like TCP and TP4, which are based on single finite state machine. The feasibility of this approach is demonstrated by the current experimental implementation of HTPNET, which has been designed to execute on a network of T800 transputers.

3.1 Protocol Architecture

The abstract architecture of HTPNET is horizontally structured as opposed to the conventional vertical structuring of most protocols. The rationale for the horizontal structure of HTPNET is based on the division of the protocol into separate loosely-coupled management functions. These functions are essentially independent of each other, and can be executed on different processors in parallel with minimal inter-communications between the functions. HTPNET is divided into four main functions: transmitter data packet processor, receiver data packet processor; transmitter control packet processor and receiver control packet processor (see Figure 1). A dual-port shared packet store at each node is incorporated to store the raw data that is moved across the network, while only the address reference to the data is processed. This memory architecture effectively eliminates the overhead of moving large amounts of data around the memory, while the dual-ported memory allows simultaneous access from host and network interface.

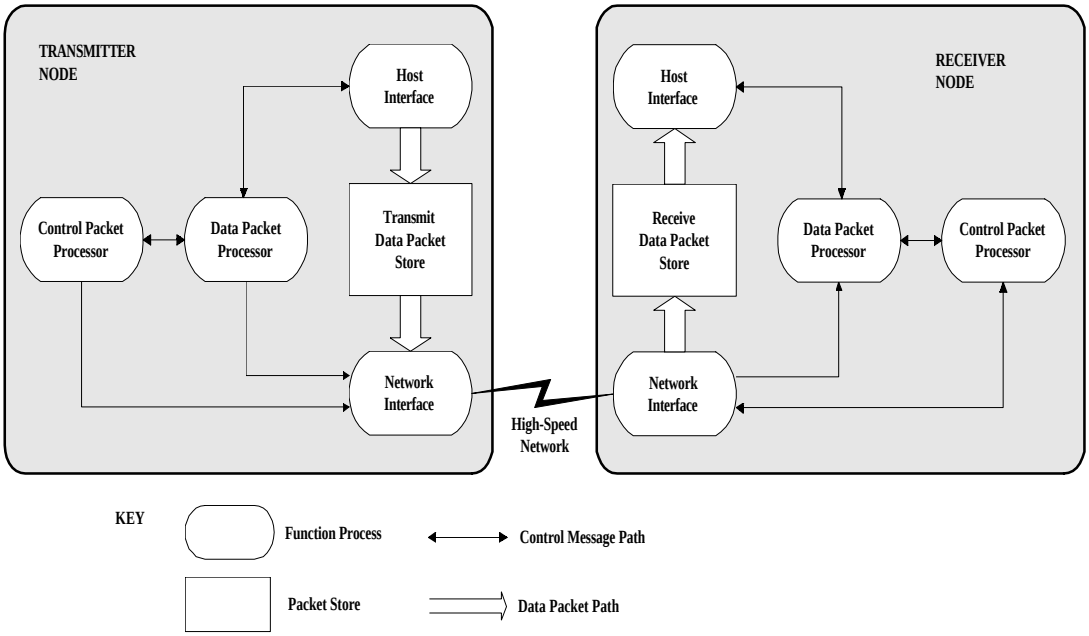


Figure 1 HTPNET Abstract Architecture

The segmentation of HTPNET into distinct horizontal functions allows the data packet processor to dedicate its computing power to handling data packets. Concurrently, multiple processors can be used to process control messages, maintain state information, perform error recovery and implement flow and congestion control functions. In addition to simplifying the data processing function, the parallel architecture also removes the high overheads caused by context switching between functions. Instead, occasional control messages are passed between processors using fast communication links.

3.2 Packet Format

HTPNET consists of eight types of packets: connection request (CR), connection confirm (CC), disconnect request (DR), disconnect confirm (DC), reject (RJ), data (DT),

transmitter state (TXS), receiver state (RXS). To setup a connection, a three-way handshake protocol is used to establish a connection between two end points similar to that of TCP [10]. This setup allows the endpoints to negotiate communication parameters and options, including protocol class, available buffer(flow control) and inter-packet gap(rate control). HTPNET is designed with eight variations of protocol classes to provide different quality of services (QOS). Table 1 defines the functions available for different protocol classes. Class 0 assumes a perfect network with all control functions disabled. On the other extreme, class 7 assumes a highly unreliable network and consequently requires the enabling of all control functions. This diversity of available protocol classes provides host with the flexibility to select QOS based on application's requirement and underlying network. Although the protocol class is negotiated during setup time, a connection may dynamically re-negotiate the class to allow adaptive protocol processing based on changing network environments. To release a connection, HTPNET uses a symmetric three-way handshake protocol used in establishing a connection. To avoid any data loss upon disconnection, a graceful close mechanism is used to ensure that as the connection moves from data transfer to termination, all data intended for the corresponding end points are received.

Class	Error Control	Flow Control	Rate Control
0	No	No	No
1	No	No	Yes
2	No	Yes	No
3	No	Yes	Yes
4	Yes	No	No
5	Yes	No	Yes
6	Yes	Yes	No
7	Yes	Yes	Yes

Table 1 Protocol Class Definition

The remaining three type of packets shown in Figure 2 are used during data transfer. Table 2 defines the function of each field associated with the packets. The state packets (TXS and RXS) are used for state synchronisation between the end-points. In addition to carrying control information, it may be used to dynamically re-negotiate the protocol class for adaptive protocol processing. Further, the underlying network nodes have the option of dynamically adjusting the rate field of the transmitter state packet to reflect the traffic status in the network. Similarly, at the receiver, this rate is examined to ensure that it is within the receiver's acceptable rate. This rate is echoed back to the transmitter using receiver's state packet to dynamically adjust the transmitter's sending rate. The data packet (DT) is used solely for the transport of user data, with no provision for piggyback control information. This lightweight structure of the data packet ensures minimum processing overhead by avoiding unnecessary parsing of large amount of control information as experienced in TCP.

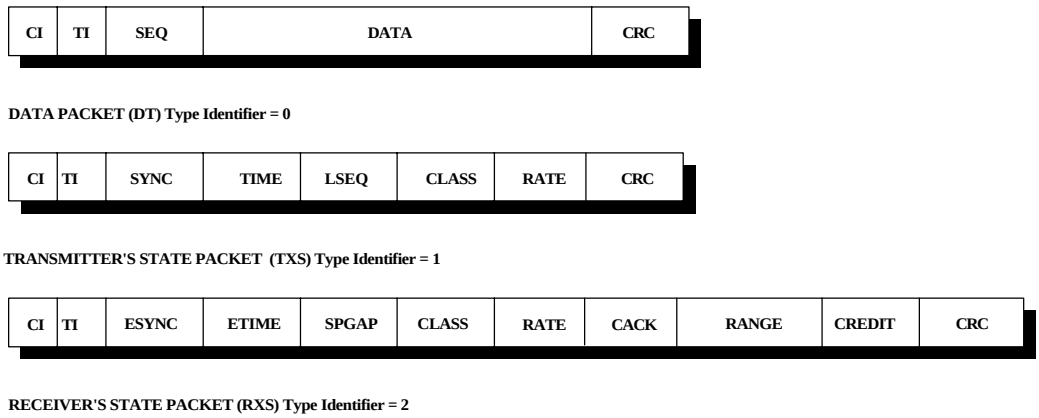


Figure 2 HTPNET Data and Control State Packets

FIELD	DESCRIPTION	BYTES	COMMENTS
TI	Type Identifier	1	Identify packet type.
CI	Connection Identifier	1	Value assigned to each logical connection during connection setup.
SEQ	Sequence	2	Sequence number assigned to data packet to ensure sequenced delivery.
SYNC	Synchronisation	2	Synchronisation number used to identify state packets during period state exchange.
DATA	Data	2K	Contains user's data to be transported.
CRC	Cyclic Redundancy Code	2	For error detection.
TIME	Time	2	Indicates the time when transmitter's state packet is admitted into network. Use along with ETIME to obtain network's RTD.
LSEQ	Largest Sequence	2	Records the largest sequence number of packets transmitted.
CLASS	Class	1	Variations of protocol classes for different level of quality of services (QOS).
RATE	Inter-packet gap	2	Use to specify the maximum rate (in terms of inter-packet gap) at which data can be transmitted. Use in conjunction with rate control mechanism.
ESYNC	Echo Synchronisation	2	Use along with SYNC for hosts state synchronisation. The ESYNC echoes the SYNC value back to the transmitter.
ETIME	Echo Time	2	Use along with TIME for RTD estimation. The ETIME echoes the TIME value back to the transmitter.
SPGAP	State Packet Inter-Gap	2	Measure of time units gap between two successive transmitter's state packets, used for dynamic rate control.
CACK	Cumulative Acknowledgment	2	Indicates that those packets with sequence number up to CACK have been successfully received.

RANGE	Range	16	To convey acknowledgment of packets beyond CACK, four pairs of range information are used. Each pair represents a contiguous sequence of packets received successfully.
CREDIT	Credit	2	CREDIT is used to convey to the transmitter of the buffer availability at the receiver for dynamic flow control.

Table 2 Field Definition

3.3 State Synchronisation and Error Recovery

To support a parallel architecture, HTPNET employs out-of-band signalling, in which peer-transport entities exchange data and state information using separate packets. A state information packet is exchanged frequently so that the state between the transmitter and receiver is synchronised. A unique feature of HTPNET is the *transmitter paced* periodic exchange of state information between the end systems. The idea of periodic state exchange was first introduced by a data-link protocol known as Checkpoint Mode Protocol (CMP) [11]. CMP uses periodic transmission of complete state information via acknowledgments from the receiver to the transmitter for the error control mechanism. More recently, the SNR protocol [12] has employed a similar idea in the exchange of state information in both directions between end systems. Importantly, in contrast to SNR which exchanges state information independently between end systems, HTPNET utilises the transmitter node to control the periodical exchange of state information. This mechanism exhibits several attractive properties, which are described in section 4.

The state information exchange is initiated periodically by the transmitter. Each state packet sent from the transmitter is associated with a synchronisation value (**SYNC**) that is incremented after each transmission. Similarly, data packets that are sent before the next state exchange are associated with the next synchronisation value in sequence. For example, if the next state exchange will contain a synchronisation value of 20, then those data packets that are sent between synchronisation value 19 and 20 will be associated with synchronisation value 20. Periodically, a state packet is admitted into the network which traverses the same path as the preceding data packets. When the receiver node receives the state packet, it copies the synchronisation value to construct an outgoing state packet which includes the complete state and error control information of the receiver. Upon receipt of the return state packet, the transmitter updates the status of the transmitted data packets by comparing the synchronisation values of the transmitted data packets with the echoed synchronisation value (**ESYNC**). Note that at this stage, the echoed synchronisation value at the transmitter has travelled a full round-trip (RTD) across the network between the transmitter and receiver since it was first admitted into the network. Consequently, for unacknowledged data packets whose associated synchronisation value is less than or equal to the echoed synchronisation value, a retransmission is necessary.

The basic state synchronisation mechanism is shown in Figure 3. For each received state packet at the transmitter, a bit-map containing the status of the data packets is assembled and communicated to the main data packet processor, which performs the necessary

retransmission and release of data packet memory. The advantage of this mechanism is the complete elimination of the reliance on accurate RTD timings to detect loss packets. Furthermore, the computational overheads of managing huge numbers of timers are now replaced by the simple management of a single table to keep track on the status of the transmitted data packets. In short, this whole mechanism greatly simplifies data packet processing, eliminating almost all error recovery procedures. The main transmitter and receiver processors can focus on handling outgoing and incoming data packets, while other processors can manage the state information, thus effectively parallelizing the entire protocol processing task.

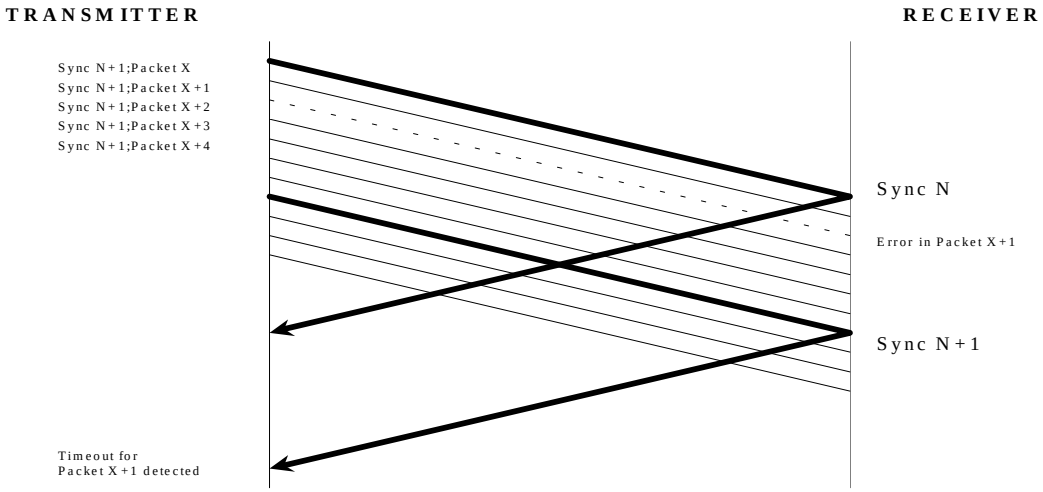


Figure 3 Basic State Synchronisation Operation

3.4 Selective Retransmission

The main drawback of *go-back-n* mechanism is the waste of bandwidth to retransmit successfully received data due to an error detected at the receiver. To avoid this unnecessary retransmission, HTPNET employs a mechanism called selective retransmission. Although the implementation of this method is more complex than go-back-n in terms of packet and timer management, the result is higher throughput under error and congested conditions.

In HTPNET, when the receiver periodically receives a state packet from the sender, it conveys the full status of the data packets received by supplying a cumulative acknowledgment (**CACK**), as well as the range information (**RANGE**) on the packets received error-free. The cumulative acknowledgment with sequence number N indicates that those packets with sequence number up to N have been successfully received. To convey acknowledgment of packets beyond sequence N, range information in a set of pairs ($bseq_n$, $eseq_n$) is used, where n represent the nth pair. The pair ($bseq_n$, $eseq_n$), represents a contiguous sequence of packets that have been successfully received. For example, if packets with sequence number up to 300 are received error-free, 301 and 302 are missing and 303 to 350 have arrived error-free, the cumulative acknowledgment N

would be 300, while the range information is (303,350). The resultant status would be 300;303;350. Currently, HTPNET is able to convey up to eight pairs of range information.

3.5 Congestion Avoidance

HTPNET is designed with a congestion control scheme based on periodic traffic intensity feedback (PTIF) reactive control. It must be stressed that we view congestion control as a two-level strategy operating on short and medium time-scales[13]. In contrast to short term control which employs an open-loop strategy, PTIF employs medium term control which uses a close-loop strategy by incorporating feedback signals that convey the status of the network to dynamically adapt the source load.

In HTPNET, control packets are transmitted periodically, with a period of α. If we assume at the congested node, x packets are queued between two successive control packets c_n and c_{n+1}, then the instantaneous arrival rate can be expressed as:

$$\lambda_{n+1} = x/\alpha$$

Under congestion, these packets will be served with a period of β, where $\beta=\alpha+\varepsilon$, where ε is the delay caused by the congestion at the queue. Since we can measure the period β at the receiver by subtracting the arrival time of c_{n+1} from c_n, we can derive the virtual service rate as:

$$\mu_{n+1} = x/\beta$$

Subsequently, we can derive the instantaneous traffic intensity as:

$$\rho_{n+1} = \lambda_{n+1}/\mu_{n+1} = x/\alpha * \beta/x = \beta/\alpha$$

Thus, we can calculate the traffic intensity signal, ρ, at the source end by supplying the period β (**SPGAP**) on the return control packet from the receiver.

The instantaneous traffic intensity signal will serve as a good indication on the traffic condition of the network. This signal directly gives us the relative imbalance between the virtual service rate at the congested node and the sending rate at the transmitter. This is in contrast with RTD delay feedback based control, in which the measured delay is directly used to adapt its sending load. Such adaptation often employs trail and error technique to obtain the best function between the delay and adjustment parameters. This results in employing a function that is relatively simple, such as a two-level control. In contrast, PTIF is designed with a more comprehensive control function to achieve fast response to traffic surges and consequently yielding higher throughput performance. Although this function is implemented as an option, it can proof to be useful in avoiding congestion, particularly for networks that do not participate in dynamic rate control.

The congestion avoidance mechanism is implemented in HTPNET by a process executing in parallel with the rest of the protocol processing functions. This example clearly

demonstrates how additional services can be added to exploit the parallel architecture of HTPNET, without having any impact on the fundamental data transport mechanism.

4. DISCUSSION

The design of HTPNET has been greatly motivated by several protocols, most significantly by SNR[12].

SNR is a light-weight transport protocol designed for data communications over high-speed networks. The protocol uses independent periodic state exchange between the transmitter and receiver in both directions for end-to-end state synchronisation. A distributed clock is used to implement "timers" at the transmitter based on the periodical receipt of acknowledgment packets (ACK) from receiver. The ACK packets are transmitted in periodical intervals Tp = RTD/c, where c is a constant. Each time a ACK is received, the "timers" associated with the transmitted packets are decremented by k(value k is embedded in the ACK packet), where k is the interval between two consecutive ACK packets. As pointed in [14], SNR exhibits a behaviour known as time-slipping. The retransmission time of the blocks at the transmitter relies heavily on the arrival of ACKs. However, if an ACK is lost along the way, then time-slips will occur. Thus, even if the time out value is set to the optimal value, due to slipping effect, the time out value is no longer optimal. This slipping effect essentially affects all the blocks that are waiting for acknowledgment. Thus, for a high bandwidth-delay network, this problem could be affecting more than 1000 packets for each lost ACK. The problem is made worse by the fact that the time-slip is cumulative, in that, each subsequent lost of ACK will further worsen the problem.

Another potential problem with SNR is that blocks can still be retransmitted unnecessarily if the transit time increases substantially between the time of packets transmission and reception, as shown in Figure 4. SNR only solves half of the problem by making the arrival of ACKs vary proportionally to the transit time variance from the receiver to the transmitter, but does not take into account the variance in transit delay from the transmitter to the receiver. Thus, SNR still faces the problem of having to rely on reasonably accurate estimate of RTD.

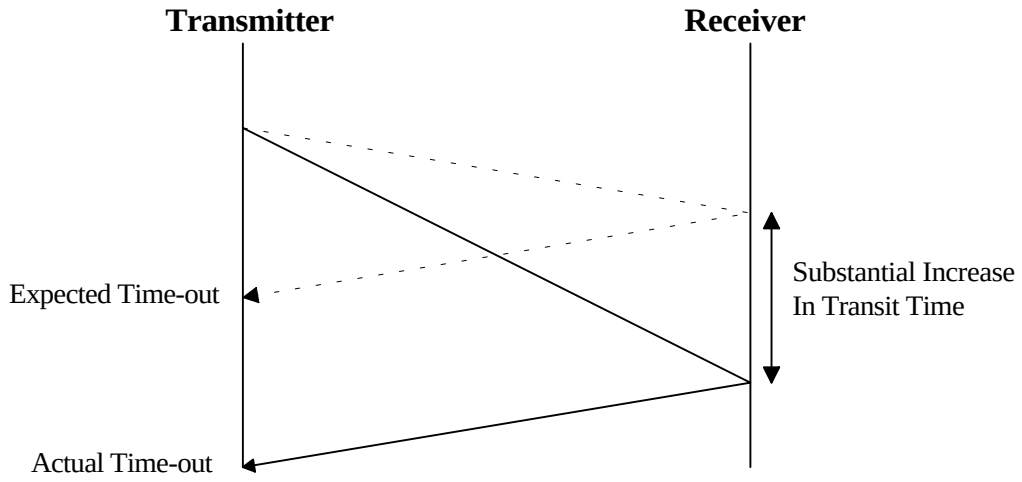


Figure 4 Transit Time Variance

In HTPNET, state exchange between end-points is initiated periodically only by the sender. The initiated state packet with synchronisation sequence S_n, effectively binds the preceding data packets that are sent between S_{n-1} and S_n. This state packet transverses the same path as the bound data packets, and thus is subjected to the same variance transit delay. In the event when the state packet S_n gets lost in the network, the next following state packet S_{n+1} automatically binds the data packets associated with S_n and S_{n+1}. This is achieved by comparing the echoed synchronisation value S_{n+1}, with transmitted data packets associated synchronisation value of S_{n+1} or lower. Thus, this technique exhibits a self-recovery property in that no spurious retransmission of data packets or time-slip can occur even if the corresponding state packet does get lost. This is in contrast with the timer mechanism, where unnecessary retransmission of data packets may be incurred due to lost of state packets.

In order to cater for the misordering of packets as experienced in a connectionless network, an offset can be added to the comparison mechanism. This is achieved by comparing the echoed synchronisation value S_n, with transmitted data packets associated synchronisation value of $S_{n\text{-}offset}$ as shown in Figure 5. This mechanism effectively eliminates the unnecessary retransmission of misordered data packets by delaying the comparison mehanism used for detection of lost packets. The offset value must be set based on the severity of the misordered packets in terms of the average time delay. For a connection-oriented network, which guarantees in-sequence delivery of packets, the offset value is set to zero. At the transmitter, only state packets with monotonically increasing received echoed synchronisation value are processed, while out-of-sequenced state packets are discarded.

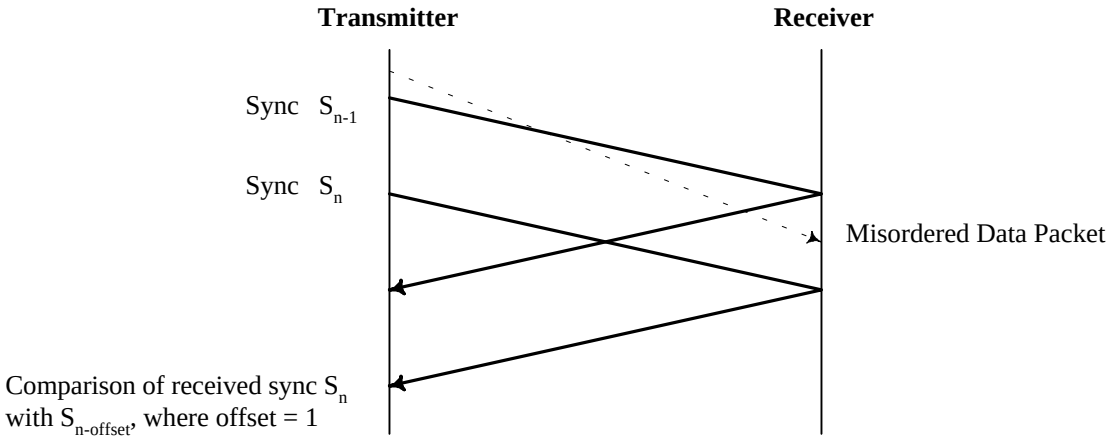


Figure 5 Misordered Data Packet

Another attractive property of HTPNET is in it's ability to adapt to the traffic conditions by subjecting the state packet to transverse the same path as the bound data packets. This adaptive property effectively eliminates completely the reliance on RTD to detect loss packets. As a result, HTPNET requires no prior knowledge of the network RTD for communication to begin between end-points. This is in contrast with TCP and SNR, where the retransmission timeout is normally set to two to three times the estimated round-trip delay[2]. This is necessary to take into consideration the possibility of excess delay due to congestion in the underlying network. Such RTD estimation often results in non-optimal timeout value being selected [8], thus affecting the overall throughput performance.

In TCP, the implementation of the delta-list [15] for efficient management of timers requires the list to be scanned whenever a timer is inserted or deleted[16]. In a high-speed network, with large bandwidth-delay product, this would incur considerable computational overheads in managing the list. Similarly, in SNR, for each state packet received at the transmitter, each entry in the timer list needs to be processed by decrementing the retransmission count for all unacknowledged blocks. Therefore, the computation overhead in managing retransmission timers for TCP and SNR is inherently dependent on window size. In contrast, HTPNET employs a computationally efficient and simple retransmission mechanism known as *FIFO*[3] *search*. Consider Figure 6, for each connection, a FIFO table containing two fields <*sync, seq*> is created, where *sync* contains the synchronisation value and *seq* contains the packet sequence number sent during the corresponding sync period. At the start of the data transfer, a group of data packets associated with synchronisation value 0 are admitted into the network. At the first periodic timeout, the first state packet with synchronisation value of 0 is transmitted. Subsequently, the *seq* of the packets sent, with the corresponding *sync* are appended to the head of the FIFO table. After each state transmission the synchronisation value is incremented and the process repeats. When the first synchronisation value of 0 is echoed back to the transmitter, only

---

[2]In TCP, the initial retransmission timeout is initialised to some initial value. Using an adaptive retransmission algorithm, this value is continually revised based on changing network delays.

[3]**F**irst **I**n **F**irst **O**ut

entries at the tail of the FIFO table with *sync* equal or less than the received synchronisation value are processed. Typically, the computation would require no more than processing of the group of data packets whose associated synchronisation value is equal to the echoed synchronisation value.

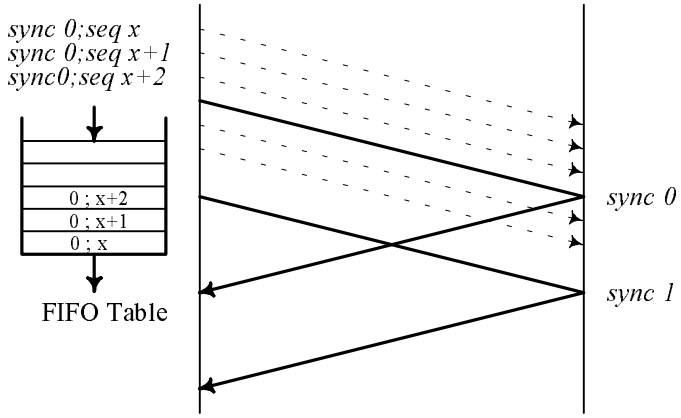


Figure 6 Basic Operation of FIFO Search

5. HTPNET IMPLEMENTATION

An experimental implementation of HTPNET has been constructed for a network of T800 transputers running at 20 MHz, with 20 Mb/s links. The protocol software is entirely written in Occam[17]. In the current experimental system, the major focus is on the achievable protocol processing capacity of HTPNET in terms of its ability to deliver packets across a high-speed network.

The process structure of the implementation model is shown in Figure 7. To simulate a high-speed network, a pipeline of processes have been configured to link the end systems. The network model simulates two parameters, namely, the RTD and packet error rate. This implementation of HTPNET consists of 10 parallel processes, 6 processes at the transmitter and 4 processes at the receiver. These processes and the network simulator communicate control packets using Occam channels.

HTPNET comprises four main control processes, namely:

- *Transmit data packet processor, Tx1*: Tx1 is dedicated to the management of the transmission of data packets into the network. As well as processing new data packets received from the host, Tx1 also manages the retransmission of data packets. It obtains the status of the transmitted packets by periodic communication with Tx2. Upon receiving a bit-map on the status of transmitted data packets from Tx2, based upon the bit-map contents, Tx1 either releases the data packet buffer or retransmits a data packet by placing the address reference into the appropriate FIFO buffers. In this way, Tx1 is relieved from almost all error recovery procedures, thus greatly simplifying the protocol processing. This is in sharp contrast with existing protocols,

where error recovery procedures are performed together with the main protocol processing functions.

- *Transmit state packet processor, Tx2*: Tx2 is dedicated to state synchronisation between the transmitter and receiver. In addition, it also manages the error recovery function. Periodically, Tx2 initiates a state synchronisation by transmitting a state packet to the receiver node. Subsequently, Tx2 communicates with Tx1 to obtain the latest information on the sequence numbers of transmitted data packets since the last periodical update. Upon receiving a returned state packet from the receiver, Tx2 initiates an error recovery function based on the latest information on the receiver's state. A bit-map on the status of the transmitted packets is then assembled and passed to Tx1 which retransmits error packets and releases data buffers accordingly.

- *Receiver data packet processor, Rx1*: Rx1 is dedicated to the processing of incoming data packets and ensuring in-sequence delivery of data. When a data packet arrives, Rx1 checks whether the packet sequence number corresponds to the expected sequence number. If the data packet is in-sequence, the address reference to the packet buffer is passed to the in-sequenced data FIFO for processing by the receiver host. On the other hand, if the data packet received is out of sequence, the data will temporary be stored in the packet store while waiting for the missing packet to be retransmitted. Periodically, Rx1 will be requested by Rx2 for the latest details on the received data packets. In doing so, a bit-map to reflect the current status of the received data packets is assembled by Rx1 and is passed to Rx2 for processing.

- *Receiver state packet processor, Rx2*: Rx2 is responsible for state synchronisation between the receiver and transmitter. In addition, Rx2 incorporates the selective retransmission mechanism for efficient error recovery. Unlike existing protocols like TCP and TP4, where an acknowledgment is sent after each received packet, HTPNET uses a block acknowledgment. After obtaining a bitmap on the status of the received packets from Rx1, Rx2 constructs the block acknowledgment by specifying a range of sequence numbers of packets that have been correctly received . The block acknowledgment is sent periodically in response to requests from the transmitter.

A number of features of Occam and the transputer helped make the protocol implementation efficient both in terms of programming effort and execution speed. Firstly the support in Occam for timers greatly simplified the implementation of periodic timing calculations. The low context switch overhead for processes on the same transputer also encouraged the use of concurrency within processors, leading to modular, well-engineered and reconfigurable software. Also, the ease with which a transputer network can be

reconfigured promoted experimentation with the software, enabling a thorough investigation of the implications of design choices in a truly parallel environment.

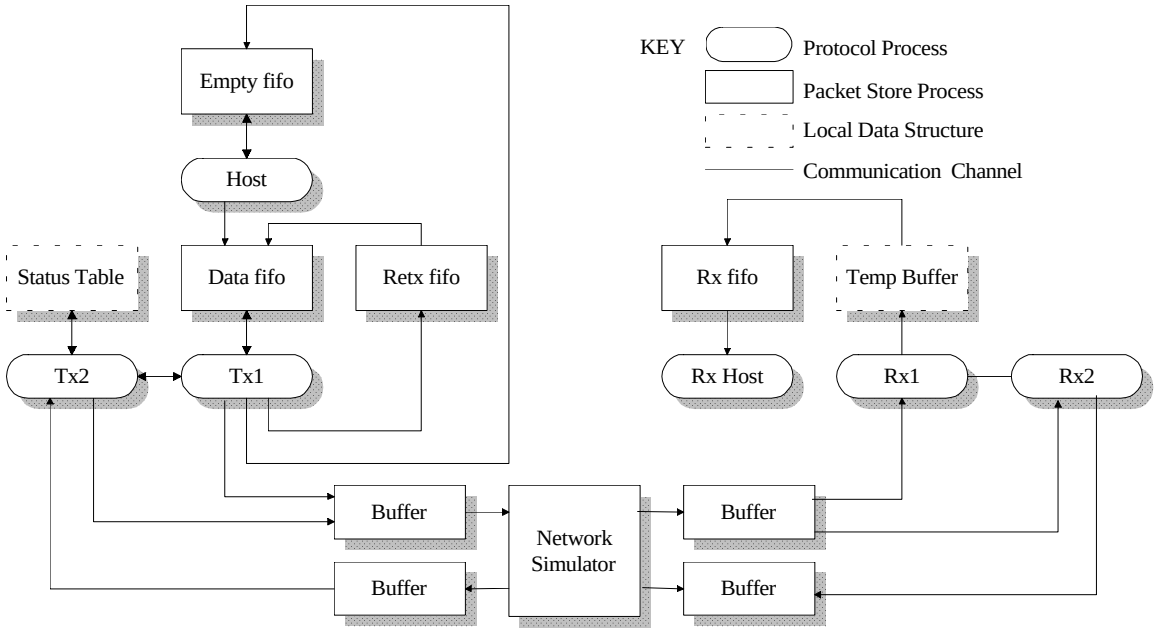


Figure 7 Occam Process Structure for Experimental HTPNET Implementation

6. PERFORMANCE RESULTS

The implementation model of HTPNET is shown in Figure 7. The corresponding performance results of HTPNET under different configurations from one to eight transputers are shown in Figure 8. This shows close to a five-fold increase in throughput for configurations ranging from one transputer to eight transputers. These results clearly demonstrate the advantage of exploiting a parallel architecture in the design of HTPNET. With a configuration of five transputers for protocol processing, a throughput performance of about 12,000 packets/s is attained[4]. Assuming data packets of 2 Kbytes length, this translates into an achievable data transfer rate of more than 200 Mb/s.

[4]The protocol processing rate obtained here does not take into the consideration of other bottlenecks, such as memory and bus bandwidth constraints, host/network interfaces and overhead in host operating systems. These issues are important for the overall successful implementation of a high-speed protocol processor. References can be found in [18][19].

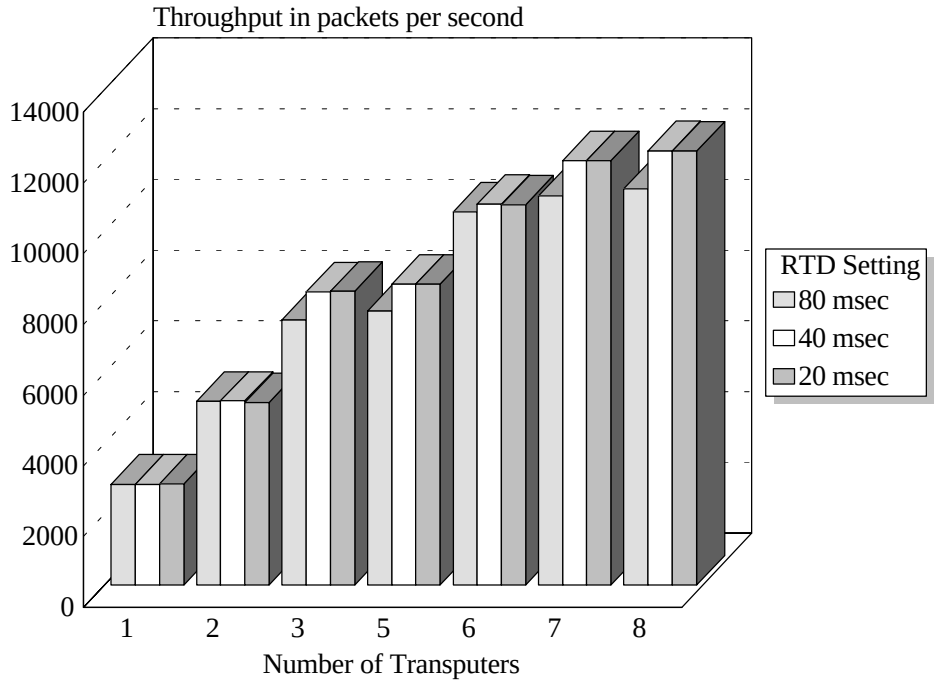


Figure 8 Performance of HTPNET

It is worth noting that in the eight transputer configuration, only five processors are used to perform core HTPNET protocol handling. The remaining three processors are used to implement a simulation of the high-speed network connecting the transmitter and receiver, including buffering of information at each end. The actual transputer configuration used is presented in Figure 9.

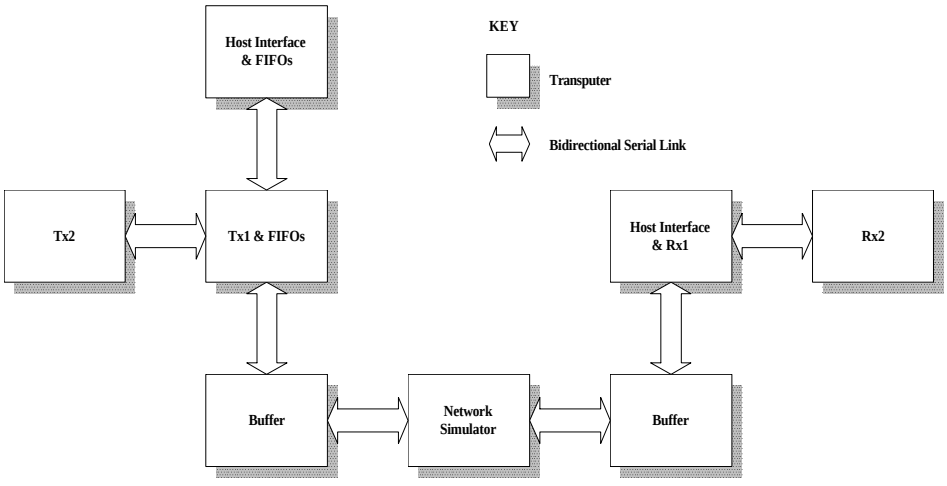


Figure 9 HTPNET Transputer Configuration

In order to verify the efficiency of the retransmission mechanism under error conditions, the network simulator can be programmed to drop packets at various rates while maintaining a constant window size. Figure 10 illustrates the performance of HTPNET when handling errors with different window sizes and RTD of 20 msec. The results

demonstrate that HTPNET's performance remains relatively stable and constant even under severe error conditions. Across different window sizes and error rates, the protocol's performance drops by no more than 25% before congestion collapse is experienced by the network.

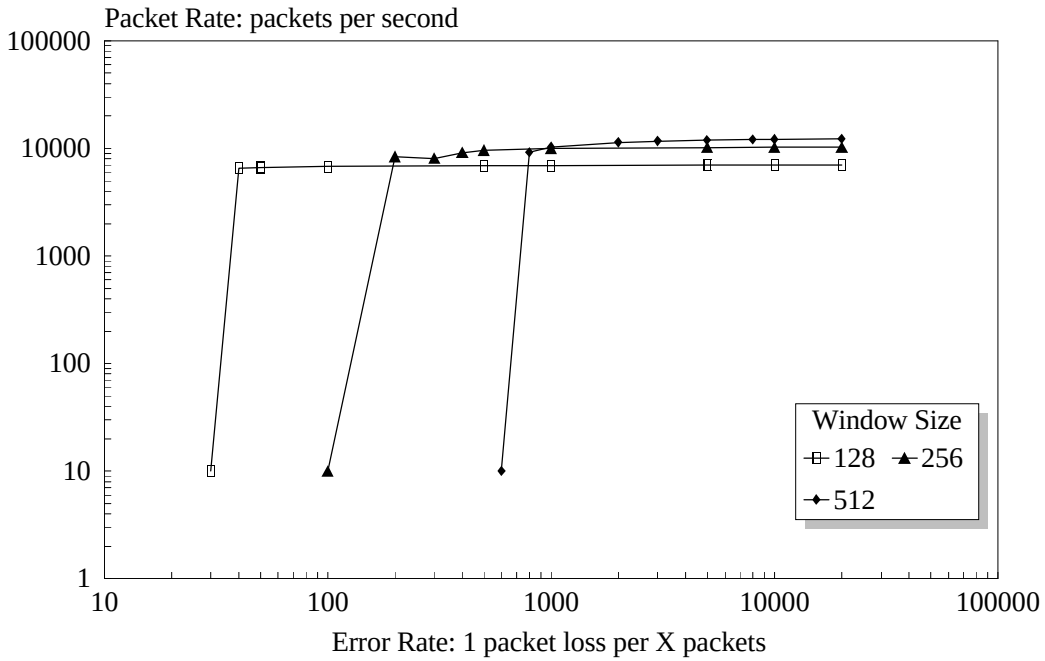


Figure 10 HTPNET Performance Under Error Conditions

To investigate the efficiency of FIFO search mechanism for operation in a high-speed environment, a version of HTPNET is compiled to perform full exhaustive search of the FIFO table. Figure 11 illustrates the advantage of a FIFO search[5]. Under small window size, with corresponding small FIFO table, the throughput performance for both protocols is almost identical. However, as the window size continues to increase, the advantage of employing FIFO search mechanism begins to be evident. With window size of 2048, a throughput performance gain of more than 75% is recorded using FIFO search.

[5]Experiment conducted on three transputers configuration: one transputer for transmitter processes, one transputer for network simulation and one transputer for receiver processes.

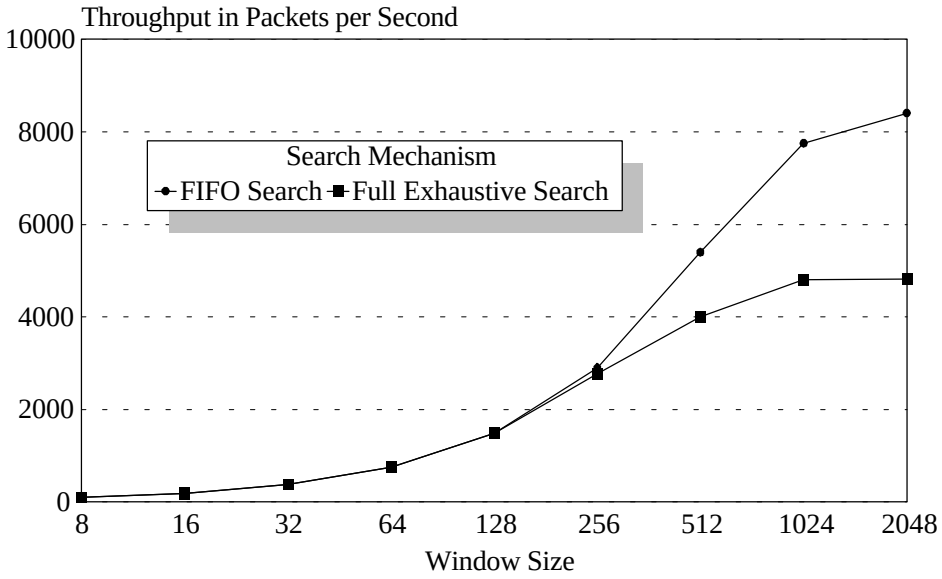


Figure 11 HTPNET Performance Under Different Search Mechanism

Shown in Figure 12 is the performance of HTPNET under congested condition. The throughput increases almost linearly with the offered load under congestion free condition. As the load continues to increase, congestion starts to build up in the network. In the uncontrolled case, the network will eventually experience a congestion collapse due to excessive loss of packets in the network. In contrast, HTPNET designed with PTIF congestion control is able to stabilise the throughput to a constant packet rate, thus avoiding a congestion collapse at the network.

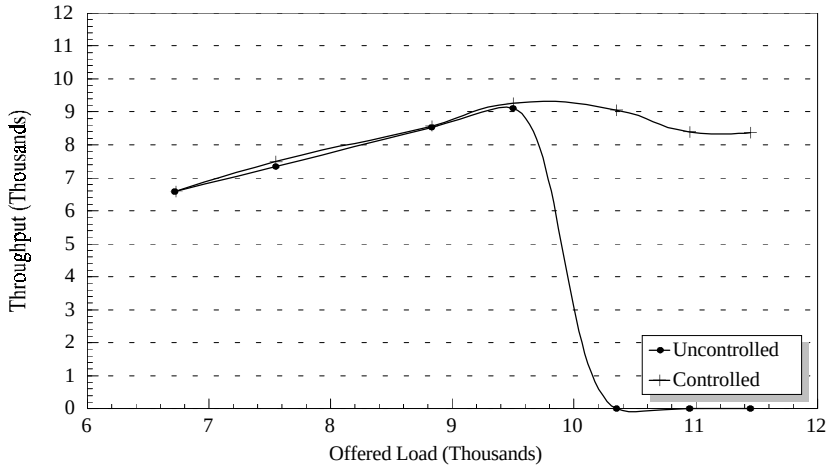


Figure 12 Performance of PTIF Congestion Control Strategy

7. COMPARISON WITH TCP

In order to verify the potential of HTPNET, a comparative study has been conducted to compare HTPNET with TCP for operation in a high-speed environment. An experimental implementation of TCP has been constructed to run on a network of transputers. To ensure fair and accurate measurement, only three transputers are used for each protocol

implementation: one transputer for transmitter processes, one transputer for network simulation and one transputer for receiver processes. In addition, the implementation of TCP protocol is optimised as much as possible to avoid unnecessary overhead in protocol processing. For example, instead of using explicit timers for each transmitted packet, a delta-list[15] is employed for efficient implementation of timers. An adaptive retransmission algorithm [10] is used to compute the timeout for the retransmission timers using the following formulae:

$$RTT = (\alpha * \text{Old RTT}) + ((1-\alpha)*\text{New RTT Sample})$$
$$\text{Timeout} = \beta * RTT$$

where α is set to 0.9 and β is set to 2.0. Further, the implementation includes Karn's algorithm and timer backoff[20][10], where the multiplicative factor, γ is set to 2.

Figure 13 shows the throughput performance of HTPNET and TCP as a function of window size for a RTD setting of 40 msec. Under small window sizes, the performance for both protocols is almost identical. However, as the window size increases, the throughput for TCP begins to saturate at a much faster rate. In contrast, HTPNET throughput continues to increase almost linearly with the window size. This upward trend continues until some saturation point, when the window size is large enough to keep the pipeline in the network full, thus, maximising the throughput. Beyond the saturation point, the throughput remains relatively constant regardless of the window size. This result verifies the insensitivity of HTPNET protocol processing to varying window size due to the efficient FIFO search mechanism.

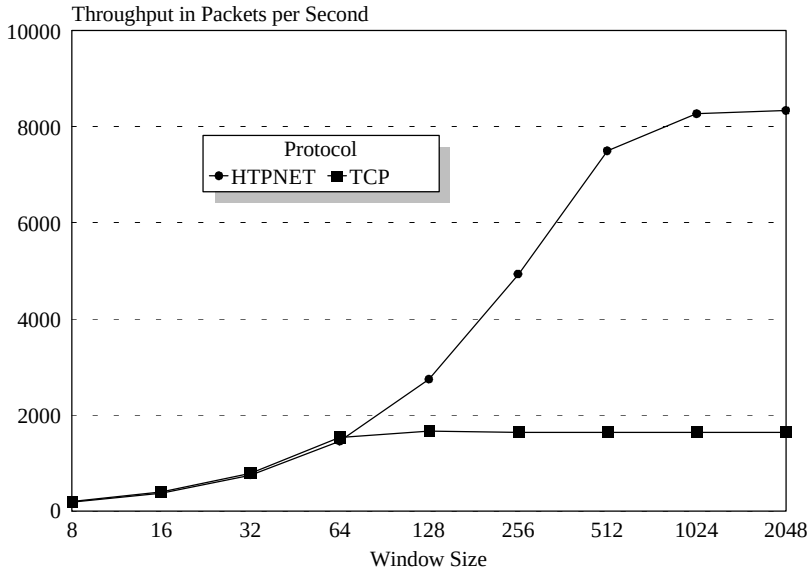
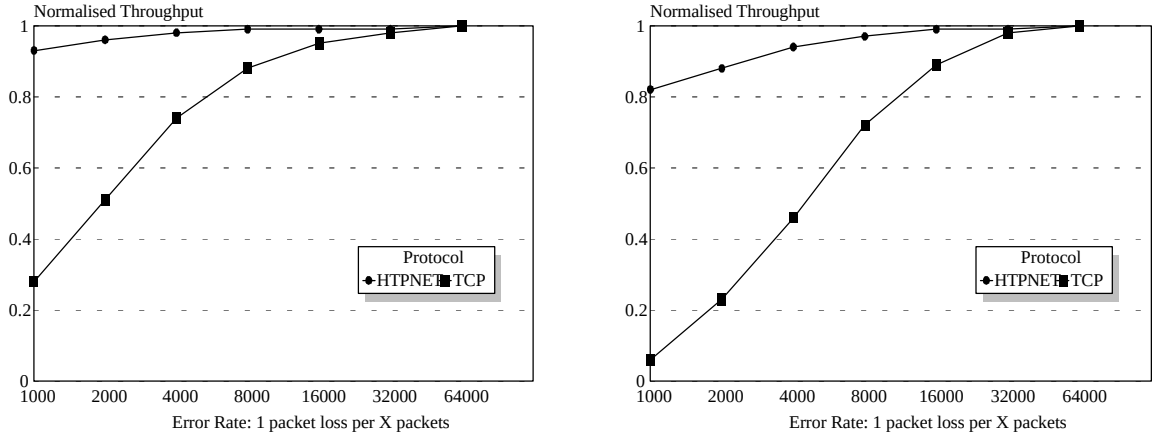


Figure 13 Throughput Performance of HTPNET and TCP

Figure 14a, b show the throughput performance of HTPNET and that of TCP as a function of error rate. The throughput data are normalised[6] to provide easy analysis of the overall comparison. In the case of low error rates, the performance for both protocols is identical. However, as the error rate increases, the throughput for TCP decreases rapidly. In contrast, HTPNET throughput remains relatively stable with slight gradual decrease in throughput. The difference is more acute as round-trip delay increases, with TCP throughput decreasing to almost zero under extreme error condition when RTD is set to 40 msec. This result clearly indicates the efficiency of HTPNET error recovery mechanism for operation in a high-speed network with high error and congestion losses.



a. RTD set to 20 msec b. RTD set to 40 msec

Figure 14 Throughput Comparison Under Error Condition

8. CONCLUSIONS

In this paper, we have examined the evolving design requirements of future transport protocols for operation in a high-speed network environment. In response to these requirements, the architecture of a new lightweight transport protocol known as HTPNET has been presented. HTPNET is designed to exploit highly parallel computing architectures, in which the protocol processing is decomposed into several concurrently operating finite state machines. Each of these finite state machines is responsible for different control functions, and the finite state machines act collectively by synchronising periodically using control messages.

In addition, the HTPNET design takes into consideration the changing characteristics of high-speed networks. HTPNET uses an out-of-band signalling system based upon transmitter-paced periodic exchanges of state information between end systems. This mechanism exhibits several attractive properties which have been demonstrated to perform efficiently in a high-speed environment with high bandwidth-delay-product. The self-recovery property enables the sender to recover from lost state packets without incurring spurious retransmission of successfully received data packets. Further, the adaptive

[6]Throughput is normalised by dividing the throughput with the maximum throughput recorded for the respective protocol.

property ensures that HTPNET is insensitive to time variance in transit delay, thus, removing the reliance on RTD to detect lost packets. In addition, a computationally simple and efficient mechanism known as FIFO search has been devised for the management of retransmission. FIFO search has been shown to perform effectively even for large bandwidth-delay-product networks. Typically, the computation would require no more than processing of the group of data packets whose associated synchronisation value is equal to the echoed synchronisation value. Error recovery is achieved using the selective retransmission technique. Selective retransmission obviates the need for retransmission of successfully received packets, and utilises block acknowledgments to provide efficient use of network bandwidth.

The experimental implementation of HTPNET on a network of transputers has demonstrated the advantage of exploiting a parallel architecture for protocol processing. On a network of five T800 transputers dedicated to protocol processing, HTPNET is able to deliver a packet transfer rate of about 12,000 packets/s. This translates into a data transfer rate of more than 200 Mb/s assuming a packet size of 2K bytes. Importantly, the scalability of parallel architectures makes it feasible to add more processors to support new protocol functions, such as adaptable congestion control algorithms, without impacting on data processing performance. It should further be noted that advances in microprocessors designed for parallel systems offer even greater performance potential for protocol processing systems. For example, the release of the 50 MHz T9000 transputer with 100 Mb/s links should give a ten-fold increase in speed over the T800[21]. We believe that this development could further enhance the throughput of HTPNET, with potential to deliver over the gigabits range.

REFERENCES

1. W. Doeringer, D. Dykeman, M. Kaiserswerth, B. Meiser, H. Rudin and R. Williamson, "A Survey of Light-weight Transport Protocols for High-speed Networks," *IEEE Trans. Commn.*, vol 38, pp. 2057-2071, Nov 1991.

2. W. S. Lai, "Protocols for High Speed Networking," *Proc. of INFOCOMM '90, Panel Discussion* (June 1990), pp 1268-1269.

3. W. R. Bryne, "Broadband ISDN Technology and Architecture," *IEEE Network*, Jan 1989, pp 7-13.

4. M. Zitterbart, "High-Speed Transport Components," *IEEE Network Magazine*, Jan 1991, pp 54-63.

5. G. Chesson, "The Protocol Engine Project," *Unix Review*, Sept 1990.

6. David D. Clark, Van Jacobson, John Romkey, Howard Salwen, "An Analysis of TCP Processing Overhead," *IEEE Commn. Magazine*, pp. 23-29, June 1989.

7. H. Kanakia and D.R. Cheriton, "The VMP Network Adapter Board, High-Performance Network Communication for Multiprocessors," *Proc. ACM SIGCOMM Symp.: Communications, Architecture and Protocols*, 1988.

8. L. Zhang, "Why TCP Timers Don't Work Well," *Proc. ACM SIGCOMM Symp. on Commn., Architectures, and Protocols,* Stowe, VT, 1986, pp 397-405.

9. Van Jacobsen, "Congestion Avoidance and Control", *Proceedings of the ACM SIGCOMM'88*, Aug. 1988, Standford CA, pp.314-329.

10. D. E. Comer, "Internetworking with TCP/IP: Principles, Protocols and Architecture", Englewood Cliffs, NJ: Prentice-Hall, 1988.

11. R. A. Donnann, "Method and System for Retransmitting Incorrectly Received Numbered Frames in a Data Transmission System," U.S. Pat. 439859, Mar. 1984.

12. A. N. Netravali, W. D. Roome, and K. Sabnani, "Design and Implementation of a High Speed Transport Protocol", *IEEE Trans. on Commn.*, vol. 38, no. 11, pp 2010-24, Nov 1990.

13. Doshi B.T., H.Q. Nguyen, "Congestion Control in ISDN Frame-Relay Networks," *AT&T Technical Journal*, Vol. 67, Nov-Dec, 1988, pp. 36-46.

14. D. C. Feldmeier, E. W. Bierack, " Comparison of Error Control Protocols for High-Bandwidth-Delay Product Networks", *Protocols for High-Speed Networks, II*, IFIP, North-Holland Publisher, 1991, pp 271-295.

15. D. E. Comer, D. L. Stevens, "Internetworking with TCP/IP Vol 2: Design, Implementation, and Internals", Englewood Cliffs, NJ: Prentice-Hall, 1991.

16. D. C. Feldmeier, "A Survey of High Performance Protocol Implementation Techniques", in *High Performance Networks - Technology and Protocols*, Chapter 2, Ahmed Tantawy editor, Kluwer Academic Publishers, 1993.

17. "Occam 2 Reference Manual," Prentice Hall, INMOS Limited, 1988.

18. B. S. Davie, "The Architecture and Implementation of a High-Speed Host Interface", *IEEE Journal on Selected Areas of Communications*, vol. 11, no. 2, pp 228-239, Feb 1993.

19. K. K. Ramakrishnan, "Perfromance Considerations in Designing Network Interfaces", *IEEE Journal on Selected Areas of Communication*, vol. 11, no. 2, pp 203-219, Feb 1993.

20. P. Karn and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", *Proceedings of ACM SIGCOMM '87*, pp 2-7, Aug 1987.

21. SGS-THOMSON, "The T9000 Transputer Products Overview Manual", first edition 1991.