1983

The Proposed DARPA IP-to-X.25 Interface Standard: Performance Optimization with Multiple Circuits

Douglas E. Comer
Purdue University, comer@cs.purdue.edu

John T. Korb
Purdue University, jtk@cs.purdue.edu

Report Number:

84-473

The Proposed DARPA IP-to-X.25 Interface Standard: Performance Optimisation with Multiple Circuits*

Douglas Comer
John T. Korb

CSD-TR-473

October 1983
Revised May 1984

Computer Science Department
Purdue University
West Lafayette, IN 47907

ABSTRACT

The Defense Advanced Research Projects Agency recently proposed guidelines for interfacing the Department of Defense Internet Protocol (IP) to public packet-switched networks that use the CCITT X.25 protocol. This paper briefly reviews the problem, and gives the details of an implementation that adheres to the guidelines. It describes experiments with the interface software that isolated a serious bottleneck, and shows how performance can be improved dramatically by multiplexing traffic over multiple virtual circuits. Finally, it describes a model of the underlying network that can be used to accurately predict saturation, a point at which increasing the number of virtual circuits increases the cost without increasing throughput.

* This work was supported by the National Science Foundation under contract MCS-8109322.

1. Introduction

The Defense Advanced Research Projects Agency (DARPA) has been experimenting with long-haul networks for more than a decade [3]. It has built and maintained a long-haul network, popularly called the ARPANET. Recently, DARPA has been studying ways to interconnect several networks into a unified Internet. Although individual component networks may use different transmission media, the collective entity functions as a whole, allowing any pair of computers that connect to it to exchange information with a single transport-level protocol.

Standard protocols provide the key to uniformity in the Internet. Individual computers connected to the Internet are called hosts. Each host must use the standard transport-level protocol, which is divided into two layers [11]. The upper layer, the Transmission Control Protocol (TCP), is called by applications programs like mail and file transfer routines. It establishes connections to the destination host, and provides end-to-end reliable data transfer (e.g., uses a timer to judge whether the destination machine is responding, and informs the sender if it is not). The lower layer, the Internet Protocol (IP), provides an (unreliable) datagram service. It receives datagrams from the upper layer and routes them onto the appropriate network based on their destination address. Most importantly, the IP layer understands the topology of the Internet well enough to route datagrams even if the ultimate destination is on a network that does not connect directly to the host computer. In such cases, IP routes the datagram through a locally available network to a machine called a gateway that will forward the datagram toward the destination net.

Uniformity in the Internet is achieved merely with protocols—individual component networks use a variety of transmission technologies and host-to-network interface mechanisms to transport data. For example, the ARPANET component transfers data over leased wires. Another component uses satellite links. Thus, each component has its own hardware-dependent low-level protocol beneath the standard IP layer. All that is important is that the lowest level accepts datagrams from one host and delivers them to the specified target host. Previous work on the Internet has explored low-level links that use radio [9], satellite [7], and fiber optic [1,87] connections. (If both hosts do not use the same end-to-end protocol, some form of protocol translation is required. Das and Cole [5] describe some of the problems inherent in this technique.)

At Purdue we have been considering the question of how to connect hosts to the DARPA Internet using public packet-switched networks like GTE Telenet to provide the lowest-level links. Because public carriers use the X.25 protocol [2], the question reduces to finding a way to layer IP over X.25. Previous work [4] described the motivation for our design, and gave preliminary measurements of its performance. The initial positive results led DARPA to propose a standard for IP-to-X.25 layering [10].

Although our initial design demonstrated feasibility, its performance was puzzling. Running on an otherwise idle Digital Equipment Corporation VAX 11/780, the initial IP-to-X.25 interface could transfer user data approximately 23% as fast as the X.25 interface hardware (a 9600 baud HDLC/LAPB connection). Unable to explain why the software was not running the low-level link at or near its capacity, we undertook a series of experiments to identify the bottleneck. This paper reports on those experiments, and the resulting optimizations. We note that these optimizations fall within the guidelines of the proposed standard. Thus, they should be viewed as suggestions for implementors, not suggestions for changes in the proposed standard.

2. Simple IP-to-X.25 Interface

This section reviews the methods and results given in [4] for simple transmission of IP datagrams over X.25-based networks.

2.1 IP Datagrams and X.25 Virtual Circuits

The X.25 protocol used by public carriers provides point-to-point communications with logical connections called virtual circuits. To transfer data over an X.25 network, the sender first opens a virtual circuit to the destination. Opening a circuit implies negotiating with the network, and indirectly, with the destination machine; it is both expensive and time-consuming. After the circuit has been established, data can be transferred in either direction. Compared to the cost of opening a circuit, the cost of transfer is inexpensive. Thus, the total cost of high-level operations like file transfer is minimized by minimizing the number of open operations that must be performed on the underlying X.25 network. However, only a limited number of virtual circuits may be open simultaneously by a given host, so circuits must be closed when no longer needed.

Unlike X.25, IP deals only with datagrams—it does not use the notion of virtual circuits. Each datagram contains both the data to be transferred, as well as the destination address. To use an X.25 network as the lowest layer, an interface must be inserted between IP and the X.25 network to manage X.25 virtual circuits. The interface opens a virtual circuit when one is needed, and closes circuits that are no longer in use. Managing such circuits is awkward at best, because the interface software has no way of knowing whether more datagrams will be sent to the same destination or whether the circuit is no longer needed.

2.2 Managing Circuits to Optimize Cost and Throughput

The IP-to-X.25 interface attempts to maximize throughput and minimize cost. For example, current tariffs charge heavily for opening a virtual circuit. Opening a virtual circuit also consumes time because the sender must wait for the destination to accept the call before data can be sent (ignoring fast select). Thus, throughput is maximized and cost minimized by reducing the number of times a circuit must be opened.

Because minimizing opens reduces cost and maximizes throughput, one might expect that the interface software would open circuits as needed, and then leave the circuit open even if there was no traffic. Other constraints prohibit leaving idle circuits open. First, some networks charge for circuits when they are idle. Second, a given host can have only a fixed number of virtual circuits open at a given time.

If all circuits are in use, the interface must decide when to close one call to make way for a new one.

The proposed IP-to-X.25 standard [10] does not specify how the interface makes decisions about when to close circuits. The obvious algorithm for circuit management, taken from page replacement algorithms in virtual memory systems [6], employs a Least Recently Used (LRU) strategy in which the circuit that has been idle longest is closed whenever a new circuit is needed. Comer and Korb [4] show that LRU exhibits anomalous behavior under heavy loads, however, and propose a Modified Least Recently Used (MLRU) algorithm in which a tuning parameter, T_{min} controls circuit closing. According to MLRU, a circuit with longest idle time is candidate for closing only if it has been open at least T_{min} time units. Furthermore, MLRU reserves K circuits for incoming calls. Typically, $K = 2$.

3. Experiments to Analyze the Performance Bottlenecks

An early implementation of the MLRU interface was reported in [4]. Although it performed correctly, performance was disappointing. Throughput of user data, measured by the DARPA File Transfer Protocol (FTP), indicated a maximum transfer rate of less than 2200 bits per second, or about 23% of the capacity of the line linking our host computer to the X.25 network. It was easy to speculate about the performance bottlenecks, but difficult to determine the limits on throughput more precisely. Four causes were suspected: overhead in the host protocol software (FTP, TCP, IP, and the IP-to-X.25 interface); insufficient CPU power in the X.25 device; delays introduced by the X.25 network; and the interactions among higher-level protocols, the IP-to-X.25 policies, and X.25 protocols. Host software overhead was ruled out by measuring the amount of CPU used on an otherwise empty machine.

Testing the X.25 device was more difficult. The obvious technique consists of connecting two machines back-to-back to eliminate interference from the X.25 network. A somewhat less satisfactory solution consists of connecting the X.25 device in a self loop, so that transmitted data comes back to the same machine. Both of these tests are difficult to make because X.25 is not a symmetric protocol—the "network" side behaves differently than the "host" side. In X.25 terminology, responses from the Data Circuit-terminating Equipment (DCE) are distinct from requests by the Data Terminal Equipment (DTE). Thus, is it not possible to connect two X.25 hosts without a network between them. To test the capacity of the X.25 device, we first had the manufacturer modify the X.25 Level 2 (frame) protocol, making it symmetric, and then ran the board in loop-back mode.

Data from the loop-back test indicated that the network, not the X.25 device was limiting throughput. The test was inconclusive, however, because it relied on a modified protocol and a single host. Opportunity for a more definitive measurement arose when Purdue agreed to test an experimental X.25 connection to the ARPANET as part of work conducted by Bolt, Beranek, and Newman for DARPA. Measurements of the experimental connection showed that even with protocol overhead, the X.25 board was capable of transferring user data faster than 10,000 bits per second (when the line speed was increased). Subsequent experiments concentrated on measuring delays on the public X.25 network, and finding ways to improve performance.

A significant insight into the problem came from the observation that the X.25 network being used (GTE Telenet) allowed at most two unacknowledged Level 3 packets on a given virtual circuit (presumably to avoid network congestion). This limit is called the window size. We hypothesized that when presented with a queue of packets to send, the underlying X.25 network would exhibit *burst* behavior, sending two packets and then waiting for an acknowledgement. It turned out that the burst-mode behavior and network delays limited throughput severely. To understand the throughput limit, it was necessary to develop a model of the network behavior that abstracted away unnecessary details.

4. A Model of Burst-Mode Behavior

Figure 1 gives our model for burst behavior. As in the figure, let T_N denote the mean time to transmit a packet across the network, let T_p denote the time to transfer a packet onto the network, and let T_f denote the queueing and protocol delays introduced by the frame level (e.g., frame acknowledgement delay). Note that the receiver does not acknowledge every X.25 packet immediately. It sends an acknowledgement when the window is full (for Telenet, the window size is two), or after an extremely long time elapses.

A bound on the maximum data transfer rate can be computed from the burst model. Assuming that packet sizes are fixed, and that the time required to traverse the network is constant, the data rate is given by the ratio of the size of a burst and the time between bursts:

$$\textit{data rate} \le \frac{\textit{data in a burst}}{\textit{time between bursts}}.$$

For a burst of two packets, the bound is:

$$\begin{aligned}\textit{data rate} &\le \frac{2 \times \textit{packet size}}{\textit{time between bursts}}\\ &= \frac{\textit{packet size}}{0.5 \times (T_p + T_f + T_p + T_N + T_f + T_N)}\\ &= \frac{\textit{packet size}}{T_p + T_N + T_f}. \qquad (1)\end{aligned}$$

To check the validity of this bound, we computed T_p, T_N, and T_f for our implementation and compared the predicted bound to the observed data rate.

Computing T_p is straightforward. The IP-to-X.25 interface breaks each datagram (typically 492 bytes) into X.25 packets, making the average packet size 123 bytes. For X.25 packets that contain 123 characters of data and 7 characters of header, 8 bit characters, and a 9600 bps network interface,

$$\begin{aligned}T_p &= 130 \times 8/9.6\\ &= 108\,\text{ms}.\end{aligned}$$

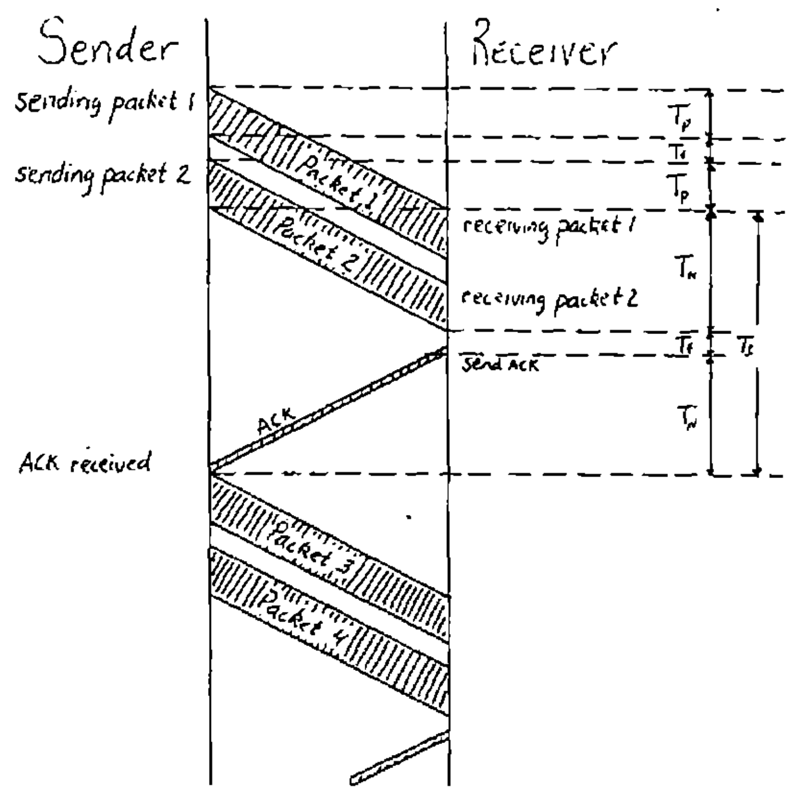

Figure 1. Burst mode model. Time proceeds down the page. Sender events are shown on the left, receiver events on the right, and network events in the middle.

Estimating T_N is not as simple. Figure 2 shows how, for the public network we use, T_N varies dramatically with network load. The data plotted consists of throughput rates obtained by transferring a 100,000 byte file between Purdue University in Indiana and the The Rand Corporation in California each hour over a 24-hour period.[1] The highest throughput, which occurred during non-business hours, is nearly double the worst, which occurred during business hours, so we know that T_N depends directly on network load. Nevertheless, to simplify the model we will assume that T_N is constant, and use the mean network delay as reported by the vendor:

$$T_N = 235\,\text{ms}.$$

The measured value of T_f is approximately 15 ms. Thus, the burst model predicts that the X.25 network data rate should be bounded by:

$$\begin{aligned} \textit{data rate} &\leq \frac{1024}{108 + 235 + 15}\,\text{bps} \\ &= 2860\,\text{bps}. \end{aligned} \tag{2}$$

[1] This experiment was done with three virtual circuits open simultaneously to improve throughput. See the next section for details.

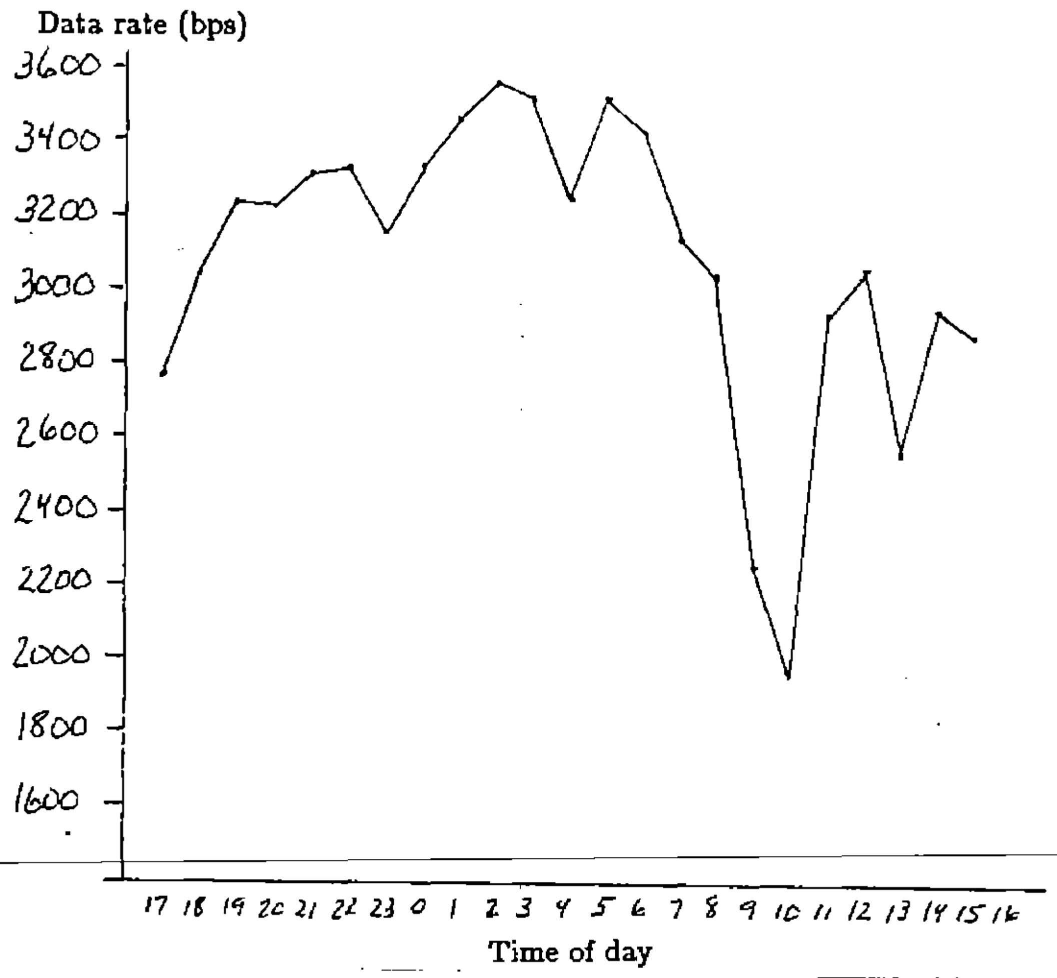

Figure 2. Graph of data rate versus time of day showing variation in throughput of X.25 network (using three open virtual circuits).

In practice, the rate at which user data can be transferred is even lower than the bound given by (2). First, in addition to user data, the X.25 network must transfer the TCP and IP headers associated with each datagram. Taking into account these headers makes the bound on the rate of user data transfer 2631 bps. Second, long network delays can trigger retransmission of datagrams by TCP, increasing the volume of traffic without increasing the amount of user data transferred. (In several tests, over 10% of the datagrams were retransmitted). Third, the IP-to-X.25 interface does not fill every X.25 packet completely (the proposed standard prohibits the interface from using the space at the end of a packet to send the beginning of a new datagram). Fourth, the network throughput varies dramatically during the day, even though we have assumed constant delay. Such overhead and variations in network performance easily explain why the observed rate of user data transfer has never exceeded 2200 bps (on a single virtual circuit) even though the model predicts an upper bound of 2631 bps.

5. Adding Multiple Connections to an MLRU Interface

How can throughput be increased? The burst-mode model predicts that the X.25 link will be idle[2] much of the time. This suggests that, in the absence of other traffic, the interface could increase throughput by opening multiple virtual circuits to a given site, and multiplexing datagram traffic to that site over all circuits available.

Using multiple vitual circuits changes a basic property of the transport system because X.25 does not guarantee that the order of delivery will be preserved on independent circuits. Fortunately, IP does not rely on datagrams being delivered in the order they are sent. Hence, the IP-to-X.25 interface can send packets round-robin, using all the virtual circuits that are open to a given site without expending effort to reorder them at the receiving end (of course, the TCP module at the receiving end will have to reorder them, but it is already prepared to do this task).

What is a reasonable bound on the number of virtual circuits that should be opened? The burst-mode model gives an upper bound for the case where all connections go to a single site. Consider Figure 1. Assuming no other circuits are open, the interface will be idle from the time that packet two is sent until the time the acknowledgement arrives. We call this the inter-burst delay, T_I. For a window of two packets, we have that

$$T_I = 2T_N + T_f.$$

Note that T_N is the cross-network delay time, so $2T_N$ is essentially the round-trip network delay.

During the inter-burst delay on a given virtual circuit, the X.25 device is free to transmit packets on another circuit. According to the model, interleaving two circuits will double the throughput. With three virtual circuits, the throughput is tripled. Throughput can be increased until saturation is reached. Saturation is the condition in which the underlying X.25 interface is completely busy.

How many additional circuits can be opened? Let C_{max} denote the minimum number of multiple circuits that produces saturation. From Figure 1, we can see that C_{max} is constrained by the inter-burst idle time and the time to transmit a burst. On a network with window size W,

$$C_{max} = 1 + \frac{T_I}{W(T_p + T_f)}.$$

For example, a network like GTE Telenet, which has T_p and T_f as defined

[2] HDLC/LAPB is a synchronous protocol, so the line is never idle; we refer to packet traffic.

above and window size $W = 2$,

$$\begin{aligned} C_{max} &= 1 + \frac{2T_N + T_f}{2(T_p + T_f)} \\ &= 1 + \frac{2 \times 235 + 15}{2 \times (108 + 15)} \\ &= 1 + \frac{485}{246} \\ &= 3\,\text{circuits}. \end{aligned}$$

To test this prediction, we measured throughput on GTE Telenet using from 1 to 13 virtual circuits. Figure 3 reports the results of these measurements. The experiment consisted of sending a 100,000-byte file over GTE Telenet from Purdue University to The Rand Corporation. No other X.25 traffic was sent by either host during the test. The throughput rate in bits per second was computed by dividing the number of bits in the data file (800,000) by the time in seconds for the transfer. The time includes retransmission of delayed data, but does not include the time required to establish the virtual circuits (i.e., required virtual circuits were opened before each test began). As the figure shows, the burst model accurately predicts saturation at three virtual circuits.

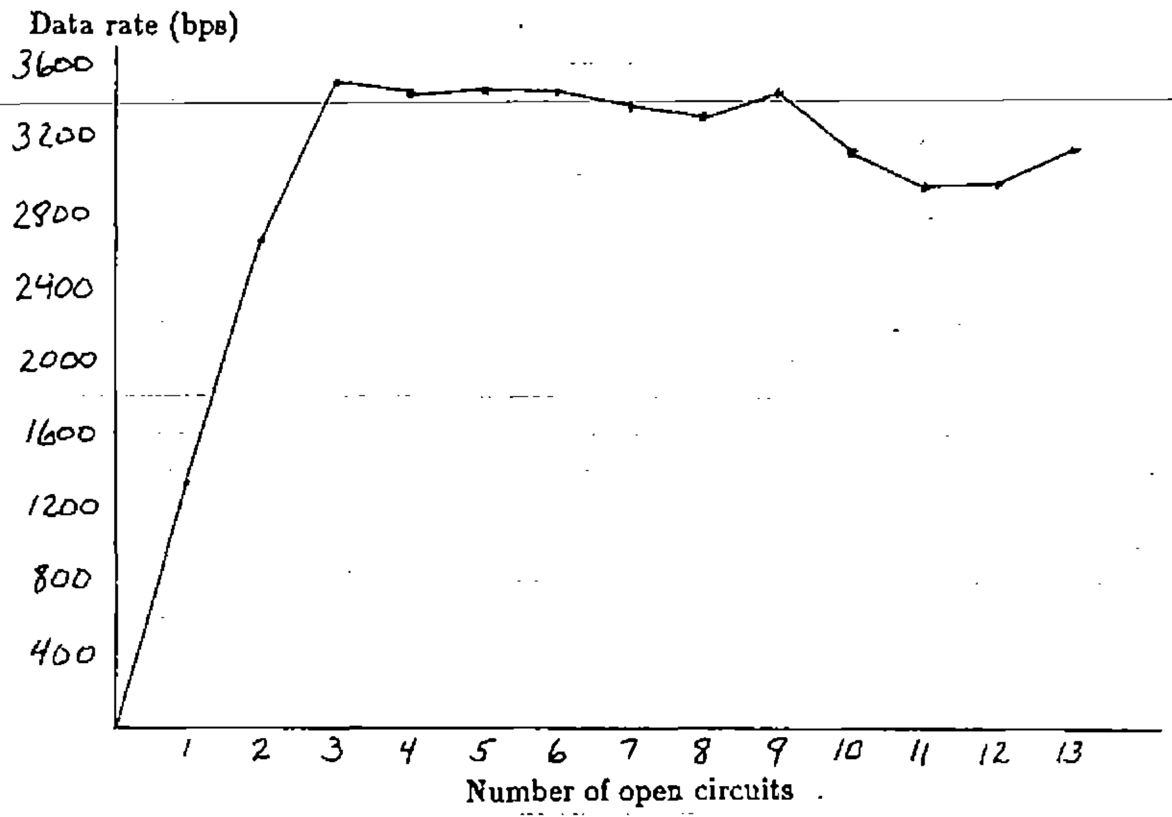

Figure 3. Graph of data rate versus number of open virtual circuits.

6. Enhanced IP-to-X.25 Interface

We have incorporated the multiple virtual circuit optimization into our MLRU IP-to-X.25 interface algorithm. Changing the MLRU algorithm to handle multiple virtual circuits involves choosing between increased cost (e.g., for opening an additional circuit) and increased throughput. Because the number of virtual circuits is limited, increasing the throughput to a given site may require closing a circuit to another site. Care must be taken to avoid instability called thrashing in which a set of sites that dynamically compete for circuits continually close each other's circuits when they attempt to increase their throughput.

The details, given below, explain how the circuit manager chooses when to open multiple connections. A description of potential problems, including consideration of gateway performance, follows the discussion of implementation heuristics.

6.1 Technical Details

When IP passes a datagram to the X.25 interface software, it also passes a destination Internet address as well. If the interface is acting as a gateway, the ultimate destination address found in the datagram may differ from the address passed to the interface. Therefore, the interface does not examine the datagram, but chooses a route based on the address passed from IP. Throughout this discussion, the term "X.25 destination address" will refer to the address specified by IP.

When the interface receives a datagram, it enqueues it in a queue associated with the destination X.25 address. In practice, the queue is divided into two pieces; one for small datagrams (typical of a terminal session) and one for large datagrams (typical of a mail or file transfer). All small datagrams are transmitted before any large datagrams, giving them priority over file or mail transfers. (Without this minor optimization terminal sessions become intolerably slow during a file transfer.)

Associated with each queue of outgoing datagrams is a set of X.25 virtual circuits currently open to that destination. In the software, virtual circuits are referred to by *channel numbers*; the terms are used interchangeably throughout this paper. There is a per-site parameter, C_{max}, that defines the maximum number of virtual circuits that the interface should open to that site. (Note: C_{max} need not be symmetric, so more than C_{max} circuits may be opened if the destination chooses to do so.)

The interface uses two timing parameters, T_{min} and T_{open}, to reduce the probability of thrashing when the demand for circuits exceeds the capacity of the X.25 interface hardware. T_{open} gives the minimum time the interface must wait before opening another channel to a given destination; T_{min} gives the minimum time a connection must be open before the interface will close it.

T_{open} adds hysteresis to prevent the interface from opening many circuits to a single site too quickly. The time required to open a circuit can be large compared to the arrival rate of datagrams. Without this parameter, the interface might attempt to open many circuits before any open request completes. Under increased load, the remote site may attempt to open multiple circuits as well. Thus, keeping T_{open} nonzero avoids opening additional circuits in cases where traffic to a single host increases for a short time.

As in [4], the minimum connection parameter, T_{min}, also adds hysteresis to the interface. It prevents closing a circuit that was just opened.

6.2 Implementation

Comer and Korb [4] contains interface procedures for MLRU without multiple circuits. They are the basis for the following set of procedures, which include heuristics for multiple circuit management. At the highest level, *output_interface* accepts datagrams and enqueues them for transmission.

```
procedure output_interface(address, datagram)
    x25addr = convert_internet_to_x25(address)
    enqueue_datagram(x25addr, datagram)
end
```

Enqueuing a datagram may also trigger the circuit manager:

```
procedure enqueue_datagram(x25addr, datagram)
    if (no circuits are open to x25addr)
        or (the queue contains more than one datagram and
            less than $C_{max}$ are open to x25addr and
            the last open was more than $T_{open}$ seconds ago) then
        if (all channels are in use) then
            begin
                select_and_close_circuit
                open_x25_circuit(x25addr)
            end
    place datagram on queue associated with x25addr
end
```

Circuit selection tries to close circuits to a site that has multiple connections before preempting one that does not:

```
procedure select_and_close_circuit
    if (there exists a site with more than $C_{max}$ open circuits
            such that its LRU circuit has been open
            for longer than $T_{min}$ seconds) then
        close it
    else if (the LRU circuit has been open for
            longer than $T_{min}$ seconds) then
        close it
    else
        fail
end
```

Procedure *select_and_close_circuit* may fail to find a circuit that has been idle long enough. If this happens, the procedure *open_x25_circuit* will also fail. Such failures are interpreted as an overload of the X.25 device, and the interface discards datagrams that are waiting to be sent to the site for which no connection can be obtained.

7. Summary and Conclusions

We have conducted experiments to analyze and improve the performance of an IP-to-X.25 interface that meets the proposed DARPA standard. Measurements revealed that throughput was limited by delays in the underlying X.25 network, not in the interface software or higher-level protocols. To understand the problem better, a model of network behavior was developed. The model captures notions of window size, cross-network delay, packet transmission delay, and frame queuing delay. To simplify analysis, each of these is assumed to be constant (even though measurements show that some parameters vary).

The model explains burst-mode behavior and predicts a bound on the maximum throughput rate. Using parameters measured from GTE Telenet for a single open virtual circuit, the model predicts that the interface cannot use more than 28% of the line capacity (2631 bps). Measurements show that it sometimes achieves 23% of the line capacity (2200 bps). Variation in network delays and other overhead easily explains the difference. We are pleased that the bound from such a simple model is so accurate.

The model predicts a linear increase in throughput with increasing numbers of virtual circuits. It also predicts a point of saturation beyond which increasing the number of circuits will not increase throughput. Measurements of GTE Telenet verify both the predicted linear increase in throughput and the saturation point are accurate. In practice, saturation values of 3–4 seem to work well for our GTE Telenet connection.

The use of multiple virtual circuits to increase performance is allowed in the proposed DARPA IP-to-X.25 interface standard. We have added this optimization to such an interface with good results.

Although multiple virtual circuits can dramatically increase throughput, the optimization is not always warranted. If the underlying hardware link is saturated, opening additional circuits will increase costs without increasing performance.

When all circuits connect to the same destination, saturation is easy to predict. When communicating with multiple sites, however, saturation may occur before any site opens multiple virtual circuits. Consider a heavily-used gateway that keeps the X.25 line saturated. Opening additional virtual circuits to a given site would result in lower throughput to other sites.

Another problem is introduced when two communicating sites use asymmetric values for C_{max}. One site may keep opening additional virtual circuits while the other keeps closing them. Our implementation, for example, can limit multiple circuits on a per-site basis, making it possible to avoid such asymmetry.

So far, our work has concentrated on improving X.25 performance. We have observed that better performance could be obtained by reducing the number of retransmissions by TCP. We conjecture that long delays and burst delivery of acknowledgements may produce significantly more retransmission than necessary. Further study is warranted.

References

1. Barnoski, M. K. (Ed.). *Fundamentals of Optical Fiber Communications*, Academic Press, New York, 1976.
2. CCITT. *Recommendations X.3, X.25, X.28, and X.29 on Packet Switched Data Services*, International Telecommunication Union, Geneva, 1978.
3. Cerf, V. G. and Kahn, R. E. A protocol for packet network interconnection. *IEEE Transactions on Communication COM-22* (May 1974), 637–648.
4. Comer, D. E. and Korb, J. T. CSNET protocol software: the IP-to-X.25 interface. *SIGCOMM Symposium on Communications Architectures and Protocols* (Mar. 1983), 154–159.
5. Das, S. and Cole, R. Protocol converter, transport level specifications. INDRA Note 806, Dept. of Statistics and Computer Science, University College, London, 1979.
6. Denning, P. J. The working set model for program behavior. *Comm. ACM 11*, 5 (May 1968), 323–333.
7. Jacobs, I. M., Binder, R., and Hoversten, E. V. General purpose satellite networks. *Proceedings of the IEEE 66* (Nov. 1978), 1448–1467.
8. Jacobs, I. Lightwave transmission in the Bell System. *Communication Networks Conference*, Washington, DC, Jan. 1979.
9. Kahn, R. E. The organization of computer resources into a packet radio network. *IEEE Transactions on Communication COM-25* (Jan. 1977), 169–178.
10. Korb, J. T. A standard for the transmission of IP datagrams over public data networks. RFC 877, Purdue University, Sep. 1983.
11. Postel, J. DARPA Internet Program protocol specifications. RFCs 790, 791, 792, 793, 794, 795, 796, USC Information Sciences Institute, Marina del Ray, 1981.