18

Implementation and Performance Enhancement of a PC Based LAN/WAN Router with a Differential QOS Feature

S.V.R. Anand and A. Kumar
Department of Electrical Communication Engineering
Indian Institute of Science
Bangalore-560012, INDIA
e-mail : anand, anurag@ece.iisc.ernet.in

Abstract

In this paper we describe our approach to, and experiences in, developing a PC based IEEE 802.3 LAN - X.25 WAN IP router. The basic router functionality is achieved by integrating a commercially available synchronous serial port card for the PC-AT bus, and our enhancements of public domain or licensed pre-production source code for the communication protocols. The router implements a strategy developed in [Kumar *et al.*, 92] for providing differential QOS to bulk transfer and interactive traffic on WAN links. Various link level policies are studied to further improve the QOS. Several other performance issues that we encountered while developing the router are discussed. We study various task handling strategies for the router software to arrive at the ideal combination that gives the peak performance. The issues, and their resolutions for physical interface handling are detailed. Experimentation done to compare the various alternatives for resolving these issues are reported.

Keywords

PC based router, CLNS over CONS internetworking, X.25, RFC 877

1 INTRODUCTION

In a typical LAN-WAN interconnection with slow WAN links (less than 64kbps), to support network traffic with different performance requirements, the router should have the capability of providing differential QOS on the slow WAN links. The absence of this capability can be felt when we try doing telnet from a LAN node while ftp is in progress from another node on the same LAN, both using

S. V. Raghavan et al. (eds.), *Computer Networks, Architecture and Applications*

the same WAN link to reach a host on a remote LAN. The response times seen by the telnet user can be quite annoying. A detailed study of this issue can be found in [Kumar *et al.*, 92], wherein various internetworking strategies have been considered. From this paper we learn that a CONS based WAN can be effectively used to support services with varied network demands. The study was made in the context of LANs running a CLNS network protocol. Preliminary experimental results were given that validated the theoretical results.

In this paper we present our approach to, and experiences in developing a PC based IEEE 802.3 LAN-X.25 WAN router, that implements the strategies discussed in [Kumar *et al.*, 92].

At the time of this report, we have implemented and successfully tested our router software in the lab environment. The system is stable and has been deployed in four places; further installations are planned.

With the commercially available routers (including Cisco's router) it has been observed that the response of an interactive application is very poor in the presence of file transfers, both using the slow WAN(X.25) link. In those implementations we found that all the IP traffic is treated alike, and the packets are queued in a single FCFS queue after routing packets onto the WAN link. This in turn giving rise to the problem mentioned. In this aspect our implementation is different from the existing routers. We make use of the X.25 connection management effectively to support differential QOS required by the IP traffic.

In section 2, we give some of the implementation details. These include the platform on which the router was built, the software development, and the main features of the software package. In section 3, we discuss the performance issues that were considered regarding support for differential QOS and our approach to resolve these. We also report the experiments done to further improve QOS. Section 4 pertains to performance issues such as task scheduling and physical interface handling. We present the approach we have taken to resolve these issues by looking at the performance bottlenecks in the router software. In the light of this, a description of various strategies considered is presented. Then we describe in detail the experiments conducted, and evaluate the strategies and discuss the results. We conclude the paper by summarizing the achievements of the work done and outline the future work underway.

2 IMPLEMENTATION

Our router software has been built on an i386 based PC-AT running DOS. Presently the router supports IP and X.25 protocols. The two physical interfaces we handle are an Ethernet interface,and a synchronous serial interface. The hardware for the Ethernet interface is a Western Digital (WD8003) compatible add-on card. For the serial interface we have used a commercially available i82530 SCC based card that can support link speeds upto 19200 bps. This card does not have an on-board memory and hence relies on the host's services.

The software is derived by integrating two modules: Retix's X.25 ([Retix, 88]) and CMU's PC/IP. The initial work involved in integrating these two software packages was to change the model of the PC/IP code from small to large. This was due to the large memory requirements of the router. The design and

the implementation of PC/IP follows the now popularly known thread mechanism([Tanenbaum, 92]). PC/IP realises a multi-tasking environment using this mechanism. The tasks are non-preemptive. The tasks are scheduled in a round robin fashion. There are no priorities associated with the tasks. We have customised the scheduler to some extent so that a priority based scheduling is possible.

The main advantage with the thread approach is that one can create various tasks based on some functionality, and these will be executing independently using their own stacks. This independent stack makes task switching possible at any point of execution. Resumption of the task execution continues from where it was left. For the non-preemptive multi-tasking environment that we have, based on an event, a task can be programmed to yield to another task, may be high priority one, at any point of the execution. Since modularity is inherent, the addition and deletion of tasks can be done without intervening in other tasks. Thus it is easy to add or remove functionality without affecting the overall system. It is possible to build a customised scheduler to schedule tasks according to some performance criteria. Since it is possible to make a task sleep, a better CPU utilisation is achieved. From the developer's view point debugging becomes very simple and stack overflow problem can be localised and corrected just by looking at one task.

Considering the advantages with this approach, especially for a router environment, we have changed the main Retix functional modules to threads. A function can be changed to a thread by giving it a stack frame, so that the function executes as an independent entity. Realising that the CMU package was designed for a host, and not for a router, additional functionality had to be introduced for routing purposes. This required an attachment of an additional logical link interface at IP, apart from the existing Ethernet interface.

We have included the TCP task along with other tasks to support telnet service at the router. The independent buffer and timer managements of both the pieces of software have been retained for the simplicity of integration.

For the PC/IP - X.25 interface (SNDCF in OSI terminology) we have implemented the *RFC 0877* specification [RFC877, 83]. The mapping of destination IP addresses to X.25 DTE addresses has been done at this interface. We have given a provision for establishing an X.25 connection directly to a remote router more than a hop away. This is useful because it avoids IP packets requiring processing at the IP layer of the intermediate routers, thus improving switching efficiency.

Interoperability tests with the Cisco(MGS) router have been done successfully. In fact Cisco router has throughout been a part of our experimental test-bed, on which all our performance related experiments were conducted. So far as X.25 connection establishment is concerned, we have used flow control negotiation facilities for packet size and window size. These parameters are useful not only to support differential QOS, but also to determine the router performance.

Finally, many performance related issues have been experimentally studied to enhance the performance. From these, we have incorporated the strategies that gave the best results. We have also attempted to explain the experimental

observations.

2.1 Main Features

- Our implementation supports differential QOS required for bulk transfer vs. interactive traffic. The window flow control that is required for this purpose is configurable. These window sizes are directly related to the QOS given to each service class. The details of this issue will be discussed later.

- On the router console we provide the utilities ping and telnet. These are useful in managing the network. For one thing, we can use ping to check for reachability. We can alter the link level parameters by telnetting to other router(s), among the other things.

- The software comes with a customisation package to configure various parameters pertaining to each layer. The channel management menu enables one to set parameters that govern QOS. Some typical configurable parameters are: window sizes, timeouts, and number of retransmissions for X.25 and LAPB layers. Of course the serial card can be configured to have an external or an internal clock. The serial line bit rates can be chosen upto a maximum of 19200 bps.

- The routing customisation can be done in a simple way by specifying the mapping information. We can also make use of X.25 switching thus avoiding IP processing if the intermediate routers to a destination support X.25 switching.

- Debugging facilities can be enabled to observe the data movement at different layers of the router stack. The information provided is related to packet and frame information and the state information at LAPB and X.25 layers.

- We have developed a program that can be used to remotely monitor the health of the router. e.g., one can find the buffer occupancy on both LAN and WAN interfaces. We have extended this feature to control router parameters from a remote host.

- We also support SNMP agent on our router based on CMU's implementation.

3 DIFFERENTIAL QOS FOR IP TRAFFIC

In a LAN-WAN environment like ours where the WAN link speeds are less than 64 kbps (typically 9600 bps), and the serial line quality is poor requiring an ARQ protocol, the delays experienced by a LAN user who is remotely logged-in to a host on a remote LAN can be quite annoying in the presence of file transfers between other nodes on the same two LANs. The reasons for this are the following:

(i) Packet sizes for the file transfers are large (552 bytes typical) when compared to interactive traffic where typical packet sizes are 64 bytes (we assume that the user is typing characters on the terminal). Each file transfer packet can give rise to several link packets depending on the Maximum Transmission Unit (MTU) of the link.

(ii) The packet rate into the router for file transfer is much higher, because TCP window sizes can be quite large and disk access times can be ignored due to the parallelism we get with the low speed WAN.

Because of these reasons we find that the link window is hogged by the file transfer packets, thus causing intolerable response times for the interactive user. Hence we have an unfair sharing of the link, in a QOS sense, i.e., two services are sharing a facility, each with different QOS requirements, and one gets relatively better QOS than the other.

Note that simple priority before the link level window will not be enough to help the interactive traffic if the link window is large. Since interactive traffic arrives slowly, with a high probability, interactive packets find the link window full of file transfer packets. Thus even if this arriving packet is given priority before entry into the link window, it will still be delayed by a window full of file transfer packets. For a link packet size of 512 bytes, and link window of 7, this can be as much as 1.5 secs. at a link speed of 2400 bps. Inserting the arriving interactive packet at the head of the transmitter's queue is infeasible as then all sequence numbers of packets admitted into link window will have to be altered.

If the serial link card has buffering and runs the ARQ protocol, then provision of priority will require modification of the card. For analysis and simulation of this issue see [Kumar *et al.*, 92].

An alternative is to limit the number of file transfer packets that can be outstanding on the link. This can be achieved by segregating the file transfer traffic into a window controlled connection thus limiting the number of file transfer packets that can occupy the link window. This is easily done with a standard protocol, i.e., X.25. Note that simple priority (to telnet) before link insertion is not adequate since the telnet packets will have to wait for a link window full of ftp link packets anyway; and the link window can be large to support a range of link packet sizes. If the link window is small, say 2, then priority before insertion into link window should be enough.

There is another issue, however, that our approach addresses; the provision of fair bandwidth sharing between, for example two ftps one originating from a LAN and the other from a slow serial link attached to the router, both being multiplexed onto the slow outgoing link (see [Kumar *et al.*, 92]). Further, the X.25 interface permits us to connect the router to a Cisco and to the public X.25 network.

There are differences between our work and the existing work on QoS control in high speed networks. The latter work assumes that quantifiable, hard QoS guarantees are required for the various traffic classes; hence very fine grained control strategies are proposed and studied in this context ([Demers *et al.*, 90]). Implementation of these strategies requires complex nonstandard procedures, and may even require hardware support. The services that we are concerned with (namely, ftp and telnet) do not have any hard QoS requirements. The

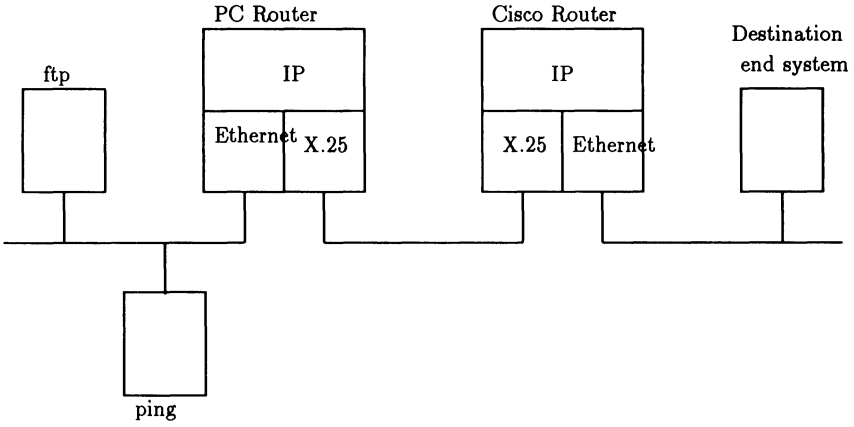

Figure 1: Setup for the QOS experiment

objective is simply to provide reasonably low response times for telnet, high throughput for ftp, fairness among services of the same class, and all this while efficiently utilising link bandwidth. We have realised this objective by utilising the existing protocol support in routers.

Since predominant number of hosts that provide Internet services do not make use of Type Of Service (TOS) field in the IP header to indicate the QOS class of the IP packet, we cannot make use of this field to identify the service class and provide the required QOS. Hence to circumvent this problem and identify the service class of the Internet service, we have used TCP port numbers which are globally unique for most of the popular services. At the router, apart from parsing IP header for routing purposes we also parse the TCP header. Once we identify the end-application, and thereby service class, we open an X.25 connection with an appropriate window size negotiation facility. Thus, for example, we open one connection for ftp packets, and another connection for telnet/rlogin packets, and so on. We find that the response time of an interactive application decreases dramatically with little decrease in ftp throughput. This corroborates the analysis in [Kumar *et al.*, 92]. To appreciate this better, quantitative measures have been shown here.

In the current implementation we classify three classes of traffic, namely, ftp, telnet/rlogin and others, and thus we have three X.25 connections with corresponding flow control parameters.

We now present the results obtained from some experiments. We have used an i386 machine running at 33 Mhz clock to run our router software. The testbed is shown in Figure 1. Since ping delays can be measured, we use ping to emulate a telnet like session. ping packets are 64 bytes long including the IP header. The serial link speed is set to 19200 bps. The ftp packet size is 552 bytes. At the serial interface with no fragmentation done at X.25, the lengths of ping and ftp packets become 69 and 557 bytes respectively with the inclusion of X.25 and LAPB headers.

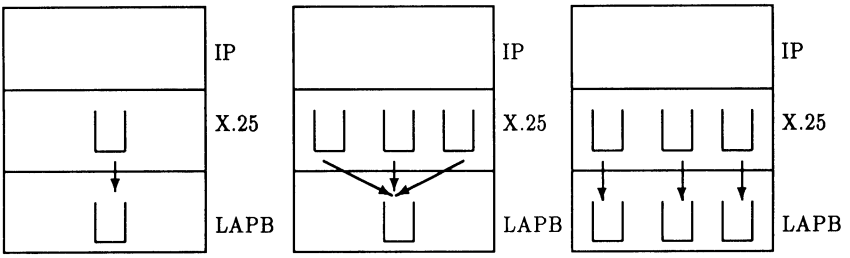

Figure 2: Traffic streams at Link level and X.25

Single X.25 connection:

To start with, we have taken throughput and delay measurements independently for ftp and ping with X.25 window size 2, and LAPB window size 7. The following are the results.

ftp throughput (in bytes/sec) : 1970
ping average response time (in millisecs.) : 270

Next, we have done a similar experiment with both ftp and ping initiated simultaneously.

ftp throughput (in bytes/sec) : 1910
ping average response time (in millisecs.) : 1300

We can clearly see the high ping response times once a bursty file transfer is in progress.

Multiple X.25 connections:

We now see ping delays by opening multiple X.25 connections with X.25 windows for ftp, and ping being 2 and 7 respectively. LAPB window size is 7 as in the earlier case. The following results were obtained.

ftp throughput (in byte/sec) : 1828
ping average response time (in millisecs.) : 518

It can be seen that even in the presence of ftp traffic, the response times for ping are quite low thus encouraging one to go for X.25 based WAN and have service based network connections. Note that we are not hurting the ftp traffic as can be seen from the less than 5% fall in ftp throughputs.

3.1 Link Level Scheduling

To attempt to enhance the QOS further, we assigned priorities to the multiple data streams obtained from the various X.25 connections. We thus provided priority based queues. When the link protocol gets credits, it serves these queues on a priority basis with the highest priority queue served first. The number of elements served will be equal to the number of credits. We have given the highest priority to the interactive traffic, and the lowest to the ftp traffic. The rest of the traffic such as UDP is assigned a medium priority.

We have implemented FCFS, Round-Robin and Exhaustive policies. In the FCFS policy, we serve the data streams in the order of their arrival to the priority queues. In other words this amounts to serving a single queue and treating the traffic alike. For Round-Robin policy, we serve the queues by taking one element from each queue. If there are no elements in a particular queue, and there are credits available, the server then serves the remaining queues in the order of priority within the remaining queues. For the Exhaustive policy, the server serves all the elements in the queue it is currently serving before serving the next queue. Segregation of traffic streams is shown in Figure 2. The first part of the figure shows the case of a single X.25 connection with a single queue at the link level. The data stream cannot be segregated at the link level since the order of packet transmission has already been decided at X.25 which assigns the sequence numbers for the packets. The second part of the same figure shows multiple data streams being put into a single link level queue. However, in this part traffic segregation has been done at the X.25 level. Both these parts depict the FCFS policy at the link level. The third part shows the traffic on multiple X.25 connections, and also the segregation done at the link level based on the priority given to the data streams.

In this context it should be mentioned that the policy that was employed by Retix's X.25 implementation was FCFS. During this experiment we also observe the delay and throughputs at the LAN end systems for various X.25 window sizes for the connection corresponding to ftp. Table 1 shows the experiments and the results obtained. From Table 1 we note that once the traffic is segregated into different channels, we get all the obtainable QOS improvement. The link level scheduling policy did not give any further appreciable improvement.

Table 1 Effect of Link Level Scheduling on QOS

Link Scheduling Policy	X.25 Window for ftp	ftp Throughput (bytes/sec)	Ping Delay (milli secs)
FCFS	2	1828	516
	4	1828	515
	7	1828	522
Round Robin	2	1828	524
	4	1828	518
	7	1828	518
Exhaustive	2	1828	514
	4	1828	523
	7	1818	521

4 TASK SCHEDULING IN THE ROUTER SOFTWARE

The software of any router comprises parts that handle the transmission and reception of packets at the physical interface, and parts that handle the movement of packets between these physical interfaces. Let us go through the software structure and see how various tasks have been handled. In doing so we identify the performance bottlenecks in the system. We experiment with various strategies for task scheduling to improve performance at these bottlenecks.

4.1 Software Structure

The software design used in building the router is inherited from CMU's PC/IP and is based on the well known concept of threads. This concept has helped us to create a cooperative multi-tasking environment under DOS. The implementation involved integration of CMU's PC/IP, and Retix's X.25 packages. Of these two main modules, PC/IP inherently follows the thread approach. We have introduced a task structure into Retix code to comply with the overall design philosophy. The main tasks created in the overall system are: Ethernet packet reception, Ethernet packet processing, serial interface reception, serial interface processing, and timer recovery. Ethernet packet processing include routing, and forwarding for packets coming from LAN. These functions are achieved through function calls from within the same task. The frame and packet level processing for the serial interface are done in the similar way from the corresponding task.

We address some of the performance issues considered for the transmission and reception mechanisms on the interfaces. We give their resolutions that enhance the router performance.

As is generally the case, there are different mechanisms for transmission and reception: blocked mode polling, unblocked mode polling, and using interrupt mode. In order to describe these, let us consider a transmission or a reception on a physical interface to be an event. In blocked mode polling, after initiating an event, CPU polls a status register till the status of the completed event is obtained. Thus the task from which the event was initiated blocks all the other tasks. In unblocked mode polling, the task which had generated the event polls a status register at regular time intervals to obtain the status of the completed event. In between the polling intervals the scheduler can schedule other tasks. For the interrupt mode, the status of the completed event is posted by an interrupt from the physical interface responsible for the event. Interrupt mode is unblocked and polling is completely avoided.

4.2 Ethernet Interface

Transmission:

In our implementation, the transmission discipline on the Ethernet interface is blocked polling. After initiating a transmission on the Ethernet interface, the CPU will be polling for an event that indicates the status of the completed

transmission. Considering the non-preemptive multi-tasking structure, this essentially results in a blocked task. However, the blocking delay can be ignored since we are dealing with the Ethernet speeds and a moderate load on our Ethernet LAN. Further, this delay is negligible compared to the slower serial interface. Hence, blocked mode transmission at the Ethernet interface cannot be the bottleneck for the system performance.

Reception:

The reception process is inherently asynchronous, and hence it is natural to prefer interrupt mode. The Ethernet adapter is programmed to post the event and interrupt the CPU. The pending reception event can also be known by polling a register of the Ethernet adapter. This method is adopted to take care of the possible missed interrupts at the CPU. The polling interval is suitably chosen so as not to overly tax the host.

On a heavily loaded network, incoming Ethernet packets arrive back to back. This results in a condition wherein one interrupt can correspond to multiple messages buffered at the receiver. On an interrupt, we do a batch transfer of messages from the Ethernet adapter to the host buffers. Thus we prevent receiver overflow and the consequential dropping of packets.

4.3 Serial Interface

It is clear that the performance bottleneck in LAN-WAN interconnection over low speed links is the serial link itself [Kumar *et al.*, 92]. It is therefore imperative in the design of the router software that the link is utilised as best as possible. Thus, for example, no time should be wasted between the completion of one frame transmission on the serial link and the initiation of the next one. On the other hand since the link is slow, and it takes long to transmit a packet, and the software should not block while a frame is being transmitted but should proceed with other activities. These rather obvious requirements need special attention, however, in the DOS environment in which multiprocessing does not come naturally. In fact, since we integrated two existing pieces of software to build our router we were initially confronted with decisions made by the original software modules that were not necessarily optimal for our application.

In order to ascertain the best combination of strategies we experimented with several. Before moving on to the experiments, and the results obtained we mention the strategies considered. In our implementation, the end of transmission or reception will be notified through an interrupt from the serial device. We use this event to wake up the respective tasks. Based on our discussion on the Ethernet interface, we can think of similar service modes for serial line handling.

Reception on the serial line is through DMA, and after the reception of a frame, the CPU will be notified by an interrupt. Interrupt routine then wakes up the task designated to do further frame processing. Hence this reception process is asynchronous. There are no significant delays involved while receiving incoming frames. We now look at the possible variations for handling serial transmission.

1. Blocked mode transmission:

A call to transmit routine gets blocked within the routine till all the frames pending get transmitted. This method hurts the performance, especially on slow links as other activities get locked out while transmission on the serial link is proceeding. We also see from our experimental results the conditions in which this mode does not hurt the performance seriously.

2. Non-Blocked mode transmission:

As the term implies, the transmitter initiates transmission but does note wait for completion of transmission. We have introduced a task which is woken up once the frame gets transmitted. This high priority task will then initiate further transmissions. This is to minimise the delay between the link access times. Let us call this the transmit_task. Though a better method than the earlier one in terms of CPU utilisation, there is still some finite delay between frame transmissions. The scheduler can schedule the transmit_task only when the currently running task yields to the scheduler. Thus there will be times when the serial interface will be idling. This in turn gives rise to the under utilisation of the serial interface.

3. Asynchronous initiation of transmission from within the Interrupt Service Routine:

Here, instead of just posting an event for the scheduler, as in the earlier case, the end of transmission interrupt routine will trigger further transmission of frames from the LAPB transmit queue. Hence we no longer need the additional transmit_task just for frame transmissions. Certainly this should be the best mechanism compared to the other two options. From the software engineering point of view, we take care of the possible race condition at the shared transmit queue. LAPB accesses this queue, to insert outbound frames, as does the asynchronous occurrence of an interrupt, which causes an element to be removed from the same queue. We use the semaphore mechanism to prevent simultaneous access.

In section 4.4 we present the experiments that were carried out for comparing the above mentioned strategies.

4.4 Test-bed for Studying Serial Line Handling Issues

For these set of experiments we have used i286 based PC-AT to run our router software as our main i386 platform was out of order during this experimentation. We have used ftp throughputs as the performance measure. We measured throughputs by doing file transfers between two end systems on LANS connected by routers as shown in Figure 1 (ignoring the machine labelled "ping"). We have conducted two sets of experiments. The first two options mentioned for the serial line will be studied in our first test. The results have been tabulated in Table 2 and Table 3.

4.5 Discussion

Description of various columns of Tables 2 and 3 is the following. The first column shows the service strategy for the incoming frame processing on the serial line. It indicates whether the task that does frame processing services all the frames (exhaustive) or processes one frame before yielding to other tasks. The second column is the mode of frame transmission on the serial line. The Ethernet packet receive strategy is shown in the third column. The interpretation of the notation given for column 1, is also applicable here with respect to the Ethernet packet processing. The last column gives the ftp throughputs for various combination of strategies. Rows 1 to 4 of the Tables 2 and 3 give the measurements obtained for the blocked mode transmission on the serial link. Rows 5 to 8 of these tables give the measurements for non-blocked frame transmission on the serial link. The Ethernet packet transmission delays are ignored because of the moderate load on our Ethernet LAN.

Table 2 ftp throughput for various reception and transmission strategies at the interfaces.
Configuration : LAPB window 7, X.25 window 7, Baudrate 19200 bps

Sr.No.	Frame Reception on Serial	Frame Transmission on Serial	Ethernet Packet Reception	ftp thruput in bytes/sec
1.	Single	Blocked	Single	1360
2.	Single	Blocked	Exhaustive	1458
3.	Exhaustive	Blocked	Single	1758
4.	Exhaustive	Blocked	Exhaustive	1458
5.	Single	Non-Blocked	Single	1734
6.	Single	Non-Blocked	Exhaustive	1734
7.	Exhaustive	Non-Blocked	Single	1734
8.	Exhaustive	Non-Blocked	Exhaustive	1734

Table 3 ftp throughput for various reception and transmission strategies at the interfaces
Configuration : LAPB window 2, X.25 window 7, Baudrate 19200 bps

Sr.No.	Frame Reception on Serial	Frame Transmission on Serial	Ethernet Packet Reception	ftp thruput in bytes/sec
1.	Single	Blocked	Single	1834
2.	Single	Blocked	Exhaustive	1834
3.	Exhaustive	Blocked	Single	1808
4.	Exhaustive	Blocked	Exhaustive	1808
5.	Single	Non-Blocked	Single	1734
6.	Single	Non-Blocked	Exhaustive	1734
7.	Exhaustive	Non-Blocked	Single	1734
8.	Exhaustive	Non-Blocked	Exhaustive	1734

From the tables we see that for a large LAPB window size (i.e. 7 in Table 2), non-blocked frame transmission on the serial line gives a better performance irrespective of the kind of reception strategies followed for the serial and Ethernet interfaces. For small LAPB window size (i.e. 2 in Table 3), non-blocking transmission mode hurts the performance compared to the blocking mode, since with non blocked transmission the transmit_task keeps yielding to other tasks more frequently after the initiation of the transmission. Because of the non-preemptive nature of the task scheduler even the high priority transmit_task has to wait till the currently executing task gives up control to the scheduler. This causes link access delays and therefore we get low ftp throughput.

In our next experiment we try the option wherein the serial link transmission is asynchronously triggered from the interrupt service routine, and there is no blocking while transmission is in progress. This option eliminates the CPU idling or the starvation of the serial link, and the link access delays are minimised. Hence we expect encouraging results from this experiment. Considering the results obtained from the previous experiment we have fixed the servicing strategies for the incoming message processing on the two physical interfaces to be exhaustive. Apart from studying the influence of serial line handling we also see how X.25 window affects the performance. The results are shown in Table 4.

We compare the best ftp throughput obtained in Tables 2 and 3 with that obtained in Table 4. We see a remarkable improvement in the performance. LAPB window size does not matter since we always get back the acknowledgement by the time the LAPB frame gets transmitted. This causes one more

frame to enter the transmit queue, which will be immediately transmitted. A larger X.25 size improves the performance further because of the parallelism that is obtained at the X.25 protocol processing while the serial line is busy sending frames.

Table 4 Configuration : Baudrate 19200 bps
Servicing incoming messages on the interfaces: exhaustive

LAPB Window	X.25 Window	ftp thruput bytes/sec
2	2	1970
2	7	2019
7	2	1970
7	7	2019

5 FINAL RESULT

From the previous section we have observed that the version of the router software incorporating asynchronous initiation of frame transmission on serial link with frame transmissions triggered from within interrupt routine has given the best performance. Recall that the performance tuning was done on a i286 PC/AT. We have used this tuned version of the software on i386 m/c and measured ftp throughput again. We now present the measurement thus obtained for i386 m/c as the final result. The X.25 window is set to 7, and LAPB window is set to 2 in accordance with our previous discussion. The serial link speed is configured to 19200 bps (i.e. 2400 bytes/sec).

ftp throughput (in bytes/sec) : 2170

Compare this result with that obtained from our earlier version prior to the performance enhancement (see section 3). We clearly see an improvement of about 10% (from 1970 bytes/sec to 2170 bytes/sec) in the ftp throughput from our latest version.

6 CONCLUSIONS & FUTURE WORK

We have proposed a feasible method for building LAN/WAN router that provide differential QOS using the well-known port numbers for Internet applications, and the channel management introduced at SNDCF. Our implementation showed an appreciable improvement in the response times for interactive traffic in the presence of bursty ftp traffic. From the measurements we can also conclude that, once the channeling of the IP traffic is done, the nature of the link level scheduling policy for the serial interface does not change the performance. Of the serial line handling mechanisms considered, and tested, non-blocking with asynchronous activation of the transmitter has given the best

performance. We have seen further enhancement in the performance by choosing a larger X.25 window. Also, the servicing strategy at the Ethernet interface for incoming Ethernet packets, played no role in improving the performance.

We are planning to introduce more serial interfaces to the router to interconnect more subnets. Presently we use static routing information. The expansion of the router demands that we use dynamic routing protocols. There is a thought in this direction to incorporate at least one of the existing dynamic routing protocols.

References

[Kumar *et al.*, 92] Anurag Kumar, T.V.J.Ganesh Babu and S.V.R.Anand, "Comparative Performance of Queueing Strategies for LAN-WAN Router in Packet Data Networks", *Proceedings of the International Conference on Computer Networks, Architectures and Applications, Networks 92* .

[Demers *et al.*, 90] A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of Fair Queuing Algorithms", *Journal of Internetworking Research and Experience*, September 1990, pp. 3-26.

[Retix, 88] Retix Manual on "X.25 Packet Level Protocol", Retix Inc., Publication Number 1080057-00-B, March 7, 1988.

[Tanenbaum, 92] Andrew S. Tanenbaum "Modern Operating Systems", Prentice Hall 1992.

[RFC877, 83] J.T. Korb, "A Standard for the Transmission of IP Datagrams Over Public Data Networks", Internet RFC 0877, September 1983.

BIOGRAPHIES

S.V.R. Anand has a BSC(Hons) from Osmania University, and a B.E. in Electrical Communication Engg. from the Indian Insititute of Science (IISc). He is currently working in Education and Research Network project (ERNET) at IISc.

Anurag Kumar has a B.Tech. in E.E. from the Indian Institute of Technology at Kanpur, and a PhD from Cornell University. He was a Member of Technical Staff at AT&T Bell Labs, Holmdel for over 6 years. Since 1988 he has been with the Indian Institute of Science (IISc), Bangalore, in the Dept. of Electrical Communication Engineering, where he is now Associate Professor. He is also the Coordinator at IISc of the Education and Research Network Project, which has set up a country-wide computer network for academic and research institutions, and conducts R&D in the area of communication networks. His own research interests are in the area of modelling, analysis, control and optimisation problems arising in communication networks and distributed systems.