

An application-oriented analysis of TCP/IP in high speed LANs†

D. Sudheer K. Maly D. E. Keyes and C. M. Overstreet
Computer Science Department
Old Dominion University
Norfolk VA 23529-0162

†This was partially supported by Synoptics Inc. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing Synoptics.

Abstract

In this paper we investigate the limitations of the normal implementation of TCP(UDP)/IP and describe an application-oriented analysis in high-speed Local Area Networks, such as ATMs.

We conducted tests to measure aberration in Quality of Service of an application in terms of connection establishment time, throughput, and loss with respect to block size. We report the effect of TCP window size and the Silly Window Syndrome (SWS). Suggestions are made to avoid the SWS and effectively control TCP window size to increase throughput. Data losses of nearly 27% are observed with a UDP application. Knowledge of the status of the network can be used effectively by a host to reduce losses. We demonstrate this point with the help of a simple rate control algorithm at the user level in the UDP/IP environment.

Results obtained from the above experiments are used to analyze a simulated Distributed Computing application. We study the effect of using a standard protocol suite (TCP/IP) for running this application in a Solaris 2.3 operating system. We identify problems occurring when an application sends large amount of data in a bursty fashion and suggest possible solutions.

Key words: Quality of Service, Traffic pattern, Communication topology, Network topology.

1 Introduction

As the necessity for High Speed Network (HSN) increase, deficiencies in shared medium network protocols become more apparent. Ethernet protocol col-

lapse at higher loads [1]. The asynchronous traffic on 100-Mbps Fiber-Distributed Data Interface (FDDI) is dependent on network configuration and load [2]. In DQDB (Distributed Queue Dual Bus), fairness problems at higher loads with respect to the location of the node on the network is a well understood phenomenon [3].

Many of the above mentioned problems disappear when using switch-based network protocols which have bandwidth dedicated to individual hosts such as ATMs [4, 6]. By having fixed size cells and end-to-end flow control and error control, it is possible for ATM to scale to larger user community. Because of statistical multiplexing [7], ATM is highly flexible for different types of traffic characteristics [5].

With the availability of flexible and scalable network protocol architectures, scrutiny now moves to the higher level (end-to-end host) protocols. It is essential to evaluate the dynamics of the higher-level protocols in an HSN environment. A description of the controlling parameters in these protocols and their effect over longer periods of time on the connection is the motivation behind this paper. A poor combination of the control parameters can lead to abysmal performance. Application developers unfamiliar with the internals of network protocols can obtain some rules of thumb from this work.

To demonstrate the effect of the results we obtained a Distributed Computing (DC) application is used. By choosing a distributed computing environment, in a switch-based network scenario, it is easy to emulate virtually any arbitrary communication topology. But in doing so, we are entering into a different set of problems, in terms of network and machine dependent behaviors, which is the topic of interest in this paper.

We assume in this paper a distributed homogeneous host architecture (hence, communication to computation ratio is constant) with homogeneous underlying network paradigm. Also it is assumed in this

paper that computation and communication can be overlapped.

In section 2, we discuss a typical DC application, and how our simulated program can create the required environment. Section 3 presents a brief overview of the end-to-end parameters that effect the performance of the application. We identify the idle network case bottlenecks encountered on the host and in the network and a simple solution to them in section 4. Section 5 discusses effect of these results on a DC application and conclusions are present in the last section.

2 About simulated DC applications

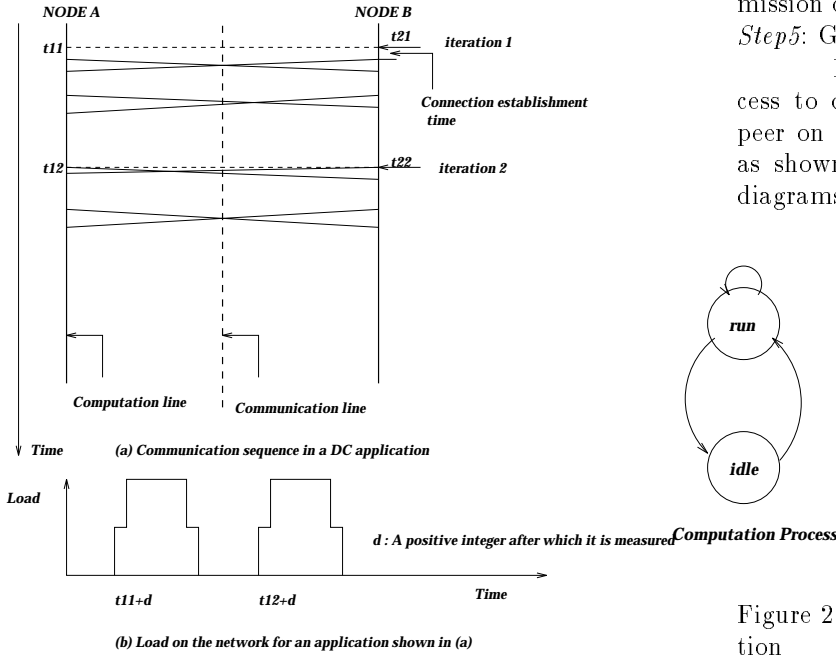


Figure 1: A typical DC application and its traffic pattern

Let us assume that two nodes *NodeA* and *NodeB* are sharing the execution of a DC application (Fig.1). Let *NodeA* and *NodeB* start their computation at times t_{11} and t_{21} respectively. A connection establishment process takes place before communication/computation begins. Since hosts are homogeneous and hence have same computation to communication ratio, communication overlaps between iteration from both the nodes as shown in Fig.1b. This might increase the computation time. If the communication channel is not a bottleneck and the computation involved in each iteration is long enough to complete the communication between iterations (i.e, data necessary at *NodeA* for iteration 2 is available before t_{12}),

then we don't observe any stalls between iterations. Rate control is inbuilt in our algorithm to avoid communication process take priority (as explained in the next section) over computation process.

The simulated application has a task running on two nodes participating in this application. Each of them will execute the following algorithm, with the exception that one of them acting as a server will establish the connection between the two.

Algorithm :

Step1: Establish Connection with each other

Step2: Start computation and communication processes

Step3: Findout the amount of time taken for computation

Step4: Use Step 3 information to find the rate of submission of data to transfer across the network

Step5: Goto step 2

In each task we have three processes. A process to do computation, one to transfer data to its peer on the other node, and another to receive data, as shown in the state diagram in Fig.2. The state diagrams are self-explanatory.

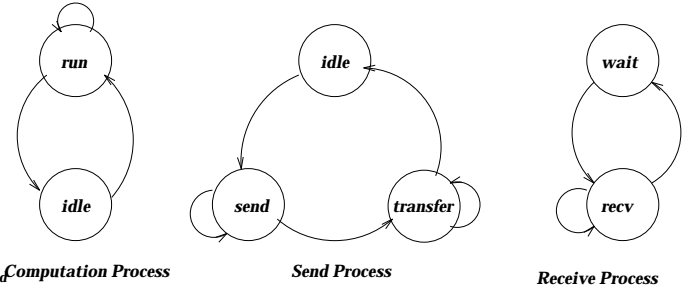


Figure 2: State diagram of the simulated DC application

Here we are making an assumption that the computational intensive process will take approximately the same order of time in the two consecutive iterations. Our algorithm implementation is flexible enough to transmit data at intermediate steps and the amount of data transmitted is configurable.

3 A brief overview of end-to-end performance issues

The load on the network, the load on the node and the end-system protocol suite behavior are the three major factors that play role in the performance of a client-server application. A Distributed Computing (DC) application, which is a combination of both CPU-bound as well as communication related tasks,

can exploit knowledge of the above three factors. We present a brief argument about the first two factors in this section and study the effect of the protocol behavior and the improvements one can obtain by knowing more about the control parameters in detail in the next two sections.

3.1 Load on the Network

Ethernet allows a DC algorithm designer to choose any communication topology for a given problem. An application broadcasting data among all the communicating nodes will generate lot of data on the network. To reduce protocol overhead and multiple transmission of messages, multicasting can be used. Although raw bandwidth available on Ethernet is 10Mbps, it is commonly observed that the throughput obtained in an actively used Ethernet LAN is between 4-5Mbps. Note that even lower-end architectures like SUN sparc 1 workstation can push data at 8Mbps and hence saturate the network at high communication periods. By using nodes from different subnetworks one need to pass through an additional bridge/router which reduce the throughput to minimum of both the Ethernets and an additional processing at the bridge at a given instant of time.

Unlike Ethernet LAN, on FDDI network bandwidth is generally not a limitation, but it is the protocol operation that could be a bottleneck. Assume a 4-node FDDI with alternative nodes communicating with each other. Note that due to the nature of a token ring only one node can transfer at any given time (until it is exhausted of data or token holding time (THT) expires, whichever is the minimum). Hence unwanted delays might be experienced at the other nodes waiting for the transmitting node to relinquish the token [2]. This decreases effective bandwidth. If the delays are very high a node might loose the advantage of overlapping communication and computation. Though we have a high data rate medium which can support global and ring like communication topologies, high delays will degrade the performance. Hence, we should look for a network medium which produce lesser delay by allowing each node to transmit at the same time and providing high data rate channel. This leads to switch-based networks like ATM LANs.

ATM is a 155/622Mbps switch-based point-to-point network paradigm. Because of statistical multiplexing [7] delay incurred as discussed in the FDDI case will be reduced. Also availability of high bandwidth will eliminate stalls at hosts because of data starvation for a DC application. ATM because of its scalability and flexibility, can support any arbitrary communication topology. But in ATM it is essential

to predict the traffic pattern, converting this pattern to network understandable QOS parameters, and conforming to the tolerance limits of the negotiated traffic characteristics. Deviating from the above characteristics might lead to loss of data in the network and hence increased retransmissions. Availability of fast connection establishment, a clear network understandable traffic pattern characterization and a good connection management policies inside ATM network will make ATM the best choice for a DC application.

3.2 Load on the nodes

Any computation intensive application can be characterized by the amount of CPU time it consumes, the memory that it occupies and the number page faults occurred during the execution of the process. Standard parallel computational libraries will let the programmer specify the resource limitations to avoid adverse performances of a poorly written code. For example in PETSc [9] provide a call called PICall, which can enforce resource limits on CPU time, elapsed time, memory size, and page faults.

As we are using a DC application where both computation and communication are important aspects, we mention some of the results that pertain to the study of the interaction of computation and communication processes on a node conducted in [8]. They found that the offered throughput on an idle Ethernet LAN drops from 6Mbps to almost 1Mbps by increasing the number of computation intensive processes from 1 to 8. Because of UNIX scheduling policy, the CPU intensive job will get lower priority to communication intensive job. It is also observed that the receive side of the communication process will slow down as the scheduling of applications is such that data is read from the socket buffers at a slower rate. Also there is an increase in protocol processing delay of about 10% on both the sending and receiving sides. One has to keep these issues in mind while handling a DC application. By an intelligent rate controlling data transmission, we can avoid the problem of giving high priority to communication intensive process.

3.3 Protocol suite behavior

We approach the problem of identifying the performance bottlenecks in the protocol suite in terms of standard user observable QOS parameters. Connection setup time is an important parameter in high speed networks, as fast connection establishment procedures will release the burden of network from holding resources when the connection is idle. Apart from this throughput, delay, delay jitter, and loss are important factors during data transfer phase. Throughput

measurements reflect the bandwidth usage of the application over the channel. Delay and delay jitter will give a measure of the protocol incorporated delays and jitter which can be used as operating system QOS for applications such as multimedia applications under no load conditions. Loss of data will give us the measure of loss probability for high data rate application on this testbed. For a DC application connection establishment time, throughput, and loss are the main QOS parameters. As long as delay is within an agreeable upper bound, it would not be of interest in this application.

We conduct experiments with a communication intensive application as a first step in no load condition to enhance the behavior of the end-system components. A clear understanding of these components help as a basis for the study of real world applications.

4 Bottlenecks in TCP/IP and their remedies

We conducted our experiments on an ATM LAN to identify the bottlenecks of TCP/IP in high speed networks. We are using a 155Mbps fiber optic point to point link between two SUN Sparc 10 workstations running Solaris 2.3 Operating System. These two Sparc 10s are connected by Synoptics LattisCell 10104 16x16 port switch which has an internal switching rate of 2Gbps.

TCP-IP and UDP-IP are linked by a set of incoming and outgoing queues. To prevent unwanted delays in these queues, some limitations on the queue lengths are enforced, which are called High Water Mark (HWM) [10] and Low Water Marks (LWM). A HWM is the maximum amount of data that can be queued at the interface. Once HWM is reached local pressure is applied to prevent data from being injected into the accepting module anymore till LWM is reached.

4.1 Connection Establishment time

Connection establishment time in the ATM environment is the time taken to setup the virtual connection between the two communicating machines. Once this connection is established, all the processes that are communicating with each other on these two machines will use this virtual connection. If the virtual connection is idle for certain amount of time (on Synoptics implementation it is 10 minutes) then the virtual connection is dissolved. In our case we assume that connection is present all the time. Hence the connection establishment time in fact will tell us the minimum amount of time required for the protocol processing on both the end-systems. In this plot [Fig.3] we are

interested in the upper and lower bound of the initial delay (generally referred to as “latency”).

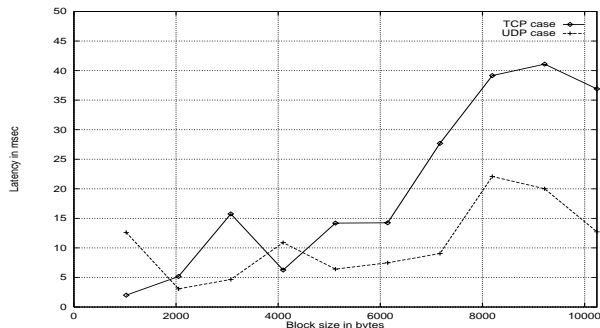


Figure 3: Connection establishment time versus block size

Fig.3 shows that there is a latency of 2-40 msec while using TCP and 2-27 msec while using UDP. In one of our experiments we found that by changing the window size with the fixed 5Kbytes block size the latency variation is between 2-15 msec. As we feel that this is within the affordable delay range for a DC application, we will not consider this as a major factor while testing the improvement in DC application in the next section.

4.2 Throughput

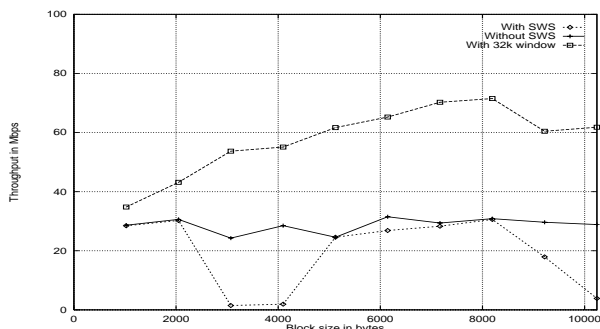


Figure 4: Improvement in TCP throughput
Fig.4 gives the average throughput observed by varying block size. We observed a maximum throughput of 30Mbps at block size of 8192bytes. Note the dip in throughput for the block sizes of 3072 and 4096. This is because of the phenomenon called “silly window syndrome (SWS)” [11]. It should be noted that by eliminating the SWS and increasing the TCP window size we have almost twofold improvement in throughput. As the block size is increased the processing time per block increases and hence the throughput decreases.

Throughput observed with UDP are high because of the lesser processing overhead of the protocol. This is also a good test to identify the buffer limitations and other overheads caused by the system.

Maximum throughput observed on the receiving side is 70Mbps, but on the sending side it is almost 120Mbps. This difference in throughput is because of the heavy loss of data in the transit when there are uneven surges in data transfer rates on the sending side. Fig.5 shows that by incorporating a rate control algorithm which submits data at regular intervals and by increasing the HWM and LWM at UDP-IP interface, there is no change in the obtained throughput. But as we see in the below this reduces data loss in UDP drastically.

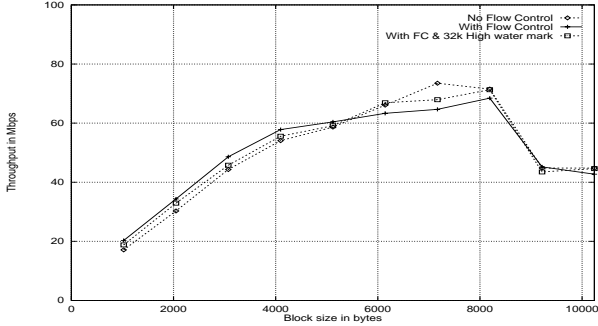


Figure 5: UDP throughput after the modifications

A steady increase in throughput is observed (Fig.5) till a block size of 8192bytes. This is because of no processing nature of UDP. UDP submits packets as it is to IP to segment them if necessary. On the receiving side these segments are reassembled by IP and are submitted to UDP. As the block size increases, the buffers in the system deplete very fast and hence UDP block until a fresh quota of buffers is available [10]. Hence, there will be higher delays at the sender-network interface.

4.3 Loss

Loss of data at high data rates on UDP is because of the resource quench on the participating systems and host-network interface rather than the network limitation. Fig.6 presents the loss percentage versus block size. Note that loss of as high as 27% is observed.

Since, all the above mentioned loss is occurring inside IP, it is left unnoticed by UDP and hence UDP cannot inform the user about the reason for the loss of data. This again allows UDP to accept data at faster rate from the user. We observed in the time domain graphs of UDP this erratic patterns. To avoid UDP from sending the data at high rates, we incorporated a flow control algorithm inside the user program of UDP to send data at fixed rate depending on the type of the network underneath. The algorithm we implemented sends packet at regular intervals of the allowed rate (in our case we experimented with the assumption of 70Mbps bandwidth being allocated to the application).

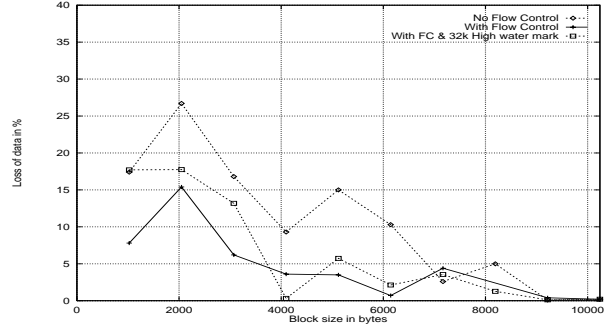


Figure 6: UDP loss at different block sizes

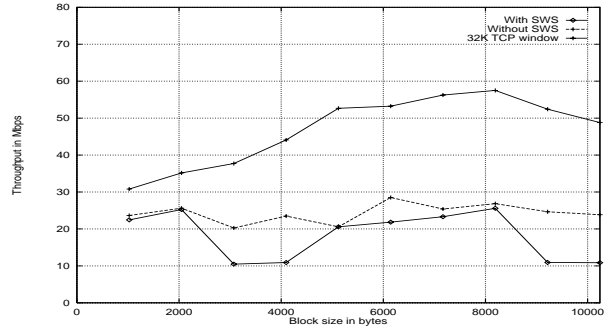


Figure 7: TCP throughput for DC application

This simple algorithm reduced the loss percentage to almost 15% from 27%. We do agree that this method is not user transparent. Hence, if this algorithm is incorporated inside the UDP protocol itself on per connection basis, we predict that the loss percentage can be minimized.

5 Improvement in DC application

With the simulated DC application we send 5Mbytes of data from either of the nodes using *send process* every second. This generates about 72 Mbps of data on the network. At the same time a moderately computational intensive process (which when run alone occupies about 30% of CPU time) is run using *computation process*. We measure the effect of the control parameters as mentioned in the previous section on this application.

A decrease in throughput by nearly 20% in all the cases (Fig.7) shows the effect of sharing the CPU by both computation and communication processes. Effect of SWS is reduced considerably because of the breakup in data transfer, unlike in the previous case where data is submitted continuously. In UDP case

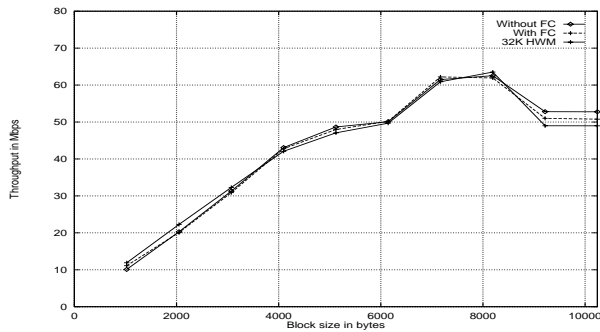


Figure 8: UDP throughput for DC application

very little difference between the three cases is observed (Fig.8) because of the decrease in loss of data. Loss of data Fig.9 though approximately following the previous profile is reduced because of the reduced communication processing time.

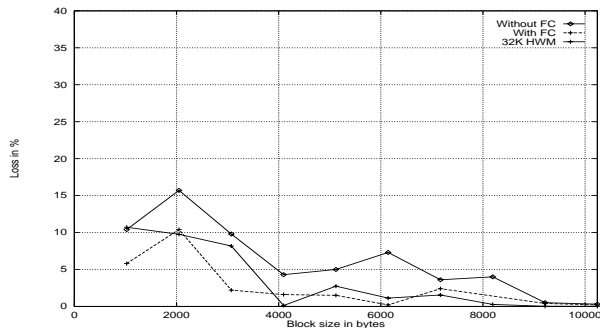


Figure 9: UDP loss in DC application

6 Conclusions

We analysed TCP-IP and UDP-IP with respect to user observable QOS parameters, namely connection establishment time, throughput, and loss. It can be seen from the results we have in this environment that TCP can be tuned to behave as less overhead protocol as UDP. We also brought out the reasons behind loss of data in case of UDP, and very low throughput in case of TCP. We demonstrated how a simple rate control algorithm at the user level can reduce the loss by nearly 10%. We presented how avoiding SWS and increasing the window size can improve the throughput in case of TCP.

A user with some knowledge of the application can change the above discussed control parameters and mould TCP or UDP to his needs. For example, a bulk data transfer application which is less delay sensitive can fix the point of operation where the throughput is high in case of TCP/IP, and high throughput and less loss in case of UDP/IP. A DC application can fix the

point of operation where the throughput is sufficient, connection establishment time and loss are minimal. We used DC application to demonstrate the results.

References

- [1] Boggs, D.R., J.C. Moughl, and C.A.Kent [1988]. "Measured Capacity of an Ethernet Myths Reality," Proc. ACM SIGCOMM'88, Stanford, Calif., Aug 1988, pp 222-234.
- [2] Jain, R. [1990]. "Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT," Proc. ACM SIGCOMM'90, Philadelphia, Sept 1990, pp 264-275.
- [3] Rao, Nageswara S.V., Maly, K., Olariu, S., Sudheer, D., Zhang, L., and Game, D. [1994]. "Average Waiting Time Profiles of Uniform DQDB Model," Proc. INFOCOMM'94, Toronto, June 1994, pp 1326-1335.
- [4] Sykas, E. D, Vlakos, K. M, and Hillyard, M. J [1991]. "Overview of ATM networks: functions and procedures," Computer communications review, Vol. 14, No. 10, Dec 1991, pp 615-626.
- [5] Angnostou, M. E et al. [1991]. "Quality of service requirements in ATM-based B-ISDNs," Computer communications review, Vol 14, No. 4, May 1991, pp 197-204.
- [6] "ATM: User-Network Interface specification - version 3.0," The ATM-Forum, Prentice Hall Publications limited - 1993.
- [7] Sykas, E. D, Vlakos, K. M, and Anerousis, N [1991]. "Performance evaluation of statistical multiplexing schemes in ATM networks," Computer communications review, Vol. 14, No. 7, Oct 1991.
- [8] C. Papadopoulos, G. M. Parulkar, "Experimental evaluation of SUNOS IPC and TCP/IP protocol implementation", IEEE/ACM Transactions on Networks Vol 1, no. 2, April 1993.
- [9] William Gropp, and Barry Smith, "User Manual for the Chameleon Parallel Programming Tools", Argonne National Laboratory, June 1993.
- [10] SunOS 5.1, STREAMS Programmer's Guide, SunSoft Technical Manual.
- [11] Clark, D.D. "Window and Acknowledgment Strategy in TCP," RFC813.