

Missing end-system QoS components: A case-study

Kurt Maly Sudheer Dharanikota
Ravi Mukkamala C M Overstreet

Department of Computer Science
Old Dominion University
Norfolk VA-23529

Phone number: (804)-683-3915

Fax number: (804)-683-4900

E-mail addresses: {maly, dhara_s, mukka, cmo}@cs.odu.edu

Contact e-mail address: dhara_s@cs.odu.edu

Abstract

In this paper, we propose a Quality of Service (QoS) architecture for an end-system protocol-suite. We use TCP/IP using ATM as the networking paradigm, as a testbed to propose our QoS architecture. With the help of no-load condition, host-load condition, and network-load condition experiments, we identify the QoS perturbations caused in such an environment. We analyze these results behind the QoS perturbations, and use them to arrive at the missing components in the current protocol architecture.

We use TCP/IP/AAL5/ATM, and AAL5/ATM as two performance comparing protocol suites to obtain knowledge on the missing QoS components. We measure the application-level QoS in terms of throughput, delay, round trip time, and loss to identify the base-line performance an application can expect from such an environment. From the no-load condition we measure the behavior of these protocols at various data rates and user submitted data block sizes. We demonstrate the trade-offs involved in obtaining high throughput, low delays, low Round Trip Time, and zero losses at different data rates. We use host-load condition experiments to understand the interaction between the CPU-intensive jobs and the communication-intensive jobs. We use network-load condition experiments to observe interaction between multiple streams of the above two protocol-suites, and its effect on the application QoS.

Key Words: *Predictable performance, Control parameters, QoS, TCP, IP, AAL5, ATM*

1 Introduction

All applications, from complex collaborative applications such as “Video Collaborative (VC)” application, to simple applications such as “ftp” application, expect *predictable performance*. The terms predictability, and performance need some explanation in the context of this work. An application behaves predictably when *the application user observes the statistical behavior of the application, and it is consistent with the requested behavior by him*. The term performance is a relative term, which is *a measure of the behavior of an application*. So an application is said to be performing predictably when it is in the behavioral range as approved by the user. An example to understand these concepts is given in the following paragraphs.

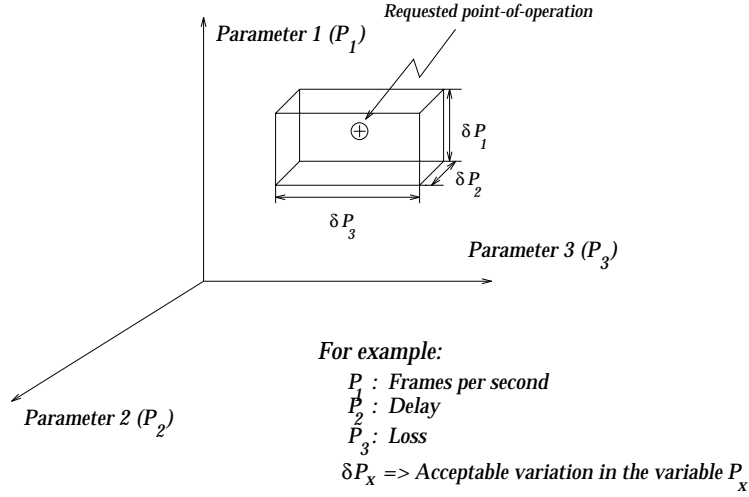


Figure 1: Meaning of point-of-operation with an example

Consider a *Video Collaborative* (VC) application in which audio, and video are sent across the network among a group of collaborative users (e.g., a teacher, and students of the class). A user in such an environment requires a set of performance measures (Quality of Service parameters) to be met to use this application. For example, a user may be interested in the number of frames received per second (FPS), latency, or delay between the sender and the receiver, for a good on-line interaction, and less loss and jitter in terms of user perceivable range. These requirements decide the point-of-operation for the application (refer to Figure 1). Most of the time a small deviation from this point-of-operation is acceptable by many applications, as shown in the figure. The volume encompassed by such a requirements is called the range of operation in this work. We can say that the application is performing predictably, if it's performance measures are within the acceptable range, as defined by the user; as shown in Figure 1.

Deviation from the point-of-operation manifest differently in different applications; in a VC application this may reflect as reduction in the number of frames per second (FPS), and in an ftp application this may reflect as reduction in the throughput. This degradation in the application performance is because of three reasons: *end-system protocol behavior in high speed networks* (HSN), *the host load condition*, and *the network load condition*.

In this paper, we address the problems an end-to-end application encounters under the above three conditions. In our first experiment we run a test application between two Sparc 10s under no CPU, and network load condition (no-load condition); these results are used to identify the behavior of the end-system protocols for different control parameters, and host machine limitations. In the second set of experiments we load the receiver side CPU to test its effect on the application behavior. And, in the third set of experiments, we observe the effect of network-load on the end-system application behavior. We show how these degradations manifest into Quality of Service (QoS) perturbations in the application, with the help of the QoS measures such as throughput, delay, Round Trip Time (RTT), and loss. With the help of the results obtained in the above experiments we identify the missing components in the current technology for the end-system QoS guarantee.

To understand an application behavior qualitatively, we use an ATM LAN, with TCP/IP as the end-system protocol-suite. We experiment with TCP/IP/AAL5/ATM (we refer to it as *TCP*), and AAL5/ATM (we refer to it as *direct*) protocols under no-load, various CPU, and network load conditions. For an application using such an environment, we fix a point-of-operation to different values of *target offered load*, and observe how QoS measures are effected to the varying protocol control parameters. These

protocol control parameters include, in case of *direct*, the application requested bandwidth (or data rate), and the application submitted data-block size; in case of *TCP*, these control parameters include MSS (Maximum Segment Size), the receive and the send window sizes, and the user block size. In this work we fix the TCP window sizes to 64 KBytes, and the MSS to 9148 Bytes to obtain maximum throughput. Refer to [1, 2] for more details on the relation between the window size, MSS, and user block size.

The organization of the paper is as follows: in section 2 we present the testbed used for this work, and discuss the relevant background to understand the protocol behavior in different experiments. Section 3 present a discussion on the effect of protocol behavior, host CPU load, and network load conditions on the application QoS. We use these results to deduct the modifications needed in the new generation of end-system protocols in section 4. Conclusions and future work are presented in the last section.

2 Testbed and application

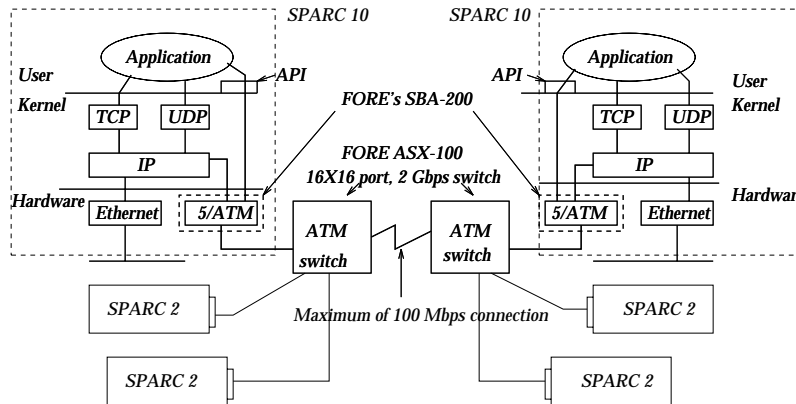


Figure 2: A detailed testbed used in this work

Figure 2 presents the testbed used in this work. We have two 16 X 16 port Fore Systems ASX-100 switches connected in tandem. A SUNsparc 10, two SUNsparc 2 workstations are connected to each of the switches. These workstations use Fore Systems SBA-200 ATM cards. The maximum bandwidth between the workstation and the switch, and between the switches is 100 Mbps. We ran the application both through TCP/IP running over AAL5/ATM, and through AAL5/ATM [3, 4]. When the application runs over TCP it uses an end-to-end ABR (Available Bit Rate) [5] connection, whereas when it uses direct AAL5 it has the option of selecting either a CBR (Constant Bit Rate) or an ABR connection. The application running in these experiments merely generates data for transmission, but with explicitly controlled properties. The command-line options to this application are the size of the user data block, the number of user data blocks sent, and the rate at which these blocks are sent (in *direct* case, the last parameter will be the peak bandwidth reserved for this application). When it is running on AAL5 directly it can reserve the requested bandwidth in the case of CBR. We use only the two SUNsparc 10 workstations for no-load, and CPU-load condition experiments. For network-load condition experiments, we use all the six workstations to load the 100 Mbps channel between the switches.

On the receiver side of this application, we measure throughput, delay, and loss (number of retransmissions in case of TCP), and on the sender side we measure RTT. These parameters capture the dynamic behavior of the host, and the network (refer to Figure 3). In all the three experiments we use these QoS measures to identify bottlenecks to obtain predictable performance for the end-user application. *Throughput* is the amount data received per second, which is measured on every N^{th} packet (as shown in Figure 3), and averaged over the total communication time. *Throughput* requirement of an application reflects the

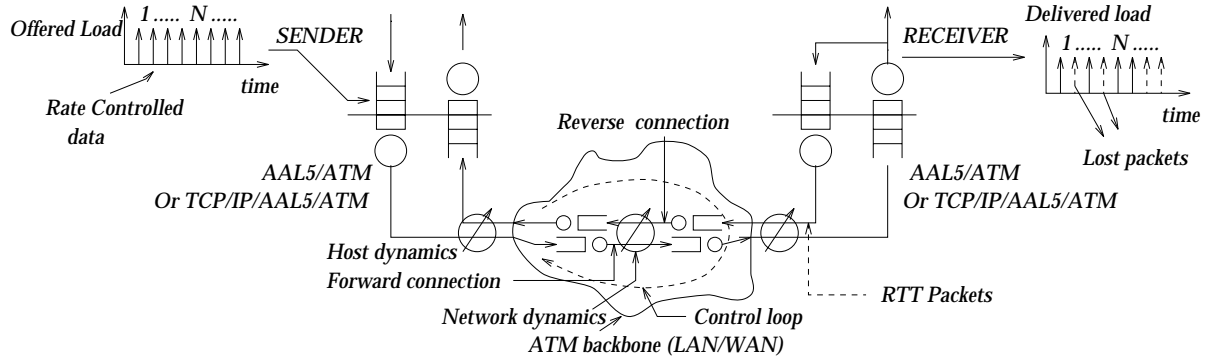


Figure 3: An insight into the QoS parameter significance on an ATM network

amount of host, and network buffers occupied by the application. In ATM-like networks, this parameter is also a measure of the amount of bandwidth to be reserved for this application. The maximum observed *throughput* on a host machine reflects some of the system deficiencies such as lesser number of system buffers, host-network interface problems etc. A counterpart to *throughput* is *loss*, which also reflects the above mentioned system deficiencies. *Loss* in our work is defined as the percentage of data lost during the total communication time. We identify data lost on the receive side as shown in Figure 3, to calculate the loss percentage. *Loss*, in case of TCP, is observed as more number of retransmissions which shrinks the TCP congestion window, which in turn reduces the end-to-end throughput. This parameter in case of UDP, or AAL5/ATM connection reflects reduced throughput due to unpredictable loss of data. *Delay* is the average delay incurred for N packets, which is averaged over the total communication time. *Delay* parameter is a measure of the queuing delays in the end-system protocol suite, delay at the host-network interface, and delay on the network. This parameter also reflects the dynamic behavior of the end-system protocol suite, such as the TCP-retransmission algorithm. *RTT* is the amount of time taken for the N^{th} packet to reach the receiver, and back to the sender; the average *RTT* is calculated over the total communication period. *RTT* gives the size of the control loop of the application, as shown in Figure 3. In ATM-like networks, *RTT* tells the network element what sort of trade-offs it can make between responsiveness, buffer size requirements, bandwidth available, number of connections it can support, etc. [5].

Figure 4 shows the interaction between TCP(UDP)/IP and AAL5/ATM in a LAN environment [6]. TCP, and UDP maintain per-connection based information. In TCP the control parameters such as send and receive windows, MSS, timers, and algorithms such as slow start and Nagle's algorithm are maintained on per-connection basis. These parameters are used to make an end-to-end user connection lossless, and react to congested host and network environment. Unlike TCP, UDP does not have elaborate control parameters to make its connections lossless. Hence, it depends on the underlying protocols to behave as lossless as possible.

Both TCP, and UDP send its data to IP, where no distinction is made between the connections from the two protocols. As a result, IP might generate varying delay to different connections, also losses are unpredictable. Loss of data in IP is never informed to its user. Hence, these losses in case of UDP manifest into loss of application packets. In TCP [7, 8] it is observed at the end of one or more RTTs, which will reflect as in the retransmission algorithm and hence the reduction in throughput. The congestion control algorithms in TCP may not be as helpful as the congestion avoidance algorithms such as slow start [10]. This is because of the high bandwidth to delay product on ATM networks. The ATM community is looking at the rate-based [9], and the credit-based [11] congestion avoidance algorithms [12].

In our environment IP feeds this data to ATM Adaptation Layer 5 (AAL5). A clear picture of the data

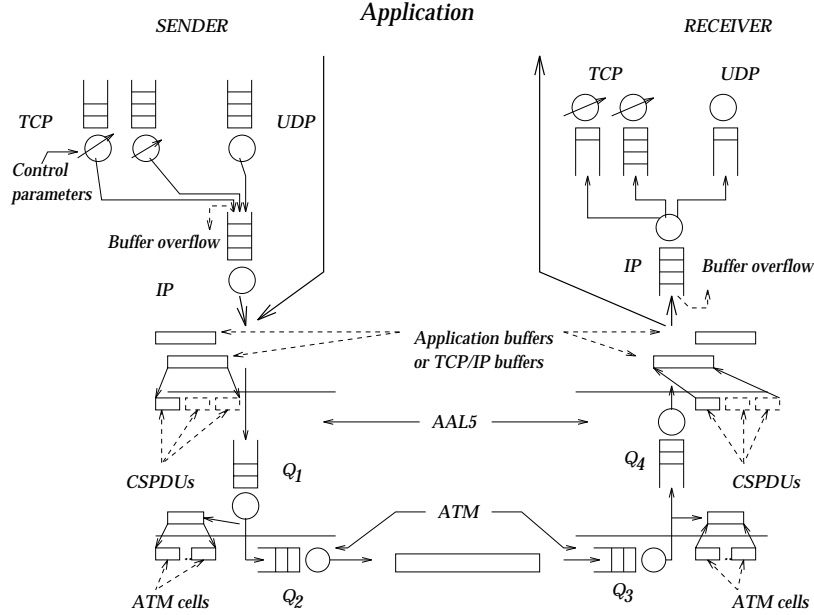


Figure 4: End-to-end flow of data on an ATM network

flow through the end-to-end AAL5s is given in Figure 4. A user data block submitted to AAL5 (either through TCP/IP, or directly to AAL5) is segmented into 9148 byte (most of the time) AAL5 CSPDUs [3] before queuing in Q_1 as shown in the figure. If the user data is not rate controlled then the Q_1 might overflow resulting in losing data on the sender side itself. These CSPDUs are segmented into 48 Byte cells and are queued in the ATM layer (Q_2) to be sent across the network. If no *per ATM connection queues* exist, even Q_2 might overflow if the combined data rates of the applications are high compared to the number of buffers allocated to Q_2 . Loss of cells might occur on the network if the application exceeds the requested bandwidth or if the network is congested. On the receive side, the cells are reassembled into a CSPDU by ATM and is given to AAL5. AAL5 then assembles these CSPDUs to form a user packet. CSPDUs are dropped in ATM (at Q_3) if one or more cells belonging to a CSPDU is lost, this loss of CSPDU results in dropping the user packet (at Q_4).

3 QoS perturbing factors

A user application requested QoS is perturbed by the protocol behavior at different host and network load conditions, and at various data rate conditions. In this section we explore the behavior of *TCP* and *direct* connections under such conditions. We chose requested bandwidth to represent different classes of applications. For example, *0.5 - 1.5 Mbps* represents MPEG-1 compressed video stream, and *10 - 65 Mbps* represents high data rate applications. We studied the above two classes of applications to gain more insight into the behavior of direct ATM versus TCP/ATM connections.

3.1 No-load condition experiments

In this paper, the behavior of the protocol observed at various requested data rates is referred as no-load experiments. With the help of these experiments we can identify the behavior of the protocols for different control parameters, and the host machine limitations. We use this information to suggest the simple, but effective modifications required in the current protocols.

In [1] we conducted a study on the effect of control parameters on the QoS of an application in the current environment. In this work we evaluated the interaction between the UDP(TCP) with IP. We reported that most of the losses in UDP occur because of the buffer overflow in IP (as shown in Figure 4). Since there is no back-pressure from IP to UDP (or to TCP) on the end-systems, these losses are unreported to UDP. We also demonstrated that losses can be reduced considerably, using a simple rate control on the sender side. When multiple connections are processed through the end-system assigning the appropriate rate to an application is a challenge. From these results we can conclude two points; firstly, to reduce the losses in IP, a simple **back-pressure algorithms should be devised between UDP (or TCP), and IP (load feedback)**; secondly, a **rate-control algorithm** must be devised, which allocates a data rate to an application depending on the QoS need of the application. The second requirement is dependent on many system dependent factors such as the total number of buffers available on the system, the maximum data rate supported by the system etc. We answer some of the system dependent questions in our no-load experiments.

Common		<i>direct</i> related					<i>TCP</i> related	
Requested bandwidth in Mbps	Application Block size in KBytes	ATM delivered throughput in Mbps	AAL5/ATM sender statistics	AAL5/ATM receiver statistics	Loss in %	Delay on AAL5 in msec	Delay on TCP in msec	TCP delivered throughput in Mbps
0.5	1	0.47	12800/12109/5.40	12109/12109/0	5.40	41.6	127.2	0.49
	8	0.48	1600/1582/1.12	1582/1582/0	1.12	48.9	131.0	0.5
	64	0.49	1600/1526/4.62	1526/1526/0	5.00	128.2	131.0	0.51
1.5	1	1.40	12800/12125/5.27	12125/12125/0	5.27	15.36	41.52	1.49
	8	1.43	1600/1591/0.56	1591/1591/0	0.56	17.44	43.66	1.49
	64	1.44	1600/1544/3.50	1544/1544/0	3.50	41.34	43.6	1.51
10	1	9.26	12800/11877/7.21	11877/11877/0	7.21	3.04	5.4	9.45
	8	9.33	1600/1561/2.44	1561/1561/0	2.44	2.64	6.5	9.9
	64	9.48	1600/1480/7.50	1480/1480/0	7.50	6.45	6.54	10.09
25	1	7.32	12800/12245/4.34	12245/4132/66.26	67.72	3.63	4.4	13.47
	8	21.89	1600/1488/7.00	1488/1488/0	7.00	1.16	2.82	22.71
	64	21.06	1600/1361/14.93	1361/1361/0	15.00	2.26	2.64	24.83
65	1	5.34	12800/12800/0	12686/2144/83.10	83.25	5.34	3.	15.50
	8	62.16	1600/1600/0	1600/1600/0	0.00	0.60	1.42	43.39
	64	62.87	1600/1600/0	1600/1600/0	0.00	0.93	1.66	38.75

Table 1: Comparison of *direct* and *TCP* applications under no load condition

The no-load tests provide a set of baseline results in selecting the control parameters to obtain a specific application level QoS. For both the *TCP* and *direct* cases we measure throughput and delay, for *direct* case we also measure loss, and for selected cases of TCP we measure retransmissions. In all of these tests, we use a user level rate control mechanism to control the amount of data submitted per second to match the user-requested bandwidth.

Table 1 presents the experimental results obtained under no-load conditions for different requested bandwidths, and user data block sizes. In this set of experiments we target to offer the same amount of load as the requested bandwidth. We present the receiver side throughput, delay, and RTT (on the sender side) for both TCP and direct cases and loss in case of direct connection. The 4th, and the 5th columns in the table represent the statistics of the sending, and the receiving AAL5. The first portion of the sender side statistics represent the number of CSPDUs given to Q_1 after segmenting (if necessary) the user packet. The second portion of the column represent the actual number of CSPDUs that left

Q_1 , and the last represent the percentage of loss in Q_1 . Similarly, the first portion in the last column represent the number of CSPDUs assembled by AAL5 before placing the data into Q_4 , the second portion represents the actual number of CSPDUs passed onto the user, and the last represent the percentage of CSPDUs lost in Q_4 . An application with certain throughput, loss, and bonded delay conditions can look into the data in the table to select appropriate control parameters.

For both TCP/ATM and direct ATM cases, providing the requested *throughput* for low bandwidth cases is never a problem. *Delay* increases as user block size increases in the case of direct ATM connection because of the processing overhead. This end-to-end delay is much less in case of 1 KByte and 8 KByte block sizes compared to that of 64 Kbyte block size because no segmentation and reassembly is done in ALL5. Because of the segmentation and reassembly in AAL5 for 64 KByte block size, delay increases by almost three fold. Delay in case of TCP is high compared to that of direct ATM connections because TCP tries to accumulate MSS worth of data before sending it to ALL5. TCP delay can be reduced by reducing the window size and MSS (if possible). The *loss* of data in the direct ATM case occurs at Q_1 due to our user-level rate control which may not supply data at the exact requested rate. *Loss* can be made zero by increasing the requested bandwidth marginally more than the offered load (refer to Table 2).

For high data rate applications, the number of interrupts generated on the receive side of the application by the ATM card are high in the case of using a 1 KByte block size. This phenomenon leads to dropping packets in Q_4 if the host is not fast enough. Hence the user-level *throughput* goes down and *losses* are high. *Delay* will also increase because of higher queueing delays at Q_4 . We also noticed that using our testbed we could not observe throughput more than *65 Mbps*, which is the host limitation. By increasing the block size to 8 Kbytes we are reducing the number of CSPDUs, and hence the number of interrupts on the receiver side, which will reduce the losses to zero.

<i>direct</i> related					Common	<i>direct</i> related		<i>TCP</i> related	
<i>Target Offered load</i> <i>Requested bandwidth</i>	Block size in KBytes	Delivered load in Mbps (ATM)	RTT in msec	Delay in msec	# Blocks	CSPDUs (ATM)	Loss in %	CSPDUs (TCP)	Retx. % in TCP
0.5/0.5	1	0.47	1060.52	39.36	12800	12109	5.40	4554	0.00
0.5/0.7	1	0.49	15.61	130.88			0.00		
0.5/0.5	8	0.50	5672.59	48.9	1600	1600	1.12	1600	0.00
0.5/0.7	8	0.49	115.39	129.10			0.00		
0.5/1.0	8	0.49	69.70	130.10			0.00		
65/65	1	5.34	305.72	5.50	12800	12800	83.25	5144	13.00
65/75	1	7.30	270.41	37.52			77.41		
65/85	1	7.05	251.52	37.36			77.62		
65/65	8	62.16	4.26	0.99	1600	1600	0.00	1600	0.00
65/75	8	62.59	4.25	1.00			0.00		

Table 2: Case analysis *direct* and *TCP* applications for different block sizes

In Table 2 we present the relation between RTT, delay, and losses at different cases of target offered load, and requested bandwidth. For *0.5 Mbps* case, high *RTT* values are observed when the requested bandwidth is same as the target offered load; this is due to all the resources allocated for the duplex connection are occupied by the forward connection (refer to Figure 3) itself. When the requested bandwidth is more than the target offered load then *RTT* becomes very less because of the resources available in the reverse direction. Average *delays* has increased as more bandwidth is reserved, because of the increase in queue sizes allocated for this connection. As observed in case of 8 KByte block size, average *delays* reach saturation, for example 130.10 msec; this is because the leaky bucket rate-control algorithm in ATM is

becoming effective.

At 65 Mbps, the target offered load and the requested bandwidth does not effect the *delay*, and *RTT* for 1 Kbyte block size, due to *losses* at Q_4 because of *buffer overflow*. At 8 Kbyte block size because of the zero data *losses* the *delay* and *RTT* are comparable. These parameters cannot be improved by increasing the requested bandwidth.

In case of *TCP*, at low data rates *TCP* will not encounter *losses* and hence no retransmissions. We use this information to calculate the percentage of retransmissions in the lossy case of 65 Mbps with 1 KByte block size. We observe 13% *retransmissions*, which is less compared to 83.25% *losses* in its counterpart in the *direct* case. At the same time, *TCP* adjusts itself to a lower throughput to avoid losses. This shows the self-healing nature of *TCP* because of its elaborate flow control mechanism.

From the above observations we conclude that, the application block size and the requested bandwidth play major role as control parameters in obtaining applications requested QoS. An application using *direct* connection should clearly balance the control parameters to obtain the desired QoS. Where as, an application using *TCP* need not balance the control parameters, at the cost of not obtaining tight bounds on the application QoS. We notice that the maximum data rate a *TCP* connection supports is around 45 Mbps, where as a *direct* connection supports upto 65 Mbps, on a Sparc 10 workstation. Hence, with the given configuration of Sparc 10 workstation the combined data rate of all *TCP* applications should not exceed 45 Mbps, similarly for *direct* it should not exceed 65 Mbps.

3.2 Host behavior experiments

Common		<i>direct</i> related					<i>TCP</i> related	
Requested load in Mbps	CPU load in %	AAL5 delivered throughput in Mbps	AAL5/ATM sender statistics	AAL5/ATM receiver statistics	Loss in %	Delay on AAL5 in msec	Delay on TCP in msec	TCP delivered throughput in Mbps
0.5	20	0.48	1600/1581/1.19	1581/1581/0	1.19	49.34	131.18	0.50
	40	0.48	1600/1582/1.12	1582/1582/0	1.13	49.08	131.20	0.50
	60	0.48	1600/1581/1.19	1581/1581/0	1.19	52.98	131.10	0.50
	80	0.48	1600/1583/1.06	1583/1583/0	1.06	49.95	130.99	0.50
1.5	20	1.44	1600/1594/0.38	1594/1594/0	0.38	18.32	43.46	1.50
	40	1.44	1600/1595/0.31	1595/1595/0	0.31	18.51	43.74	1.49
	60	1.44	1600/1594/0.38	1594/1594/0	0.31	18.50	43.84	1.49
	80	1.44	1600/1594/0.38	1594/1594/0	0.38	17.75	43.74	1.50
10	20	9.33	1600/1562/2.38	1562/1562/0	2.38	2.66	6.46	9.94
	40	9.34	1600/1566/2.13	1566/1566/0	2.13	1.53	6.52	9.89
	60	9.34	1600/1567/2.06	1567/1567/0	2.06	2.66	6.61	9.88
	80	9.46	1600/1560/2.50	1560/1560/0	2.50	2.67	6.51	9.98
25	20	21.82	1600/1501/6.19	1501/1501/0	6.19	1.14	2.78	22.96
	40	21.80	1600/1507/5.81	1507/1507/0	5.81	1.20	2.69	23.61
	60	22.03	1600/1494/6.62	1494/1494/0	6.63	1.20	2.72	22.94
	80	25.92	1600/1503/6.06	1503/1503/0	6.06	1.31	2.63	23.67
65	20	43.50	1600/971/39.31	39.31	2.94	1.49	1600/1600/0	37.11
	40	45.77	1600/1600/0	1600/920/42.50	42.50	2.87	1.47	36.51
	60	49.98	1600/1600/0	1600/1095/37.90	37.19	3.07	1.81	35.85
	80	55.97	1600/1600/0	1600/1228/23.25	23.25	1.64	1.40	39.73

Table 3: Comparison of direct and TCP application at different CPU-loads

To obtain predictable performance, QoS guarantee should be assured on the end-systems too. We conduct a preliminary study on the behavior of the two protocols stacks under consideration, using host-load condition. In [13] Guru Purulkar et al. reported a study on a SunOS 4.1, in which one of their

conclusions was - in a UNIX-like Operating System a communication-intensive gain higher priority over a computation-intensive job - because of its dispatcher algorithm. In our host-experiments we conclude similar results, with the help of a software decompression of an MPEG-1 stream as a computation-intensive job, and our application as a communication-intensive job. Other communication jobs sharing the same protocol suite, and the end-system resources need a special consideration in reserving resources etc., which we will discuss in the next section.

We used 8 KByte user data blocks with TCP control parameters set to 64 KBytes of window size, and 9148 Bytes of MSS under different receive side CPU-load conditions.

From Table 3, we make the following observations: In the *0.5 - 25 Mbps* range the *losses* are mainly at Q_1 , because of the inaccurate user-level rate control at the sender. *Delay* using TCP is still higher compared to that using a direct ATM connection. The *throughput* is comparable in both the cases.

For the *65 Mbps* case *losses* shift from the send side to the receive side because the application is working in the maximum machine throughput range, and a slight neglect of this connection leads to loss of data in Q_4 . TCP on the other hand adjusts itself to lower *throughput* in the process of reducing the number of retransmissions because of the losses on the receive side. AAL5 throughput is still higher than TCP but this might be a bad option in such cases because of random losses. TCP has a better *delay* characteristics because it dynamically adjusts itself to the lossy behavior of the communication path.

3.3 Network behavior

Dynamic behavior of an end-system protocol can be well understood at various network-load conditions. These experiments reflect the deficiencies in such protocols. We use 4 different sets of experiments to evaluate the interaction of *TCP* with *direct* connections under network-load conditions. The conclusions from these experiments are used to identify the missing components in TCP/IP-like end-system protocol-suite. To be able to compare appropriate results, we used *ABR* connection for *direct* case in the following experiments.

In our first experiment, results of which are shown in Figure 5, we use a *TCP* connection between the two SunSparc 10 workstations, and two *direct* connections between the SunSparc 2 workstations. The *TCP* connection is rate-controlled to produce 45 Mbps, where as the *direct* connections are tuned for 35 Mbps each. Figure 5 presents the throughput, delay, and loss values of the above three connections over a period of time.

As can be observed from the throughput graph in Figure 5, *TCP* behaves poorly in combination with the other *direct* connections-even though almost 30 Mbps bandwidth is available. This is because of the TCP retransmission algorithm. As a consequence of low throughput, the average delay for this connection is high till the *direct* connections are active. As soon as the *direct* connections are closed *TCP* picks up high data rate, and hence lower average delays are also observed. Even at lower data rates of *direct* connection, *TCP* performs the same way reflecting the deficiencies in its retransmission algorithms. *Direct* connection on the other hand, works at higher data rates, because it does not have error-recovery. Losses in *direct* connection occurs in bursts as shown in figure.

From this experiment we can make two conclusions. The **retransmission timers of TCP need to be adjusted** properly for a given networking environment. And, to obtain maximum utilization of the network resources, some sort of **resource allocation** has to be done on per TCP connection basis.

We conduct the next experiment with two *TCP* connections each using two SunSparc 2 workstations, and a *direct* connection between Sparc 10 workstations. The *TCP* connections are rate-controlled for 30 Mbps each and the *direct* connection is rate-controlled to produce 65 Mbps. In Figure 6, we present the

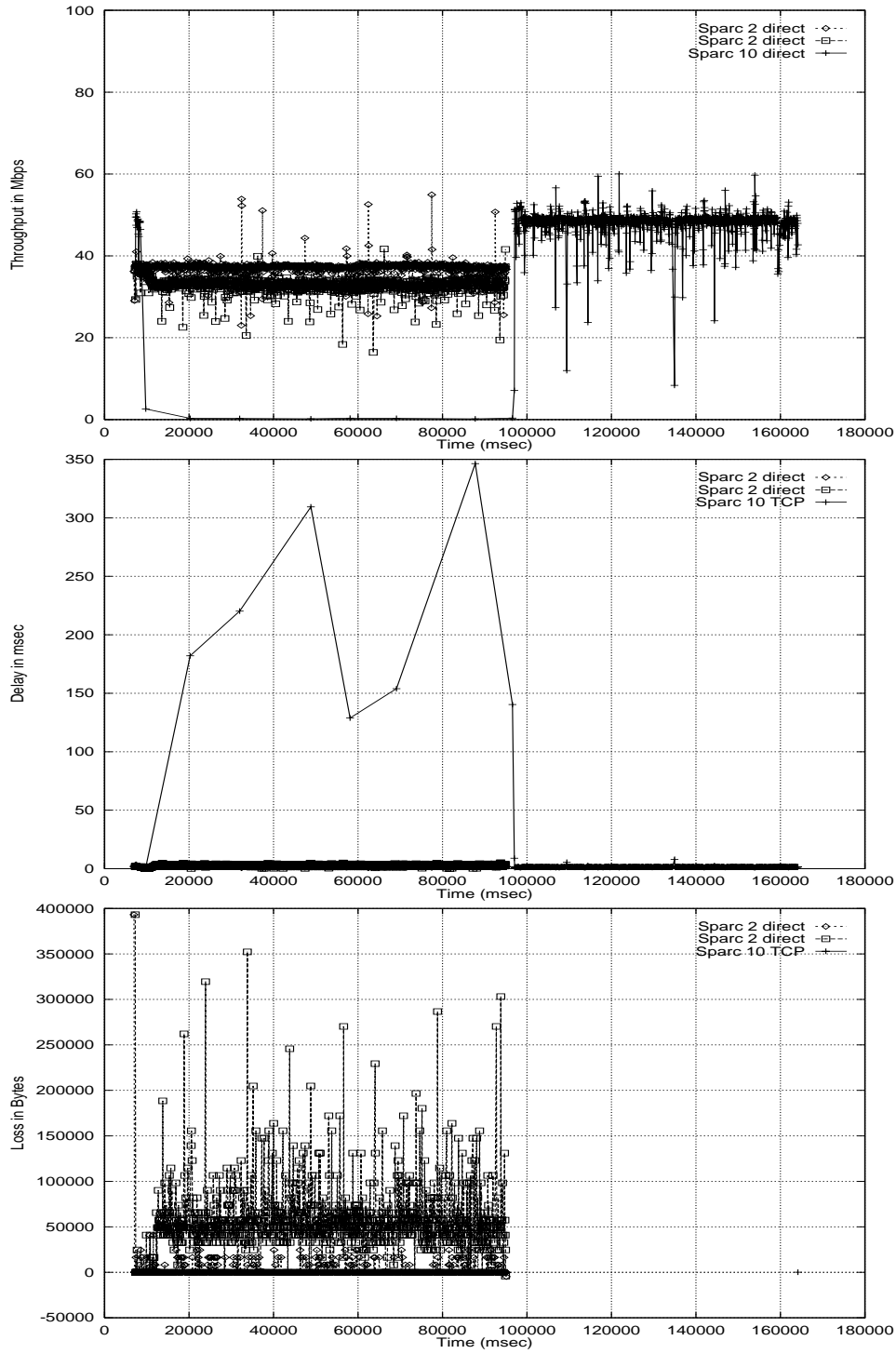


Figure 5: One *TCP* at 45 Mbps, and two *direct* at 35 Mbps

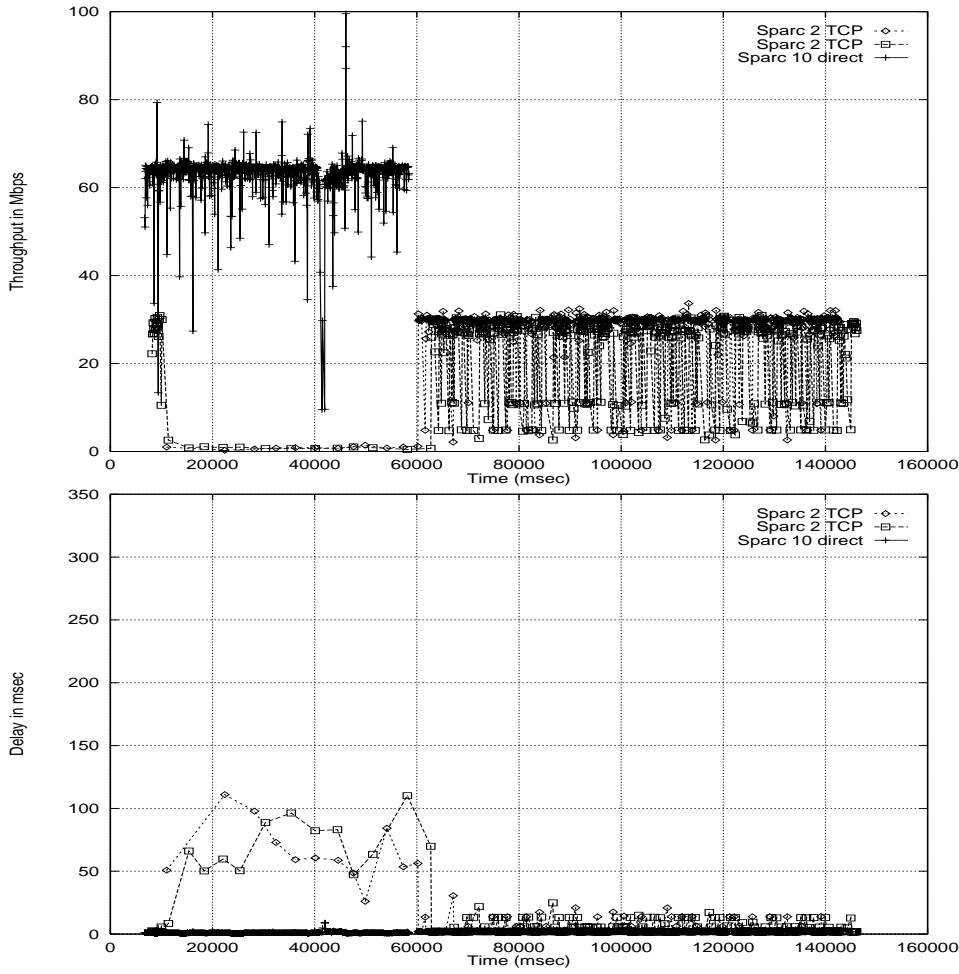


Figure 6: Two *TCP* at 35 Mbps, and one *direct* at 65 Mbps

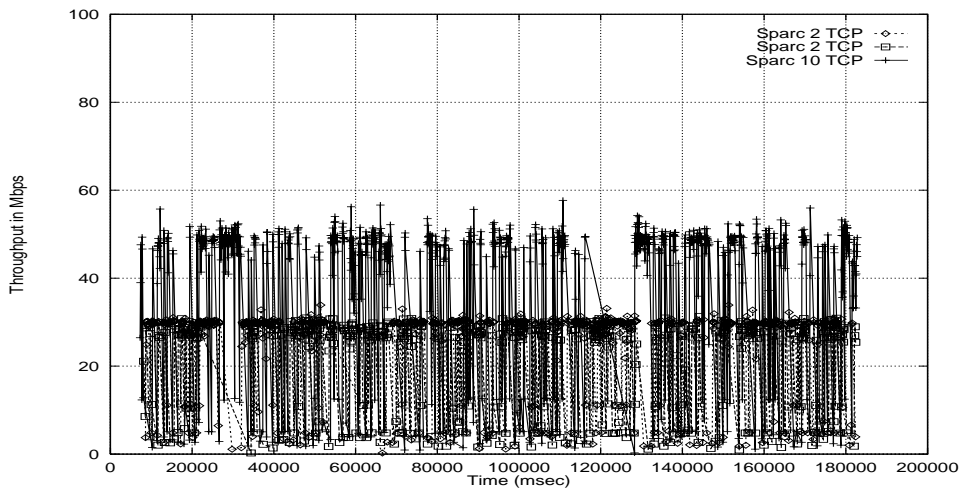


Figure 7: Three *TCP* connections one at 45 Mbps and two at 35 Mbps

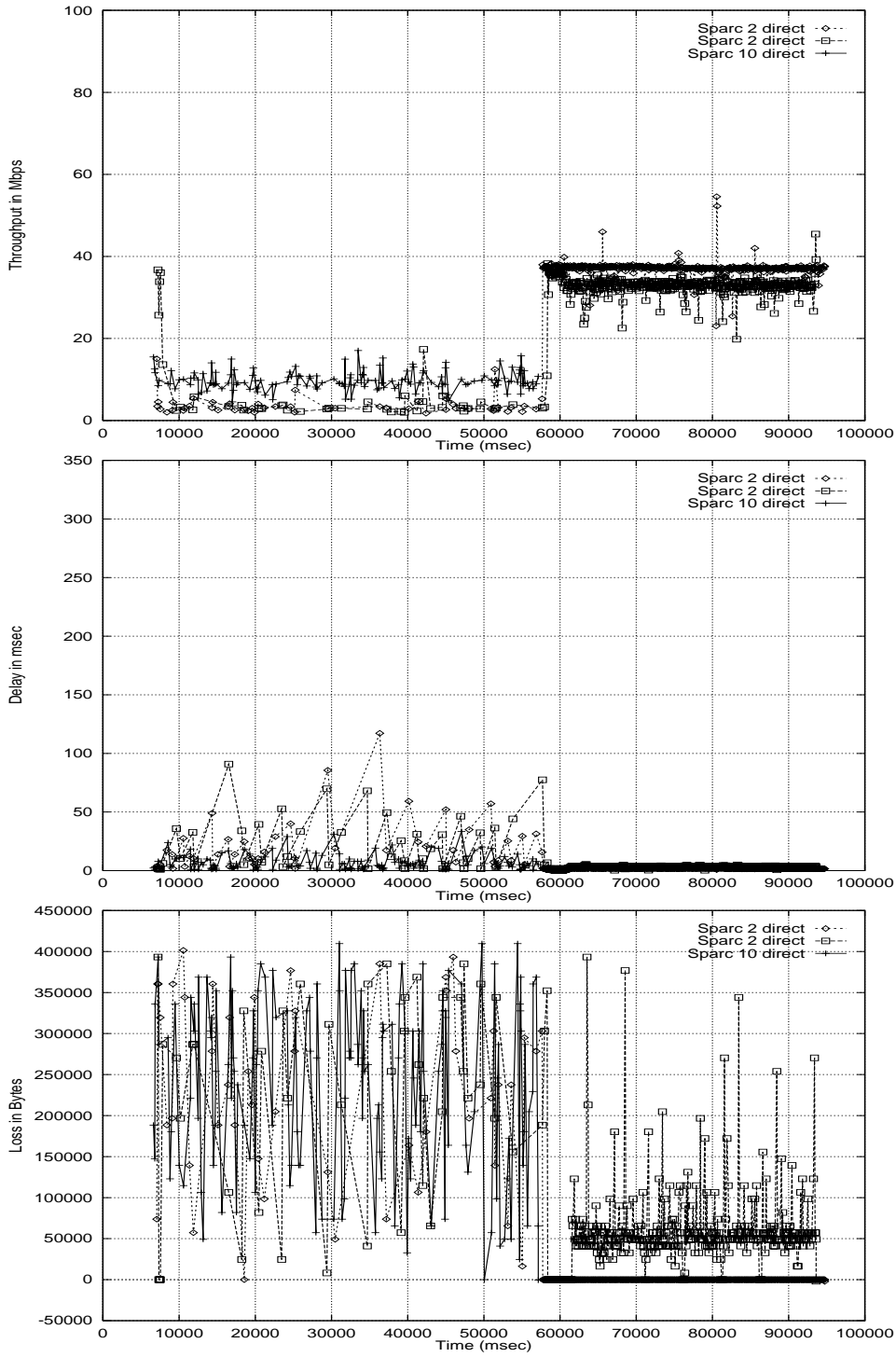


Figure 8: Three *direct* connections one at 65 Mbps and two at 35 Mbps

throughput, and the delay results as a function of time.

We observe that both the *TCP* connections produce very low throughput, because of the interfering traffic on the network; and as a consequence, higher delays are observed. After the *direct* connection is terminated, both the *TCP* connections compete for the resources on network (in our case, the output port buffers on the ATM switch). This causes varying delay, and also regular throughput dips for both the connections, as shown in Figure 6. To obtain a bounded delay for any of these connections, we need to **reserved certain bandwidth for each connection**; which consolidates the outcome of our previous experiment. To reserve bandwidth a connection should know at what data rate it is submitting data to the network. This requirement need **rate-control** some where in the path of the application on the end-system.

A prolonged effect of throughput dips can be observed from the throughput graph in Figure 7. In this experiment we use three *TCP* connections, with one of them rate-controlled at 45 Mbps and the other two are rate-controlled at 35 Mbps. For the graph we can make a note of the fact that the higher the number of *TCP* connections sharing the resources, higher the degradation in the throughput characteristics of the applications.

By comparing the throughput graphs of Figures 6 & 7, we can make an interesting conclusion. Though the average throughput for the three *TCP* connections in Figure 7 are low, they are better compared to that in Figure 6. This reflects that TCP's retransmission algorithm performs better when it is interacting with other TCP connections. But when TCP shares network resources (in this case the output buffers on the ATM switch) with ATM-like traffic it suffers. Because TCP relies on the other connections observing loss should take some action, such as backing off for certain duration. Hence, though the available bandwidth on the network is more than the resultant bandwidth, TCP connection cannot utilize the slack bandwidth. This observation consolidates the idea of **reserving bandwidth on per TCP connection basis**.

To predict the behavior of a connectionless transport protocol such as UDP in our environment, we conduct our 4th experiment. In this experiment, we use three *direct ABR* connections, one running at 65 Mbps and the other two running at 35 Mbps. Figure 8 presents the throughput, the delay, and the loss characteristics of these applications with respect to time.

We can observe from the graphs that, when the aggregate throughput of the three connections is more than the capacity of the channel (here 100 Mbps), there is a chaos on the network. Delays are high as a resultant of queuing on the ATM switch. Hence, the losses are high for all the three connections, resulting in lower throughput. Such characteristics, disturbs the QoS requirement of all the three connections. When one of the connections complete its transmission, normal status is achieved on the network. From this experiment we can conclude that, in a LAN environment, by preventing the use of connections whose aggregate results into chaos, we can sustain the requested QoS on all the connections. Another solution to this problem is to degrade the QoS of all the connections with the help of **feedback from the network**. By assuring all the applications are in the range of predictable performance, and by avoiding the network from reaching its maximum bandwidth limitation we can rectify this situation dynamically.

4 QoS architecture

In this section, we present a higher-level preliminary design of the end-system QoS architecture. This is based on the conclusions we made in the previous section. Our architecture has three main components: namely, application-level, protocol-level, and global components (as shown in Figure 9).

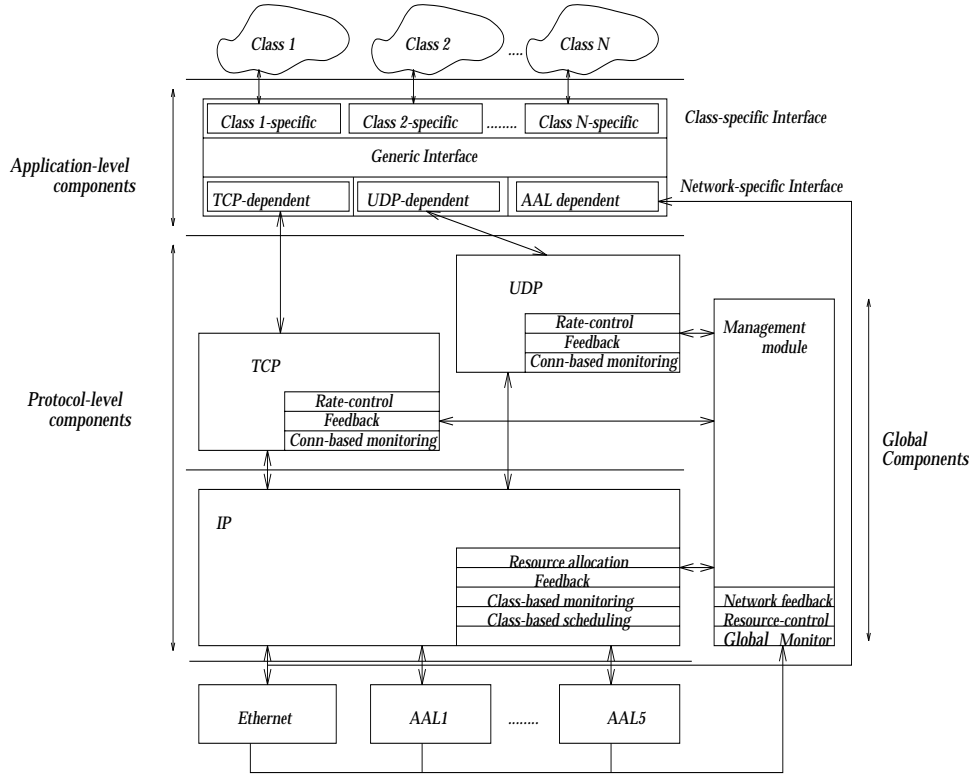


Figure 9: End-system QoS architecture

Application-level components

We divide applications into different classes depending on their QoS requirements. Each class of application talk to its corresponding module in the class-specific interface module in the application-level component. These QoS requirements are translated into a generic set of QoS requirements by the generic interface. These generic requirements are translated into a set of protocol-related control parameters, with the help of the results we presented in the previous section and in [1]. These control parameters are sent to the service-providing protocol to set the appropriate values for this connection. This interface will also help an application to dynamically adjust its QoS requirements depending on the network, and the host status, with the help of the feedback from the service-provider. Design details of this interface is out of scope of this paper.

Protocol-level components

From the host, and network behavior experiments, to provide predictable performance to an application, the following components are essential in the end-system protocol architecture.

- *Rate-control algorithm:* Both TCP, and UDP require a connection-based rate-control algorithm, to limit the user to behave in its requested QoS. In TCP, this algorithm works below window-based retransmission scheme to retain the flavor of the exiting TCP. In UDP, rate-control prevents user from using higher data rate than the agreed upon data rate, by blocking the application. This scheme reduces the losses in UDP, due to uncontrolled transmission of data, which leads to buffer overflows.
- *Local feedback algorithms:* Feedback from the service-provider, such as, from IP to TCP(UDP) or

from TCP(UDP) to the application user, helps to change the control parameters, and in turn retain the user requested QoS. For example, IP feedback information could be used in UDP to reduce the data rate of an application to avoid further losses in IP.

- *Connection-based monitoring:* All the applications with specified QoS requirements should confirm to the initial negotiation. A deviation from the negotiation will effect the other connections using a common resource pool. Hence, to retain the isolation between the applications, we need to monitor the application to check if it is within its allocated resources. This monitoring is effective if it is done closest to the user application. Hence, we propose to do the monitoring on per connection basis at TCP, or UDP.
- *Resource allocation:* To avoid the complexity of handling resources on per connection basis, we provide class-based resource allocation. For example, we have real-time, and time-shared classes inside IP. In a real-time class we schedule the data to meet dead-lines, where as in a time-shared class we are interested to obtain a given throughput.
- *Class-based monitoring, and scheduling:* We provide class-based monitoring to avoid overflowing the resources, which effects the QoS of all the applications in that class. The scheduling algorithms we use in each class of applications are different. A scheduling algorithm is dependent on the combined knowledge of all the applications in its class.

Global components

To maintain information about all the resources, and to gracefully degrade the performance of an application as a result of changing status of these resources, we need global components. These global components include network feedback, resource control, and global monitoring. The network feedback information is used to monitor the status of the network and react to it to reduce loss, or control delay for an application. Global resource control is used to allocate resources dynamically to different classes of applications. And, global monitoring is used to predict the degradation in the performance of the applications and inform their class schedulers about modifying the scheduling parameters of the algorithms.

This QoS architecture is generic enough to work with a simple link-level protocol like Ethernet protocol, to the complicated protocol such as ATM in a LAN environment. Also, the algorithms we are developing in this architecture are generic enough to be incorporated in any transport-level protocols such as TCP or UDP, and network protocols such as IP.

5 Conclusions

In this paper we use an application-oriented approach to propose a QoS architecture for a TCP/IP-like end-system protocol-suite. We conducted no-load, host-load, and network-load condition experiments to identify the missing components in the current architecture of TCP/IP. These missing components include rate-based control of TCP(UDP) connections, TCP(UDP) connection-based monitoring, local feedback in TCP, UDP, and IP, resource allocation etc.

We presented the base-line QoS that can be achieved by an application in a LAN environment. We compared the behavior of an application using TCP/IP/AAL5/ATM and direct AAL5/ATM with respect to their control parameters. We identified the bottlenecks an unwary user might encounter, such as, high delay at higher block size, heavy losses for 1 KByte block sizes at high data rates, relation between the requested bandwidth and target offered load etc. We demonstrated the trade-offs between the QoS

parameters with the help of the control parameters, such as obtaining zero loss, reducing the RTT etc. Also, an application using certain throughput, loss and bounded delay requirements can refer to the tables presented in this paper to select appropriate protocol control parameters.

It is also demonstrated in this work that TCP/IP can still be used over ATM for different classes of applications, provided appropriate resource reservation policies are incorporated into TCP and IP. TCP acts as a self-healing end-to-end protocol in many unwanted cases, such as losses at the receiver interface.

Our future work [14] include devising the above mentioned algorithms, implementing them, and comparing the performance of the architecture with the results presented in this paper. One of the important issues in devising these algorithms is the fairness in allocating the resources among the competing applications, which we consider in our future work.

References

- [1] Sudheer, D., Maly, K., and Overstreet, C. M., “*Performance evaluation of TCP(UDP)/IP over ATM networks*,” Department of Computer Science, Old Dominion University, Technical report, # TR-94-23, September, 1994.
- [2] Moldeklev, K., and Gunningberg, P., “*Deadlock situations in TCP over ATM*,” in 4th International IFIP workshop on Protocols for HSN, Vancouver, B.C.Canada, 10-12 August, 1994.
- [3] Sykas, E. D, Vlakos, K. M., and Hillyard, M. J., “*Overview of ATM networks: functions and procedures*,” in Computer communications review, Vol. 14, No. 10, Dec 1991, pp 615-626.
- [4] Boudec, J. Y. L., “*The Asynchronous Transfer Mode: a tutorial*,” in Computer Networks and ISDN Systems, 24(1992), pp 279-309.
- [5] “*ABR Signalling FAQ*,” from ATMforum signalling group discussions, June, 1995.
- [6] Fischer, W., Wallmeier, E., Worster T., Davis S., and Hayter A. “*Data Communications Using ATM: Architectures, Protocols, and Resource Management*,” in IEEE Communications, August, 1994, Vol.32, No.8, pp 24-33.
- [7] Borman, D., Braden, R., Jacobson, V., “*TCP Extensions for High Performance*,” 05/13/1992, Request for comments 1323.
- [8] Braden, D., Jacobson, V., “*TCP extensions for long-delay paths*,” 10/01/1988, Request for comments 1072.
- [9] Bonomi, F., and Fendick, K. W., “*The Rate-Based Flow Control Framework for the Available Bit Rate ATM Service*,” in IEEE Network, Vol. 9, No. 2, March/April 1995, pp 25-29.
- [10] Jacobson V., “*Congestion avoidance and control*,” in ACM Computer Communication Review, Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, vol. 18, August, 1988, pp. 314-329.
- [11] Kung, H. T., and Morris, R., “*Credit-Based Flow Control for ATM Networks*,” in IEEE Network, Vol. 9, No. 2, March/April 1995, pp 40-56.
- [12] Hong, D., and Suda, T., “*Congestion Control and Prevention in ATM Networks*,” in IEEE Network Magazine, July, 1991, pp 10-16.
- [13] Papadopoulos, C., and Parulkar, G. M., “*Experimental evaluation of SUNOS IPC and TCP/IP protocol implementation*,” in IEEE/ACM Transactions on Networks Vol 1, no. 2, April 1993.

- [14] Sudheer, D., and Maly, K. “*Providing predictable performance over High Speed Networks*,” Department of Computer Science, Old Dominion University, Technical report, # TR-95-12, June, 1995.