

QUANTA: Quality of Service Architecture for Native TCP/IP over ATM networks

Sudheer Dharanikota Kurt Maly
Department of Computer Science
Old Dominion University
Norfolk VA-23529

E-mail addresses: {dhara.s, maly}@cs.odu.edu

Abstract

In this paper, we propose a Quality of Service (QoS) architecture, QUANTA, for an end system protocol suite. We use TCP(UDP)/IP over ATM as a testbed to develop the architecture. 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.

Given these observations we define the missing components in the current protocol architectures to provide tighter control on the QoS guarantees. Components we define in QUANTA include, a two-level application to network QoS translator, protocol tuning components, local feedback component, class-based scheduling etc.

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

1. Introduction

All applications, from complex distributed applications such as "Video Collaborative (VC)" application, to simple point-to-point applications such as "ftp", 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 ob-*

serves the statistical behavior of the application, and it is consistent with the requested behavior. The term performance is a relative term, which is a *measure of the behavior of an application*. Thus, 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.

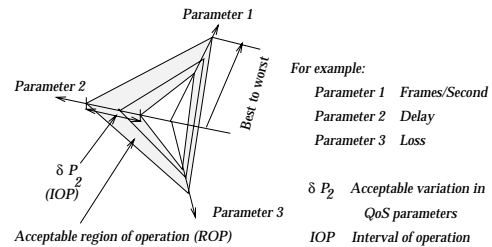


Figure 1. Acceptable ROP for an application

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 perception. These requirements decide the *Interval Of Operation (IOP)* for the application as shown using a Kiviat diagram in Figure 1. Each axis in the Kiviat diagram represent a QoS parameter. The QoS marking on these axis range from the best quality, being the innermost point, to the lowest quality, being the outermost point. As shown in the figure, these acceptable deviations create a *region of operation (ROP)*. In the diagram, the innermost triangle represent the best quality the application would expect. This is limited by factors such as the minimum delay which is introduced by the protocol suite processing overhead and the propagation delay of the signal. The outermost triangle represent

the worst performance an application would accept. We can say that the application is performing predictably, if its performance measures are within ROP, as defined by the user. Deviations in IOP manifest themselves differently in different applications; in a VC application this may be reflected as a reduction in the number of frames per second (FPS), and in an ftp application this may be reflected as a reduction in the throughput. This degradation in the application performance three causes: *end-system protocol behavior in high speed networks (HSN)*, *the host load condition*, and *the network load condition*.

In this paper, we investigate the effects of these factors on an end-to-end application. In the 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 the resultant degradations result in Quality of Service (QoS) perturbations in the application. From the results obtained in the above experiments we identify the components in the current protocol-suites missing to obtain end-system QoS guarantees.

The proposed QoS architecture is independent of the testbed machine architectures, excepting the values of the bounds imposed on the QoS by the end-system machine architecture's limitations. Some of the experiments we present in this paper are conducted on both Synoptics LattisCell 10104 and Fore Systems ASX 100 switches. The observations using either switch conform to the results we present in this paper, excepting for the maximum throughput observed at the application-level due to the differences in the design of the end-system ATM cards. We adopt to Fore Systems solution because of the API they provide to develop our own application code.

As the representative High Speed network we use an ATM LAN, with TCP/IP as the end-system protocol-suite. For the no-load condition experiments we use TCP(UDP)/IP over AAL5/ATM, and direct AAL5/ATM. We use UDP/IP over AAL5/ATM only for the first set of experiments and for the remaining experiments use only AAL5/ATM directly, as it is equivalent to the above but with lesser protocol overhead. We experiment with TCP/IP/AAL5/ATM (referred to as *TCP*), and AAL5/ATM (referred to as *direct*) protocols under no-load, various CPU, and network load conditions.

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. We discuss, in section 3, the trans-

lator which will transform the requirements of an $M : N$ application to network QoS parameters. Section 4 relates the effects 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. A summary of the QoS components is in section 5. Conclusions and future work are presented in the last section.

2. Testbed and protocols

The testbed contains 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(UDP)/IP running over AAL5/ATM, and through AAL5/ATM [5]. When the application runs over TCP(UDP) it uses an end-to-end ABR (Available Bit Rate) [2] connection, whereas when it uses direct AAL5 it has the option of selecting either a CBR (Constant Bit Rate) or an ABR connection.

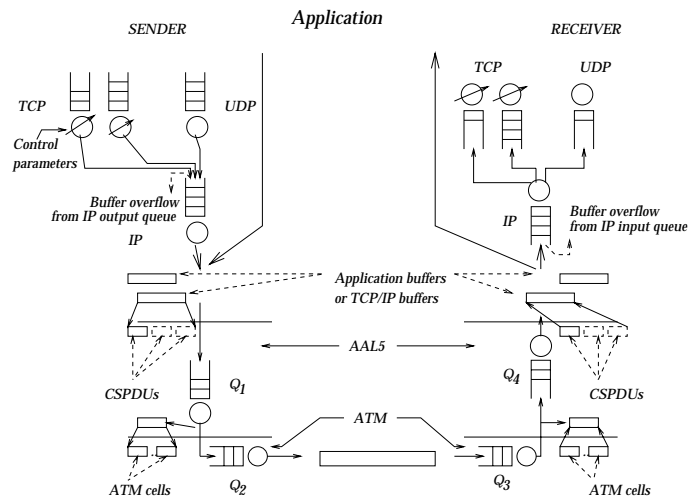


Figure 2. Flow of data in an ATM network

Figure 2 shows the interaction between different protocols and their individual behavior. Every protocol has some control parameters which can be used to adjust the dynamic behavior of the application. In TCP these control parameters include, end-to-end variables such as send, receive and congestion windows, retransmission timers etc., and algorithms such as slow start and Nagle's algorithm. These control parameters are maintained on a per-connection basis. They can be used to make an end-to-end connection react to congestion in the host and the network environments. 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. Operating System (OS) related parameters such as STREAMS parameters effect the protocol behavior and hence the dynamics of the end-user application Quality of Service. In a STREAMS protocol implementation, such as in Solaris OS, the protocol modules are linked to each other by a set of incoming and outgoing queues. These queues have lower (Low Water Mark [LWM]) and higher (High Water Mark [HWM]) limit on the amount of data they can store at any time. Once HWM of data are present at the interface queue, STREAMS back pressure algorithms prevents data from entering the queue till the accepting module drains data in the queue to reach LWM. Refer to [1] for more details.

Both TCP, and UDP send their 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 and provides unpredictable losses. IP does not inform the user of loss of data. Hence, these losses in case of UDP result in loss of application packets. In TCP [4, 6] losses are observed at the end of one or more round trip times (RTTs), which will result in retransmission 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 [9]. This is because of the high *latency X bandwidth* on ATM networks. The ATM community is looking at rate-based [3], and credit-based [10] congestion avoidance algorithms [8]. Commingling a high data rate UDP application and a TCP application, as we will observe in the following sections, will spoil the QoS of both the connections on the sending end-system because of the erratic loss of data in IP.

In our environment IP feeds these data to the ATM Adaptation Layer 5 (AAL5). A representation of the data flow through the end-to-end AAL5s is given in Figure 2. 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 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 the loss of 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* exists, 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 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 CSPDUs results in dropping the user packet (at Q_4).

Loss of data anywhere in the queues will reflect as an

QoS degradation to the end-system application user. With so many queues and dependencies, the newer generation of the protocols needs to provide a *tighter control* to be able to guarantee application QoS, which is the major consideration in this work.

3. Application's *versus* Protocol's view of QoS

An application defines the *region of operation* in terms of its user-level QoS parameters. The application expects the service provider (both the sending and receiving end-systems, and the network) to provide QoS within this region during its service. The protocol control parameters, the host-system and the network traffic pose limitations on the application performance. As a first step towards the *application-oriented QoS architecture*, we propose a component which incorporates the knowledge of the protocol control parameters and translates the application QoS into a set of protocol control parameter values. The translator sets the end-to-end protocol control parameters and OS dependent parameters, and hides the relation between these parameters from the application user.

If we have M different applications and N different protocol suites, the translation component should contain $M \times N$ translation stubs, to achieve the complete translation set. We propose a two-step translator, which translates the application QoS into a set of generic QoS parameters (such as throughput, loss, delay, RTT [*ATranslator*]), and translates these generic QoS parameters into end-system protocol-dependent control parameter values (*EPTranslator*). This method calls for $M + N$ translation components. The translation between the application to generic is dependent on the application characteristics. Hence, M increases linearly with the increase in the number of applications. To control M , we classify applications into different classes as shown in Figure 7. We group protocols into connection-oriented protocols (e.x., TCP, TP4), and connectionless protocols (e.x., IP, IPX). But the subtle differences between the protocols (such as TCP is *stream-lined* protocol, where as TP4 *maintains the integrity* of the protocol data unit) need to be treated differently as far as it concerns mapping the QoS onto control parameters.

4. Study of QoS perturbing factors

On the receiver side of this application, we measure throughput, delay, and loss (number of retransmissions in case of TCP, or the amount of data lost in case of UDP), and on the sender side we measure RTT. These parameters capture the dynamic behavior of the host, and the network (refer to Figure 3). *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 commu-

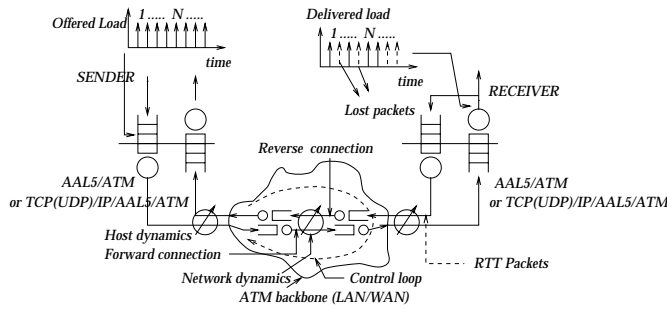


Figure 3. QoS parameter's significance

nication time. The *Throughput* requirement of an application reflects the amount of host, and network buffers occupied by the application at any given instant of time. In ATM-like networks, this parameter is also a measure of the amount of bandwidth to be reserved for this application. A low maximum observed *throughput* on a host machine implies system deficiencies such as a lesser number of system buffers and host-network interface problems. A counterpart to *throughput* is *loss*. *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 relates to 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, available bandwidth, and number of connections it can support [2].

In our experimental set-up we use uniform loading because our intention is not to provide an analysis of a particular applications, but to provide a base-line. We measured throughput as the average over a set of every 50 packets and observe the delay for the 50th packet. We ran this experiment for 200,000 packets to obtain enough knowledge about the statistical behavior of the protocol suites at high data rates and determine the average values. In the case of UDP, we calculated loss of data after every 50th packet and present loss percentages over the life time of the con-

nection. Every 50th packet is sent back to the sender to measure RTT.

4.1. No-load condition experiments

We divide the no-load experiments into two sets. In the first set, we observe the interaction between the user and the end-system protocol suite. We run the test application on TCP/IP and on UDP/IP protocol suites. We present only the throughput and the loss (in case of UDP/IP) graphs, because the delay and RTT graphs are statistically not significant due to the unpredictable behavior of these protocols (as demonstrated below). In the second set of experiments we measure the interaction between the application and the network. This is achieved by running the test application on AAL5/ATM. In the process, we also compare these results with that of TCP/IP running on top of AAL5/ATM.

4.1.1 Throughput

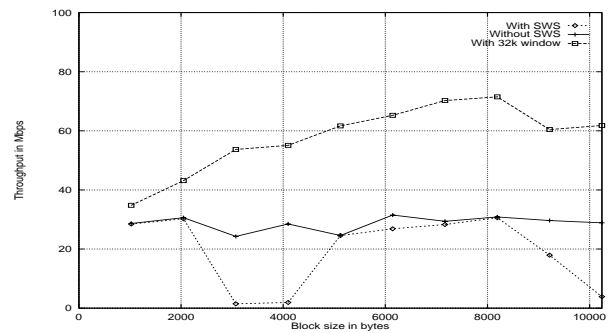


Figure 4. Effect of TCP control parameters on end-to-end application throughput

Figure 4 gives the average throughput observed on the receive side by varying block size. We recorded a maximum throughput of 30 Mbps at block size of 8192 bytes. Note the dip in throughput for the block sizes of 3072 and 4096. A phenomenon called “silly window syndrome” (SWS)[7] is the reason for this dip. It should be noted that by eliminating the SWS and increasing the TCP window size (from 8 Kbytes to 32 Kbytes) we can obtain an almost twofold improvement in throughput. As the block size is increased, the processing time per block increases, and hence the throughput decreases. These graphs show that protocol control parameters such as window size, user-submitted block size and algorithms such as SWS will control the behavior of the end-system application. Hence, this experiment suggest that one of the tasks of the *generic to network translation* component is to set these control parameters to appropriate values.

Throughput observed with UDP is high because of the smaller processing overhead of the protocol. The UDP experiment is also a good test for identifying buffer limitations and other overheads caused by the system. Maximum throughput observed on the receiving side is 70 Mbps (Figure 5), but on the sending side it is almost 120 Mbps. This difference is due to heavy loss of data in the transit when there are uneven surges in data transfer rates on the sending side (analysis of these losses are presented below). Figure 5 shows that by incorporating a rate control algorithm that submits data at regular intervals and by increasing the HWM and LWM at UDP-IP interface, data loss in UDP is drastically reduced while there is no change in the obtained throughput.

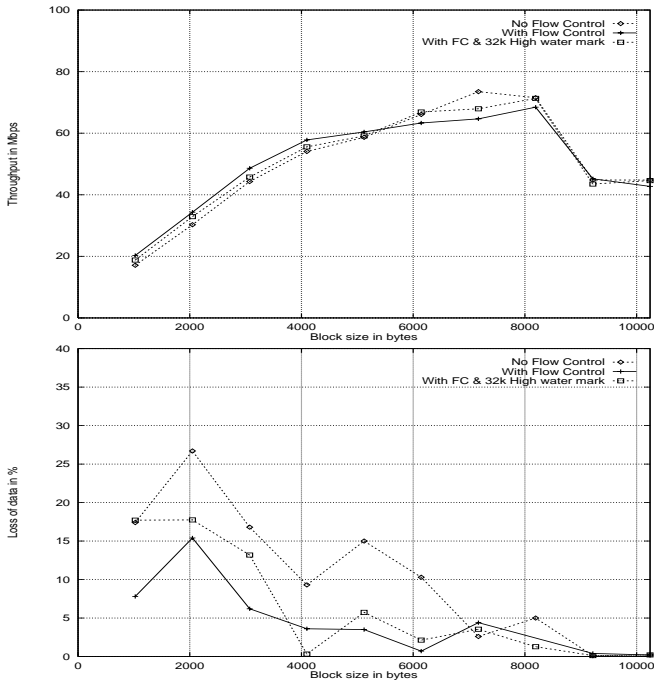


Figure 5. UDP throughput and loss versus with and without flow control, and HWM

A steady increase in UDP throughput is shown in Figure 5 until a block size of 8192 bytes. This is due to the no-processing nature of UDP. UDP submits packets as they are to IP, which in turn segments them as 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 quickly and UDP blocks until a fresh quota of buffers is available [1]. Hence, there are higher delays at the sender-network interface.

4.1.2 Loss

Loss of data at high data rates using UDP is due to resource demands on the participating systems and host-network interface rather than any network limitation. Figure 5 presents

the loss percentage versus block size. Loss rates as high as 27% are observed.

Since, loss occurs inside IP, it is left unnoticed by UDP, and hence UDP cannot inform the user about the reason for the loss of data. The loss in IP is due to the overflow of the output data queue, as shown in Figure 2. This is seen by calculating the number of IP packets dropped during this connection (We used Solaris *ndd* command to get these details.). The loss of data in the IP queue allows UDP to accept data at faster rate from the user. Such a combination of queue up and dropping of packets in UDP-IP produce erratic time domain delay patterns on the receive side.

To investigate the the impact that aspect of UDP-IP interaction has on QoS, we incorporated a rate control algorithm inside the UDP user program in order to send data at fixed rate depending on the type of the underlying network:

$$Delay = \frac{Block\ size\ in\ bytes}{Allotted\ Bandwidth\ in\ Mbps}$$

Run forever :

 SendData(data)

SendData(data)

 SubmitDataToUDP(data)

 wait(Delay - Timetaken to submit data -

Other Processing time in the runforever loop)

This simple algorithm reduced the loss percentage to almost 15% from 27%. The method clearly is not user transparent and also is not very accurate. However, were this algorithm incorporated inside the UDP protocol itself on a per connection basis, we predict that the loss percentage can be minimized. When other connections are on the same host sharing the output queue of IP, it is difficult to maintain the application QoS even with the rate-control.

From the data in the throughput and loss experiments we conclude that a proper QoS architecture needs to include a resource manager and a local feedback component to isolate the connection, as shown in Figure 7. Although we could set the initial IOP of an application in TCP using the control parameters, it is difficult to maintain it with interfering connections on the host machine. Hence, even for TCP we need to use the combination of the rate-control, the resource manager and the local feedback to provide QoS guarantees.

4.1.3 No-load case studies with direct and TCP connections

In this section we use different values of *target offered load*, and observe how QoS measures are effected by the varying protocol control parameters; in particular, we are interested in how different classes of applications over direct ATM versus TCP/ATM. The protocol control parameters for the *direct* case are the application block size and the requested bandwidth. The bandwidth reserved (and requested) for an application in *direct* case is equal to the data rate it attempts to offer(*target offered load*) using rate control. In case of

TCP, the TCP window size is set to 32 KBytes and MSS is assigned 9148 bytes.

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 the *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 intelligent QoS architecture can use these data to select appropriate control parameters. We selected the 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. Certain data are bold-faced in the table to draw reader's attention to those values (explained in the next two subsections).

Case (i): 0.5 - 1.5 Mbps applications

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 (41.6 msec and 48.9 msec) compared to that of 64 KByte block size (128.2 msec) because no segmentation and reassembly is done in AAL5. Because of the segmentation and reassembly in AAL5 for 64 KByte block size, delay increases by almost threefold. 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 AAL5. 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).

Case (ii): 10 - 65 Mbps applications

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 (5.34 Mbps for 65 Mbps case) and *losses* are high (83.25% for 65 Mbps case). *Delay* will also increase because of higher queuing 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.

In Table 2 we present the relation between RTT, delay, and losses at different level of target offered load, and requested bandwidth. For *0.5 Mbps* case, high *RTT* values are observed when the requested bandwidth is the same as the target offered load; this is due to all the resources being allocated for the duplex connection are used by the forward connection (refer to Figure 3) itself. When the requested bandwidth is more than the target offered load then the *RTT* becomes very much 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 (*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 satisfying an application's requested QoS. An application using *direct* connection should clearly balance the control parameters to obtain the desired QoS. Whereas, 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, whereas a *direct* connection supports up to 65 Mbps on a Sparc 10 workstation. Hence, with the given configuration, the combined data rate of all *TCP* applications should not exceed 45 Mbps, similarly for *direct* it should not exceed 65 Mbps.

These observations lead us to further modify the *proposed QoS architecture*, as shown in Figure 7.

The maximum observable QoS parameters on a system are bounded by the kernel-resources (such as buffers, and the host machine's architectural limitations). These values will define upper bounds on the QoS which can be requested from that host. Therefore, the *resource manager* component should consider these host limitations while allocating resources to multiple connections.

The *direct* connection analysis shows the sensitivity of QoS to network control parameters, for example, we demonstrated this in Table 2 by showing that zero loss can be obtained by reserving higher bandwidth than requested. Hence, to translate the application QoS into different backbone network protocol QoS parameters, we need a *end-system protocol to network translator (NTranslator)*. If the application QoS is sent in generic format, we can avoid two-level translation at the protocol to network interface.

From the loss analysis on the receive side (due to higher number of interrupts), we can conclude that we need to provide feedback from the network device to the resource manager. This information should be communicated to the other-side of the end-system protocol.

As the number of connections through IP increases the feedback becomes complex and even simple rate-control becomes complicated. This suggests that the control in IP should be performed on a class-basis rather than on a per-connection basis. By selecting the classes which can be recognized at the user-level itself, we can reduce the complexity of mapping between classes at different layers.

4.2. Host behavior experiments

The next set of experiments is to determine the range of impact the end-system load has on the ability to maintain predictable QoS performance. We study the behavior of *TCP* and *direct* protocol stacks, using host-load condition. Guru Purulkar et al. [11] reported a study on a SunOS 4.1, in which one of their conclusions was: in a UNIX-like Operating System a communication-intensive job receives higher priority over a computation-intensive job because of Unix's dispatcher algorithm. In our host-experiments we obtained similar results but relate them to QoS performance.

The computation-intensive job is a software decompression program of an MPEG-1 stream and the communication-intensive job is the modified *ttcp* applica-

tion. We use 8 KByte user data blocks with *TCP* control parameters set to 32 KBytes for the window size, and 9148 Bytes for MSS. For *direct* connection, we use a user data block size of 8 Kbytes, and allocate bandwidth equal to the *target offered load*. The experiment is run under different receive side CPU-load conditions. From Table 3, we make the following observations: In the 0.5 - 25 Mbps range (irrespective of the percentage of the CPU load) 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 throughput range for that machine, and even a slight CPU load 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.

The conclusion from these results is that the *resource manager* needs to reserve resources on the end-system host machines and should accept feedback from the network interface component.

4.3. Network behavior

The final component we analyze to determine what is needed in a QoS architecture to provide predictable performance is the one which relates to network behavior. The following experiment is set to determine the deficiencies in *TCP* and the interaction of *TCP* with *direct* connections under network-load conditions. We present time domain plots of throughput, and delay of the end-to-end applications. For a detailed analysis of the network-load experiments refer to our extended version of this work in [14].

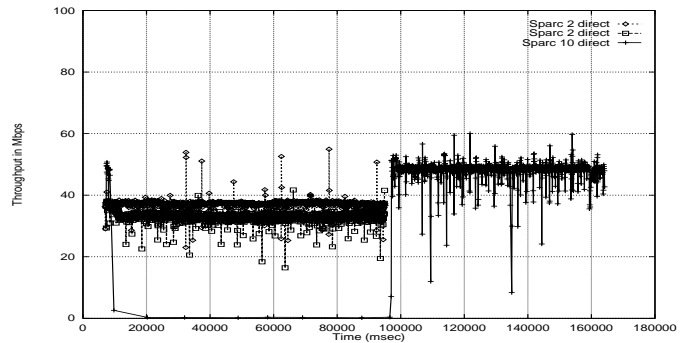


Figure 6. One *TCP* at 45 Mbps, and two *direct* at 35 Mbps

We use *ABR* connections for the *direct* case in this experiment, the results of which are shown in Figure 6. We set up 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, whereas the *direct* connections are tuned for 35 Mbps each. Figure 6 gives the throughput values for the three connections over a period of time. We intentionally overload the link between the ATM switches to observe behavior of the end-system protocols in adverse conditions and its effect on the application QoS.

As can be observed from the throughput graph in Figure 6, *TCP* behaves poorly in combination with the other *direct* connections - even though almost 30 Mbps bandwidth is available. This is because *TCP* does not obtain the information on the residual bandwidth. 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 at a high data rate, and hence lower average delays are also seen. Even at lower data rates for the *direct* connection, *TCP* performs the same way. The *direct* connection on the other hand, works at higher data rates, because it does not have elaborate error-recovery. The penalty paid for the higher data rate in the *direct* connection is the bursty losses.

This leads us to the conclusion that *resource allocation* has to be done on a per-*TCP* connection basis to obtain maximum utilization of the network resources. And the allocation should be adjusted dynamically according to the feedback from the network. These components are incorporated in the complete architecture shown in Figure 7.

5. Discussion on QoS architecture

In this section, we present a summary of the high level design of the end-system QoS architecture we propose (refer to Figure 7). It is based on the conclusions we have drawn from the experiments described in the previous sections. A complete specification and the status of its implementation can be found in [12]. The architecture has three main components: application-level, protocol-level, and global components, as shown in Figure 7.

Application-level components

We divide applications into different classes depending on their QoS requirements. Each class of applications talks to its corresponding module in the class-specific interface module in the application-level component. The 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, using data we presented in the previous section and in [13]. These control parameters are sent to the service-providing protocol to set the appropriate values for this con-

nection. The interface will also interact with an application to dynamically adjust its QoS requirements depending on the network, and the host status, using feedback from the service-provider.

Protocol-level components

Based mostly on the host, and network behavior experiments, the architecture provides the following modifications to the end-system protocol architecture:

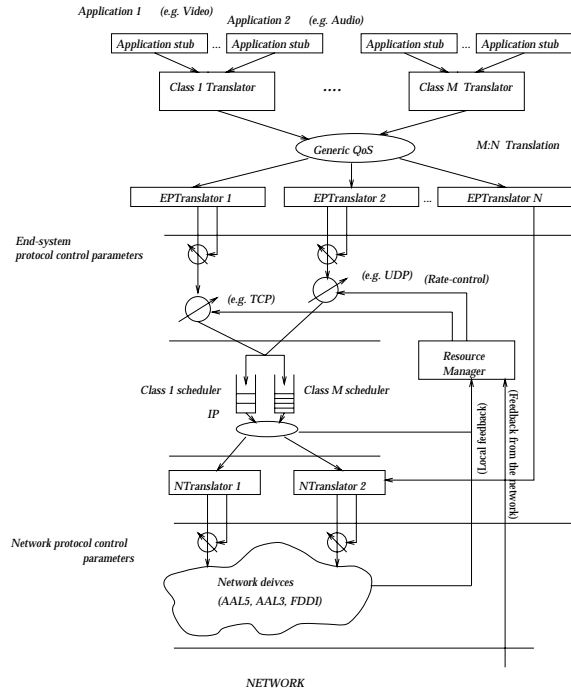


Figure 7. End-system QoS architecture

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 the window-based retransmission scheme to retain the flavor of the existing *TCP*. In *UDP*, rate-control prevents a user from sending at a 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, serves 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 conform 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

When the status of resources changes we need to maintain information about all these resources to be able to degrade the performance of an application gracefully. This has to be done on a global basis. 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 in order 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. 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.

6. Conclusions and Future work

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 a two-level application to network QoS translator, protocol tuning components, local feedback component, and class-based scheduling.

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(UDP)/IP/AAL5/ATM and direct AAL5/ATM with respect to their control parameters. We identified 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. 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. The proposed QoS system can refer to the tables presented in this paper to select appropriate protocol control parameters to provide an application specified throughput, loss and bounded delay requirements.

In this paper we report on the experimentation and analysis phase of QUANTA. Preliminary results demonstrate that QUANTA has the potential for a considerable improvement over the base-line case. For more details on the design aspects of QUANTA and these preliminary results refer to [12].

References

- [1] *STREAMS Programmer's Guide*. SunSoft Technical Manual, 1994.
- [2] ABR signalling FAQ. *ATM forum signalling group discussions*, June 1995.
- [3] F. Bonomi and K. W. Fendick. The Rate-Based Flow Control Framework for the Available Bit Rate ATM Service. *IEEE Network Magazine*, 9(2):25–29, March/April 1995.
- [4] D. Borman, R. Braden, and V. Jacobson. TCP Extensions for High Performance. *Request for comments 1323*, May 1992.
- [5] J. Y. L. Boudec. The Asynchronous Transfer Mode: a tutorial. *Computer Networks and ISDN Systems*, 24:279–309, 1992.
- [6] D. Braden and V. Jacobson. TCP extensions for long-delay paths. *Request for comments 1072*, October 1988.
- [7] D. Clark. Window and Acknowledgment Strategy in TCP.
- [8] D. Hong and T. Suda. Congestion Control and Prevention in ATM Networks. *IEEE Network Magazine*, pages 10–16, July 1991.
- [9] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review*, 18:314–329, August 1988.
- [10] H. T. Kung and R. Morris. Credit-Based Flow Control for ATM Networks. *IEEE Network Magazine*, 9(2):40–56, March/April 1995.
- [11] C. Papadopoulos and G. M. Parulkar. Experimental evaluation of SUNOS IPC and TCP/IP protocol implementation. *IEEE/ACM Transactions on Networks*, 1(2), April 1993.
- [12] D. Sudheer and K. Maly. Design of quanta and evaluation methodology. *Submitted to - IFIP Fifth International workshop on Protocols for High Speed Networks*, October 1996.
- [13] D. Sudheer, K. Maly, and C. M. Overstreet. Performance evaluation of TCP(UDP)/IP over ATM networks. Technical report, Department of Computer Science, Old Dominion University, # TR_94_23, September 1994.
- [14] D. Sudheer, K. Maly, C. M. Overstreet, and R. Mukkamala. Missing end-system components: A case-study. Technical report, Department of Computer Science, Old Dominion University, # TR_95_15, June 1995.

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	3.00	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

<i>direct</i> related					Common	<i>direct</i> related		<i>TCP</i> related	
<i>Target Offered load 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

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	80	0.48	1600/1583/1.06	1583/1583/0	1.06	49.95	130.99	0.50
1.5	80	1.44	1600/1594/0.38	1594/1594/0	0.38	17.75	43.74	1.50
10	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	1600/1600/0	1600/971/ 39.31	39.31	2.94	1.49	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