Experimental Study of Issues in End-to-End QoS

Klara Nahrstedt and Jonathan Smith*
Distributed System Laboratory
Computer Science Department
University of Pennsylvania
e-mail: klara@aurora.cis.upenn.edu

Abstract

Quality of Service (QoS) guarantees for 'delay sensitive' networked applications must be end-to-end. This paper presents an experimental study of this class of applications where the endpoints are computer workstations. The experiments show that operating system effects dominate any jitter in the network.

Our conclusion is that QoS provision by the workstation operating system is as important for maintaining end-to-end guarantees as is network QoS. In local-area settings, operating system influences may be more challenging for end-to-end QoS than network influences. The important influence variables are the degree of multiprocessing, the employed transport protocol (e.g. UDP or TCP), and the priorities assigned to processes.

1 Introduction

One of the important forms of information networks carry is "delay sensitive multimedia" traffic. The information is carried between *end-points* using delay-bounded communication protocols. Such protocols and their architecture have been of great interest in the computer networking community [3], [2], [6].

The other element in delay-sensitive networking is the behavior of the endpoints. This is the focus of our paper. The endpoints of future networks, except for the most trivial, will contain a processor, and should be flexible enough to offer some customization. This customization will take the form of "Quality of Service" (QoS) specifications for the applications, which must then be translated into appropriate sets of endpoint behaviors and network customizations.

1.1 Problem Description

Consider, for example, an application requiring video services. Such an application must make certain demands of the network, such as an allocated bandwidth, and perhaps some specification of the burst characteristics of the application traffic. These demands will change significantly in the case, e.g., where variable bit rate characteristics are induced on the video by coding schemes such as MPEG, which also serve to reduce the average bandwidth required. But the software infrastructure supporting the application, such as an operating system scheduler, must be integrated into the model of the endpoint in order to periodically schedule the application when data arrives or is to

*This work was supported by the National Science Foundation and the Advanced Research Projects Agency under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives. Additional support was provided by Bell Communications Research under Project DAWN, by an IBM Faculty Development Award, and by Hewlett-Packard.

be sent. *Control of this scheduling is crucial to providing QoS in an end-to-end fashion; it appears that some features of "real-time" operating systems (OS) must be available for application desiring use of the deterministic delay capabilities.*

Section 2 details our experiments and the results. Section 3 describes the implications of the experiments. Section 4 concludes the paper and offers ideas for future work.

1.2 Related Work

Related work to our research is done in two areas. One area is guaranteed services and predictability in the networks, and the other area looks at real-time operating systems and their predictability.

- **Real-Time Communication in Networks**

  Real-time communication in networks has been studied in the Tenet group at Berkeley. Ferrari et al. devised an establishment scheme for real-time channels [17] where processing and queuing delays in the networks are discussed in terms of deterministic and statistical bounds. The bounding of delays in an internetwork for real-time communication is an important assumption for guaranteed services [9].

  In order to establish real-time channels in the networks, we need predictions of the behavior of the shared medium, in this case the network. Therefore, analysis of different scheduling and queuing mechanisms at the network interfaces and switches, and the delay bounds, introduced by the scheduling (multiplexing) and queuing algorithms, are part of the knowledge which the services such as *admission service* need to predict what kind of guarantees can be made by the network [9], [13].

  Resource scheduling for guaranteed services in switches was studied by Lazar et al. at Columbia University [18], [7]. There is extensive research work done on computing delay bounds under different multiplexing and queuing algorithms such as Cruz's method for computing delay bounds under FCFS multiplexing [11], [12], or Parekh and Gallager's method for computing delay bounds under weighted fair queuing [8].

  Admission service, as one of the services, estimates the available shared resources and gives the permission or denies requests for resources, is studied at different levels in the networks. Admission control at the cell level is presented in [7], [10]. In [4] the call admission problem at the packet level is discussed. The end-to-end delay of the traditional voice traffic is analyzed using FCFS queuing in the network switch model.

- **Real-Time Support in Operating Systems**

  Real-time operating system research provides different mechanisms for real-time support to the application developers. Recent foci have included real-time thread models (e.g. Real-Time Mach) and scheduling mechanisms of the processor (e.g. Integrated Time-Driven Scheduler) [1], fine granularity of timers and clock interfaces (e.g. POSIX 1003.4, AIX 3.1, SVR4 UNIX). Further, there are operating systems like FORMULA, which provide user primitives for coordinating timing requests from the applications [19]. Scheduling primitives were implemented in FORMULA for the Computer Music Performance.

Unfortunately, there are too few links between these two areas and even fewer links among all three areas - real-time applications, networks, and OS. There are partial experimental results where OS performance and relation between application and OS are shown. In [5] it is shown that having real-time support in SVR4 UNIX is not sufficient to have good performance for local

video applications. For example, some priority-induced pathologies with respect to X windows were presented in [5] (e.g. video didn't get displayed if X server had higher priority than the video application.)

2 Experimental Analysis of AIX OS Real-Time Support for 'Delay-sensitive' Networked Applications

We performed our experiments using IBM RS/6000 workstations connected to an Ethernet. These workstations run the AIX operating system. In order to study 'delay sensitive networked applications, it is important to analyze these workstations as endpoints with respect to (1) different components in OS delay, and (1) relative contribution of OS delay versus network delay.

AIX Version 3.2 is a general purpose operating system which claims to provide some level of real-time support through use of priorities and priority classes [14]:

- The kernel allows real-time applications to be executed within a user program. Since the kernel is fully pre-emptible, the context switch time between user processes is shorter than in traditional UNIX systems (we measured over 60,000 getpid() calls/second on a model 320).
- AIX allows assignment of *fixed priority values* to the application developer with minimal interference from OS's own scheduling algorithm, i.e. the priority values are not altered by the scheduler.
- The user subroutines offer the same *timer resolution* as the kernel services.
- The real-time performance of some critical programs may be improved by adding facilities to the kernel, or by coding part of the applications as a *kernel extension*.
- AIX provides a locking mechanism for a task to lock its code and/or data into real memory, so that it can guarantee predictable response to an interrupt.

For our experiments, we used fixed priorities for the applications. There are 128 (0-127) priority levels for all processes. Generally, a real-time process will be assigned a fixed value between 0-40. This property allows us to split the priority range into two classes: *Real-Time* priorities (RT) and *Non-Real-Time* priorities (NRT). The tasks within NRT class obey the AIX scheduling algorithm, which is round-robin by priority, where priorities are adjusted to provide good interactive response.

We prototyped simple networked applications with video/robotics data, ran different application scenarios under different priorities and analyzed the delays at an endpoint.

2.1 Video Application

The first experiment involves a video application (Figure 1) where the *sender* is a video source with sample rate of 5 frames/s uncompressed, and sample size of 136 KBytes. The *receiver* is an X server displaying the video images.

2.1.1 Experimental Method

The video application is tested with three communication mechanisms: (1) IPC communication, which represents UNIX-STREAM socket communication; (2) TCP/IP communication (X11 protocol), which represents TCP/IP-STREAM communication; (3) UDP/IP communication, which

Figure 1: *Video Application*

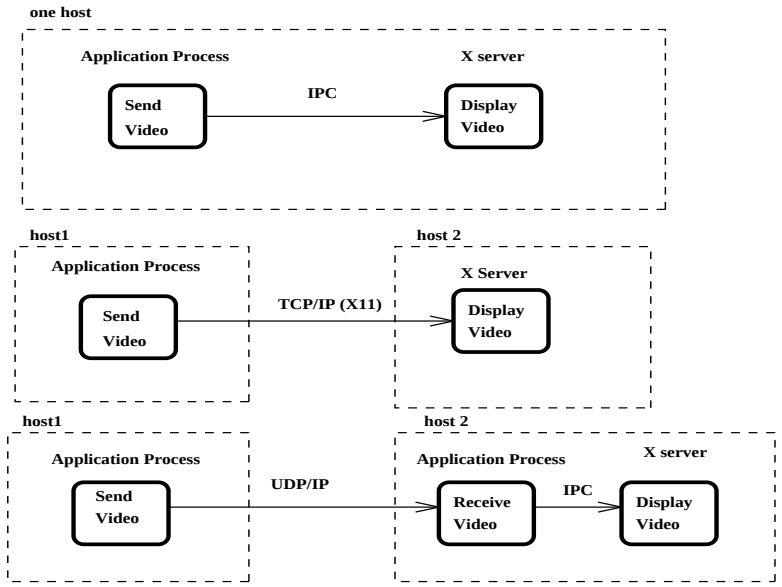

Figure 2: *Scenario 1*

represents UDP/IP datagram socket communication . All application tasks of the video application are implemented in one OS process.

The *measurement metric* is the delay between loading a video frame and displaying that frame through the X server. This delay shows if the receiver can display the video frames at the same rate the video frames are generated. The measured delay is equal to the the processing time of **XPutImage**[16] in case of IPC and TCP/IP communication between application client and X server, and in case of UDP/IP we measure the time between sending the first byte of the video frame at the sender side and display of the full video frame with **XPutImage** through IPC at the receiver side. The video application runs under different scenarios with respect to the number of applications/users.

1. **Scenario 1 - single user/single application**

   Scenario 1 (Figure 2) shows one user running at the sender/receiver and one application with one medium (video). All application tasks and X server are tested in NRT as well as in RT priority range.

2. **Scenario 2 - single user/multiple applications**

   The setup in our second set of tests was made by adding a second video application (video application 2 with IPC communication) at the receiver side to all cases in scenario 1. This represents the situation when one user can run concurrently two video applications. The

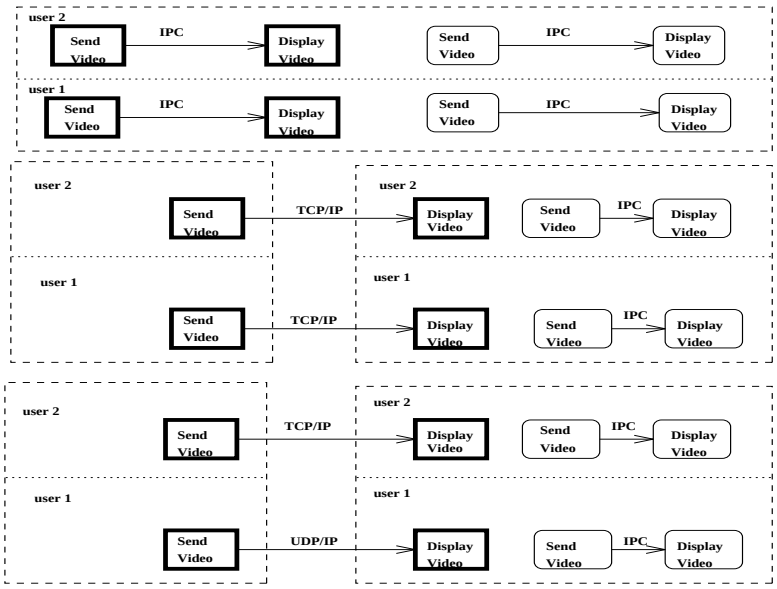

Figure 3: *Scenario 4*

video application 2 runs in NRT priority range and we measure the video application 1 and X server in NRT as well as in RT priority range.

3. **Scenario 3 - multiple users/single application**

   Scenario 3 adds to scenario 1 an additional user (user 2) at the receiver. The user 2 runs a local video application (with IPC communication). Hence, we have two users at the receiver sharing the CPU and X server to display the motion video. The user 2 runs the video application in NRT priority range. We measure user 1's video application performance in NRT as well as in RT priority range.

4. **Scenario 4 - multiple users/multiple applications**

   Scenario 4 (Figure 3) merges the scenario 1, 2 and 3 plus we add an additional user to the sender (for TCP/IP and UDP case). We measure the user 1's video application performance.

The results are presented using a box-plot[1] to illustrate the relative end-to-end delays and the variance in these delays for some interesting cases. The summary of all measurements are shown in Tables 1, 2, 3 and 4.

2.1.2 Results

We will discuss at first the results of each scenario and then across the different scenarios.

Results of Scenario 1 (Table 1)

[1] Boxplot provides much more information than an X-Y plot for this type of data. For a given x value, the box defines the middle 50 percent of the data, the horizontal line inside the box is the median, and the bar at the end of the dashed line marks the nearest value not beyond some standard range (in this case, 1.5*(inter-quartile range)) from the quartiles. Points outside these ranges are shown individually. Details of boxplot presentation can be found [20]. We use them here because of the ability to visualize jitter from the vertical characteristics of the box.

protocol	appl. prio	X prio	mean	var	var/mean [%]
IPC	NRT	NRT	126.9649	7.8489	6.181
TCP	NRT	NRT	721.556	531.2445	73.724
UDP	NRT	NRT	2642.73	10.1996	0.385
IPC	RT	NRT	126.4941	16.58657	13.11
TCP	RT	NRT	1059.732	850.9559	80.29
UDP	RT	NRT	2647.25	8.7601	0.33
IPC	RT	RT	112	5.8461	5.21
TCP	RT	RT	810.3862	408.3106	50.38
UDP	RT	RT	2688.972	35.15078	1.30
IPC	NRT	RT	121.963	2.3447	1.92
TCP	NRT	RT	592.3741	340.698	57.51
UDP	NRT	RT	2675.615	10.2280	0.38

Table 1: *Measurements in Scenario 1*

The Figure 4 shows the end-to-end delay and the variance in these delays for scenario 1 when the application process (sender/receiver) has a NRT (61) priority and X server has a NRT priority.

We observe three things. First, the TCP/IP and IPC communication using X11 protocol, as expected, are much lower in end-to-end delay (in IPC case the delays are [170-200] ms, in TCP/IP case the delays are [210-220] ms), although the TCP/IP variant doesn't provide the required sample rate to display the video at the sample rate of 5 frames/s.

Second, the UDP/IP case has a very high end-to-end delay (mean value 2800 ms) due to

- fragmentation of the video frame

  We used 4K fragments, hence the end-to-end delay of the video frame is 34*send operation + 34* network-end-to-end-delay + 34*receive operation + display with XPutImage (load of the video frame from the video source is even not considered).

- additional copying to the display buffer

  The received frame wasn't placed immediately into the video screen buffer, as it is done in X11 when TCP/IP is used. Therefore, the display operation includes an additional IPC communication.

- speed of UDP/IP protocol receiver

  The UDP/IP receiver can't keep up with the speed of the sender, so artificial delay of 75 ms between two consecutive fragments was introduced.

Third, the UDP/IP case exhibits more variation in the delay [2600-3000] ms. Especially, if we compare the variation at the sender and receiver (Figure 5), we see huge variance at the receiver caused not only by NRT priority of the application process (sender has also NRT priority and the variance is small), but mainly it is caused by the performance of the X server.

Hence, we tuned the parameter 'application process priority' (at the sender/receiver side) and set it to RT priority (priority = 1) as well as the 'X server process priority'.

The result of the application process priority assignment is that the mean value improves in case of IPC communication, but not in case of TCP or UDP. The reason is that in case of TCP and UDP the load of network plays a role. This means, even having RT priority of the application

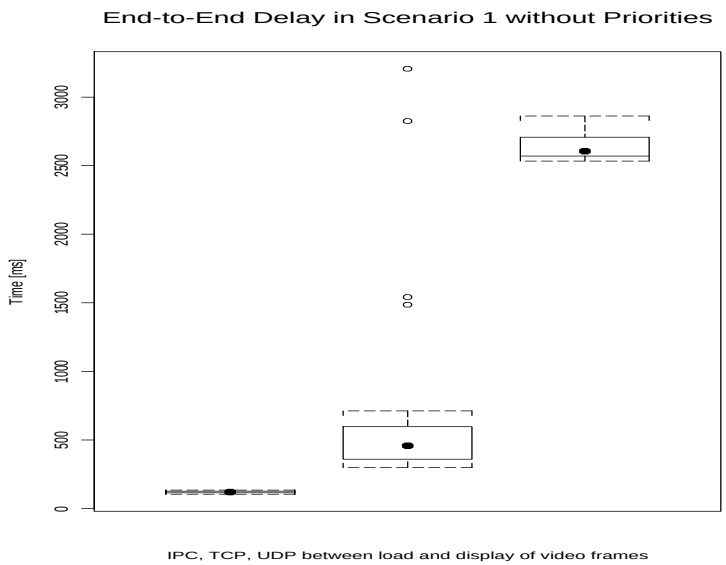

Figure 4: *Scenario 1: Comparison of the time between load and display of video frames among all three communication mechanisms (application priority=61, X server priority=61*

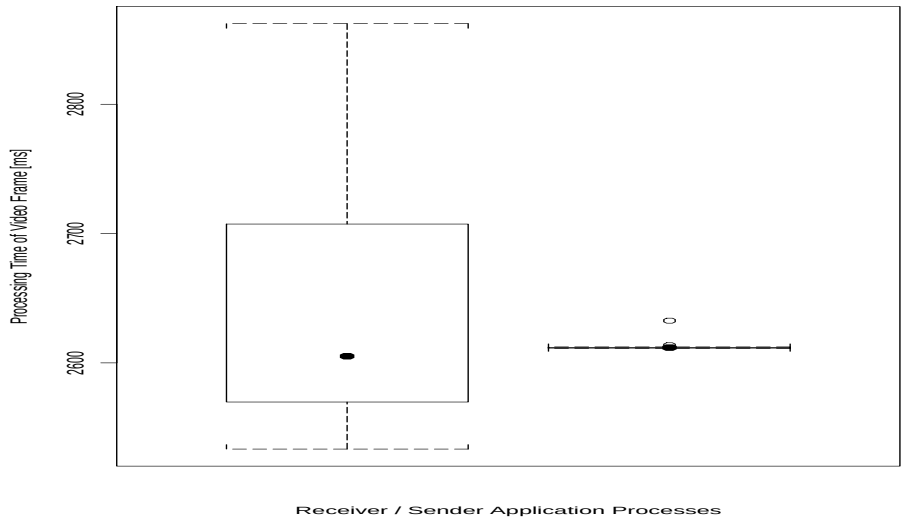

Figure 5: *Scenario 1: Comparison of the Processing Time for Video Frames at the Receiver and Sender side (UDP/IP communication between Receiver and Sender) - Sender/Receiver Application Processes have Priorities 61, X server has the Priority 61*

protocol	appl. prio	X prio	mean	var	(var/mean) [%]
IPC	NRT	NRT	532.9098	105.0293	19.70
TCP	NRT	NRT	1135.017	705.6264	62.16
UDP	NRT	NRT	2672.115	28.9675	1.084
IPC	RT	NRT	138.6292	1.190122	0.85
TCP	RT	NRT	834.9903	587.3643	70.34
UDP	RT	NRT	2659.473	11.0141	0.41
IPC	RT	RT	129.9682	0.0982	0.075
TCP	RT	RT	646.9401	212.352	32.82
UDP	RT	RT	2673.741	12.2379	0.45
IPC	NRT	RT	426.8241	37.3131	8.74
TCP	NRT	RT	790.3532	888.2305	112.38
UDP	NRT	RT	2660.821	31.6459	1.189

Table 2: *Measurements in Scenario 2, (1 user with 2 video applications)*

doesn't help to improve the end-to-end delay. The end-to-end jitter did improve in some cases. For example, UDP case improved the variance.

The result of tuning the 'X server priority' shows improvements in the mean value as well as in jitter in case of IPC, and TCP (The lowest jitter in IPC case was achieved under application process priority 1 and X server priority 3.). In case of UDP there is no improvement, on the contrary, the mean and jitter are higher. But over all the jitter in case of UDP is smaller than in TCP case, which is cased by the retransmission of the TCP protocol under loaded network. Thus, the X server priority doesn't produce stability either.

Results in Scenario 2 (Table 2)

The assignment of application process priority under X server running with NRT priority leads to improvement of mean value as well as of jitter. The X server RT priority, running application with RT priority, improves the mean value as well as the jitter in IPC case in comparison to any other IPC cases in scenario 2, the same is true for TCP, but not for UDP. UDP has the best mean value under the NRT priority for application process at the sender/receiver and X server having RT priority, although the jitter is the higher.

Results in Scenario 3 (Table 3)

The best performance for IPC communication in this scenario is when application process priority as well as X server priority are in RT range. For TCP case, the best performance is measured when application process is in RT priority range and X server in NRT range. The worst performance in case of TCP is when application process has NRT priority and X server the RT priority. UDP case performs the best when application as well as X server are in RT priority range. The worst performance in UDP case is when application runs in RT range and X server in NRT range.

Results in Scenario 4 (Table 4)

The best performance for IPC communication is when application and X server were running in RT range priorities. The worst performance for this type of communication is measured when

protocol	appl. prio	X prio	mean	var	(var/mean) [%]
IPC	NRT	NRT	299.4971	0.675	0.22
TCP	NRT	NRT	623.8552	227.4065	36.45
UDP	NRT	NRT	2693.444	12.37939	0.45
IPC	RT	NRT	496.66	29.2558	5.89
TCP	RT	NRT	560.4368	135.1564	24.11
UDP	RT	NRT	2772.11	18.9814	0.68
IPC	RT	RT	131.1341	0.0201	0.015
TCP	RT	RT	756.8429	255.733	33.78
UDP	RT	RT	2640.436	8.7178	0.33
IPC	NRT	RT	339.4741	19.8183	5.83
TCP	NRT	RT	943.8185	681.0515	72.15
UDP	NRT	RT	2660.606	9.09641	0.341

Table 3: *Measurements in Scenario 3 (2 users with each running 1 video application)*

protocol	appl. prio	X prio	mean	var	var/mean)[%]
IPC	NRT	NRT	594.1346	20.5474	3.45
TCP	NRT	NRT	1304.994	906.0536	69.42
UDP	NRT	NRT	3015.994	44.2390	1.46
IPC	RT	NRT	675.7457	18.4804	2.73
TCP	RT	NRT	1486.067	1358.044	91.38
UDP	RT	NRT	2880.658	22.7679	0.79
IPC	RT	RT	130.9498	0.05038	0.038
TCP	RT	RT	874.3641	801.3933	91.65
UDP	RT	RT	2870.951	32.98607	1.14
IPC	NRT	RT	608.7901	46.5663	7.648
TCP	NRT	RT	1198.43	1145.161	95.55
UDP	NRT	RT	2941.228	91.4583	3.109

Table 4: *Measurements Scenario 4*

application process has RT priority and X server a NRT priority. The same result is measured when TCP communication is invoked. UDP case achieves the best result when application has the RT priority and X server NRT priority. The worst mean and jitter are measured when both priorities are NRT.

Results across Scenarios

The best performance (mean=112ms, max jitter=5.8ms) in case of IPC is achieved in scenario 1 when both application and X server have RT priorities, although the jitter is higher than in the best case of scenarios 2, 3, 4. Scenario 1 is also the most predictable case. The worst performance (mean=675.7457ms, jitter=18.48ms) is in scenario 4 as expected.

The best performance in case of TCP is in scenario 3 when application process runs with RT priority and X server with NRT priority (mean=560.43ms and jitter 135.135 ms). It shows that the number of users at the workstation doesn't matter. The performance of TCP case depends on the load of the network. The load changes and with it the performance of the end-to-end delay and end-to-end jitter. The worst case again is in scenario 4 (mean=1486.067ms, and jitter=1358.044ms).

UDP case has the best performance in scenario 3 when both application and X server have RT priorities (mean=2640.436 ms and jitter=8.7178ms). The worst case is in scenario 4 when both application and X server had NRT priorities.

An interesting comparison is between scenario 2 and scenario 3. When application and X server are in NRT priority range, the scenario 3 performs better than scenario 2. The reason is that the OS scheduler allocates every user, each running one application, the same time slice and hence it performs better in CPU utilization than when the user runs two applications concurrently. The time in scenario 2 is not as well-balanced. Assigning priorities to the application/X server improves the performance of scenario 2 versus scenario 3 in some cases (e.g. TCP case with RT priorities for application and X server).

The most predictable case (as expected) is the single user/single application (scenario 1) although it doesn't perform the best in all cases. The problem, we experience in scenarios 2, 3, and 4, is that *adding a new application and/or user immediately results in unpredictable performance.* For example, if we have application and X server running with IPC in RT priority range, and we add another video application with NRT priority (scenario 2 or 3), the mean value (for the video display of the measured application with RT priority) jumps from 112 ms to 129.9682 ms (scenario 2) resp. 131.1341 ms (scenario 3). Therefore, the OS can't keep up the performance which we had before. This behavior makes the OS actions very unpredictable when adding other application and/or users to the scenario 1.

2.2 Video/Robotics Application

The second experiment involves a video/robotics application where the sender side includes in one application video source with 5 frames/second uncompressed, and sample size of 136 KBytes, and robotics source with sample rate 100 samples/second and sample size of 64 bytes. The receiver application displays the video and passes the robotics data to a robotics program. The application requirements for the robotics data are very strict, it means the sender has to send every 10 ms the robotics data (they are arriving from the robot hand).

This application is tested with UDP/IP communication between the sender and the receiver. The media are sent over two separate UDP/IP connections. The video frame is fragmented in 4KB fragments. The interest in this application is the implementation of two media at the sender/receiver endpoint as one process which sends/receives both media or as two processes where each process

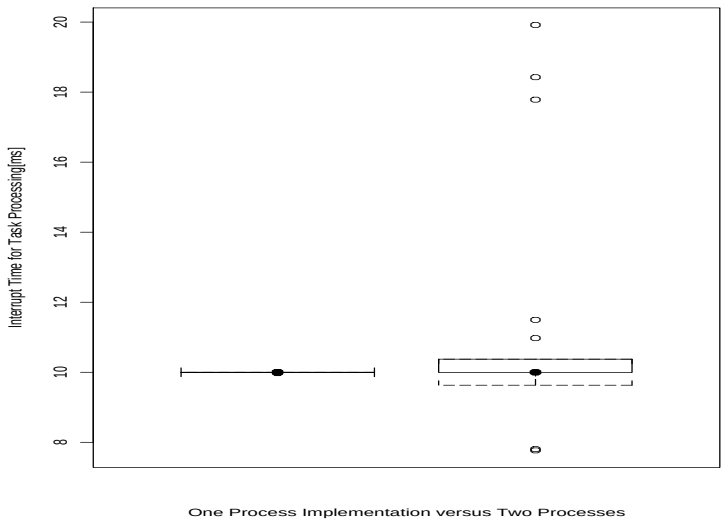

Figure 6: *One Process Versus Two Processes Implementation*

sends/receives individual medium.

The *measurement metric* is the timeout delay at which the processing of the robotics data has to be signaled and performed at the sender. We tested only the sender rate control because there is no other tool (e.g., X server) running which contributes to the interference.

The result, shown in Figure 6, is that implementing sending operation of both media in one process provides required determinism for rate control of robotics data. The implementation in two processes introduces jitter due to the context-switching and copying operation of the image fragment. It means although the robotics process is signaled, the video process finishes the copying operation and then context-switching occurs. This introduces additional jitter with respect to the processing of the robotics data.

3 Implications of the Results

There are several implications of the experiments:

- OS contributions can be larger than network contributions in some scenarios, such as lightly loaded LANs. Figure 7 shows the comparison of a lightly loaded LAN and the time to display a video frame with IPC which is the minimal bound for a networked application to display video. The OS is also lightly loaded, it means except the video application no other application/user are running (scenario 1).

- Priorities affect the variance, but must be coordinated for multiprocess applications. For example, the variance in scenario 1 for IPC case was the lowest when application process had NRT priority and X process had RT priority. In scenarios 2,3,4 the lowest variance was achieved when the measured application and X process had both RT priorities.

  In case, we had TCP or UDP communication protocols, the variance of UDP is generally lower than TCP although the mean performance of TCP case is 2-3 times better than UDP. The reason for UDP's improved variance is retransmission, used by TCP to provide reliability

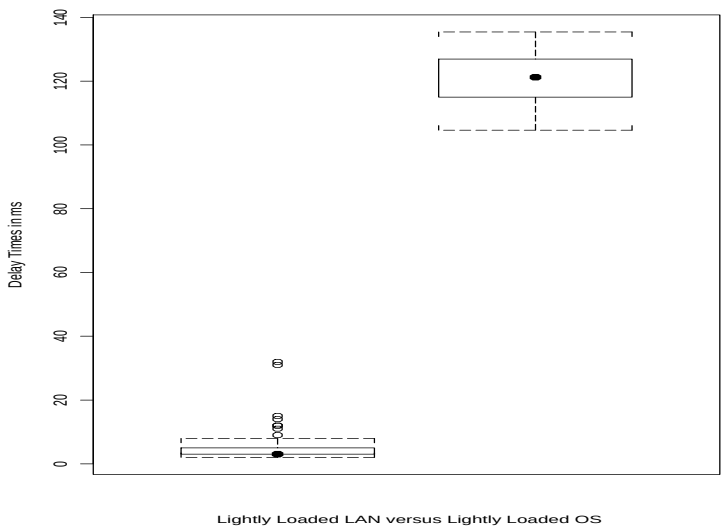

Figure 7: *Comparison between OS delays and network delays*

in case of loaded network. For the UDP case, the lowest jitter was in cases when application had RT priority and X server NRT priority (scenario 1,2,4).

- The architecture of the network protocols must be included in the orchestration of the end-points. The measurements for TCP and UDP cases support this statement. As we see in every scenario, not having the orchestration of the network protocols shows mean values of end-to-end delays and jitter which can't be improved very much even when application and/or X server had RT priorities.

- The results suggest that services providing guarantees must be able to:

  - get real-time communications,
  - control processor scheduling in some fashion,
  - match resource allocations to application choices such as a transport protocol.

- The transport protocols at the end-points for transmission of 'delay sensitive' data have to be simple with rate control mechanism and no retransmission. Hence, if data get lost, the application has to deal with the loss and not the transport protocol. This implies that if the transport protocol receiver can't keep up with sent data, it *drops* data and doesn't break, as the UDP experience showed us. (We had to introduce 75 ms delay between two consecutive fragments for flow control at the sender side because the receiver couldn't keep up with the speed and failed.) The application has to be informed what is the receiver's rate, and based on this information it has to deal with the losses.

- Further, the results suggest that in order to provide customized connections with guarantees, the connection set up protocol needs new services which (1) provide distribution of the application requirements to all other components of the communication system, (2) check the system resources of OS and network, which implies the service knowledge of the current resource status, (3) make a **'contract'** for resource guarantees with system components.

4 Conclusions and Directions for Future Research

We can conclude that workstations can (today) provide partial capability for *some* interesting QoS delivery. The challenges lie in a three directions. First, we should ensure that real-time capabilities are available in workstation operating systems. Second, we should work on models of interaction between the end-points and the network fabrics which are actually deployed, e.g., with microcontrollers on each switch port. Third, we should continue to explore the space of applications to test and refine QoS provision.

References

[1] H. Tokuda, T. Nakajima, P. Rao, "Real-Time Mach: Towards a Predictable Real-Time System" *Technical Report* School of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1993

[2] L.Zhang, B. Braden, D. Estrin, S. Herzog, S. Jamin, "Resource Reservation Protocol", *internet Draft*, October 1993

[3] C. J. Parris, D. Ferrari "A Dynamic Connection Management Scheme for Guaranteed Performance Services in Packet-Switching Integrated Services Networks", *Technical Report*, October 1993

[4] D. Yates, J. Kurose, D. Towsley, M.G. Hluchyj, "On per-session end-to-end delay distributions and the call admission problem for real-time applications with QoS requirements", *SIGCOMM'93 Proceedings*, September 1993, Ithaca, NY

[5] Jason Nieh, James G. Hanko, J. Duane Nortcutt, Gerard A. Wall "SVR4UNIX Scheduler Unacceptable for Multimedia Applications", *Proceedings from Workshop on OS for Digital Audio and Video 93*, November 1993, Lancaster, England

[6] J.M. Hyman, A.A. Lazar, G. Pacifici, "A Separation Principle between Scheduling and Admission Control for Broadband Switching", *IEEE Journal on SAC*, May 1993, Vol. 11, No.4, pp.605-616

[7] J.M. Hyman, A.A. Lazar, G. Pacifici, "A Separation Principle between Scheduling and Admission Control for Broadband Switching", *IEEE Journal on SAC*, May 1993, Vol.11 Nr. 4, pp.605-616

[8] A.K. Parekh, "A Generalized processor sharing approach to flow control in integrated services networks", PhD thesis, MIT Cambridge, MA, February 1992

[9] D. Ferrari, "RealTime Communication in an Internetwork", *Journal of High-Speed Networks*, Vol. 1, 1992, pp.79-103

[10] S. Keshav, "Report on Workshop on QoS Issues in High-Speed Networks", *Computer Communication Review*, October 1992, Vol.22, No.5, pp.74-85

[11] R.L. Cruz, "A calculus for network delay, part I: Network elements in isolation", *IEEE Transactions on Information Theory*, Vol.37., No. 1, January 1991, pp.114-131

[12] R.L. Cruz, "A calculus for network delay, part II: Network analysis", *IEEE Transactions on Information Theory*, Vol.37., No. 1, January 1991, pp.132-141

[13] K.W. Tindell, A. Burns, A.J. Wellings, "Guaranteeing Hard Real-Time End-to-End Communication Deadlines", *Technical Report Number RTRG/91/107*, Department of Computer Science, University of York, December 1991

[14] "AIX Version 3.1. RISC System/6000 as a Real-Time System", *Technical Documentation Number GG24-3633-0*, International Technical Support Center, Austin, Texas

[15] C. Topolcic, "Experimental Internet Stream Protocol, Version 2 (ST-II)", *Technical Report Number 1190 RFC-1190*, October 1990

[16] Andrian Nye, "Xlib Programming Manual for Version 11, Release 4", *O'Reilly & Associates, Inc.*, Volume One, 1990

[17] D. Ferrari, D.C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks", IEEE Journal on SAC, Vol.8, Nr. 3, April 1990, pp. 368-379

[18] A.A. Lazar, G.. Pacifici, J.S. White, "Real-Time Traffic Measurements on MAGNET-II", IEEE Journal on SAC, Vol.8, Nr. 3, April 1990, pp. 467-483

[19] David P. Anderson, Ron Kuivila "A System for Computer Music Performance", *acm Transactions on Computer Systems*, Vol.8, Nr.1, February 1990, pp.56-82

[20] Richard A. Becker, John M. Chambers, Allan R. Wilks, "The S Language", *published by Wadsworth & Brooks*, California, 1988