Experimental QoS Performances of Multimedia Applications*

Phil Yonghui Wang
pywang@nortelnetworks.com
Technology Centre
Nortel Networks

Yechiam Yemini, Danilo Florissi
{yemini, df }@cs.columbia.edu
Computer Science Dept
Columbia University

John Zinky
jzinky@bbn.com
BBN Technologies

Patricia Florissi
patricia@smarts.com
SMARTS

Abstract **−Several QoS provisioning mechanisms such as Differentiated Services (Diffserv) and Integrated Services (Intserv) have been recently devised and applied to bring Quality of Service (QoS) to the Internet. This paper studies end-end QoS performances of two QoS-demanding applications using different transport protocols. Both applications are tested in a real network environment, with end-end QoS provisioning by Intserv. They use QoSockets, a new extension of QoS specification and management to the Berkeley sockets. Their performances in terms of throughput, delay, jitter, and loss are measured under a number of test cases combining several factors: (1) single or multiple flows, with or without resource reservations; (2) normal, heavy, or overloaded scenarios; (3) uni- or bi-directional streams; and (4) TCP or UDP protocols. The experimental results show that the performances of two applications with the Intserv resource reservations are significantly improved, but not always guaranteed. It is also shown that UDP applications are able to get the requested QoS while TCP applications may not because of the nature of its bi-directional traffic flow. The paper provides detailed interpretation of the results and provides generic conclusions on application QoS.**

I.   INTRODUCTION

Two major mechanisms to support delivery of Quality of Services (QoS) have been proposed by the IETF: Differentiated Services (Diffserv) [7] and Integrated Services (Intserv) [3]. Diffserv is a packet-based priority service that provides several types of premium or assured services to meet differentiated needs of network applications. Intserv is a flow-based resource reservation service, which employs guaranteed and controlled load services to support end-end mission-critical services such as real-time service. Intserv uses RSVP (ReSerVation Protocol), the resource reservation signaling protocol [2].

This paper deals with the end-end QoS delivery from the perspective of an application. An application must not only reserve its required QoS, but also monitor and respond to the actual QoS delivered because some intermediate networks may not strictly guarantee the QoS requested.

QoSockets (Quality of Service Sockets) [1] is an extension of the Berkeley socket mechanisms to support provisioning and management of end-end QoS. A QoS-demanding application can use QoSockets to request end-end service with specified QoS guarantees. The QoS specifications are used to negotiate and allocate network resources, as possible -- in a manner that shelters the application from the underlying resource allocation mechanisms, and to generate real-time instrumentation to monitor the actual QoS delivered by the network. This QoS-management instrumentation enables QoS managers using SNMP (Simple Network Management Protocol) [13] to access the automatically generated QoS MIBs (Management Information Bases) [14]. In particular, an application using QoSockets can monitor the actual performance and adapt then to changing network conditions dynamically.

This paper describes experiments in applying QoSockets for QoS provisioning to two applications. The first is NetVideo [10], a UDP-based real-time video tool; and the other is DIRM [6], a TCP-based resource management middleware for socket-based and CORBA[4]-based applications. Both applications were originally developed using sockets and have been easily modified to use QoSockets and take advantage of its powerful infrastructure.

Each application is tested under three traffic conditions with varied flow demands and reservation scenarios. Each has its own testbed, consisting of two sub-networks with heterogeneous system environments bridged by a "bottleneck" link between two RSVP-aware routers. Intserv is used to provide end-end resource reservations in the network.

The experimental results show that these applications demanding QoS gain significant performance improvements through the use of QoSockets. NetVideo runs fairly steadily, but DIRM has a very complex behavior because its traffic is bi-directional and of large and varied-size messages, and its reservations do not cover the entire traffic route.

In addition to the guaranteed data flow, a typical TCP communication such as DIRM requires a guaranteed acknowledgement (ACK) flow. QoS provisioning services are usually unidirectional and present difficulties for the allocation of the reverse ACK traffic. In general, this fact makes QoS guarantees for TCP applications more challenging.

This paper is organized as follows. QoSockets is introduced in Section 2 with its architecture, QoS characterization model, QoS provisioning, and QoS management. Section 3 describes two multimedia applications, NetVideo and DIRM, with their QoS requirements and experimental environments (testbeds). The experimental data (throughput, delay, jitter, and loss) are detailed and discussed in Section 4. Finally, Section 5 presents conclusions on QoS- demanding applications and QoS provisioning services.

II.   QoSockets

Berkeley sockets are widely used in network programming, but by themselves do not bring QoS provisioning to applications. QoSockets [1] extends Berkeley sockets to enable applications to specify and manage QoS. QoSockets provides mechanisms to provision QoS by allocating network resources to applications, and by monitoring QoS delivery performance in real-time.
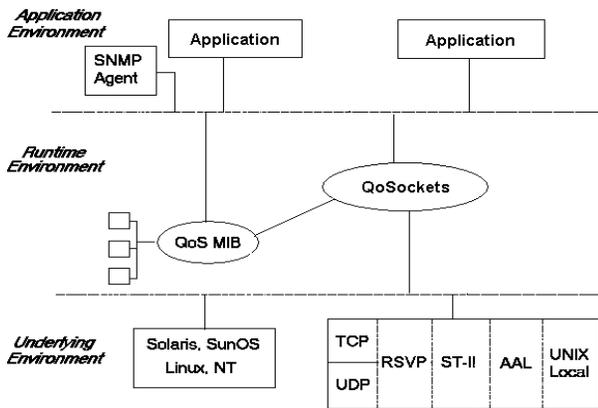

Figure 1: QoSockets Architecture

A.   Architecture & Operations

The overall architecture of QoSockets is depicted in Figure 1. An application provides specifications of its desired QoS. QoSockets compiles the specifications into respective transport protocols and mechanisms, when possible. Protocols supported by QoSockets include *TCP, UDP, RSVP, ST-II,* and *ATM* [5]. QoSockets also generates instrumentation to monitor the QoS delivered to the application and constructs appropriate QoS Management Information Bases (MIBs) to access this instrumentation.

QoSockets supports the following functions:

- *Connection Establishment*: initialize and establish connections and reserve the application QoS requirements specified.
- *Selection of Protocols*: select a specific transport protocol and bind a socket address to a QoSockets connection endpoint.

- *Monitoring QoS delivery*: monitor the QoS performance of applications communications, and store the sampled performance statistics into QoS MIBs.
- *QoS MIB Access*: access values of QoS MIBs using SNMP-based interfaces.

Figure 2 shows how QoSockets operates above an underlying RSVP service. QoSockets shelters applications from the complexity of the interface details of the specific QoS provisioning mechanism. One could use the same QoSockets specification for RSVP and ATM.
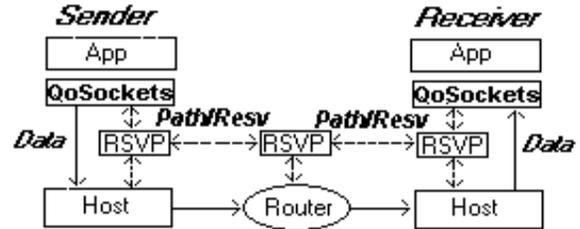

Figure 2: QoSockets and Intserv/RSVP

Figure 3 shows how QoSockets works with QoS MIBs. When an application establishes a QoSockets connection, QoSockets starts collecting the status and performance data related to the connection and its traffic, including QoS specifications, connection duration, transmission rates, delays, etc. It also detects QoS violation by analyzing the QoS requirements and real collected performance statistics.
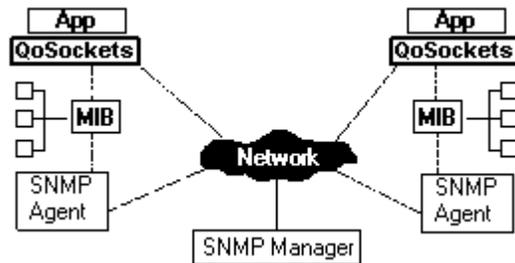

Figure 3: QoSockets and MIB

The data stored in the QoS MIBs are accessed inside the application or from SNMP agents using remote SNMP network managers. Thus, QoSockets allows applications to control and adapt to QoS performance by using application exception handling procedures (locally) or by requiring assistance from network managers (remotely).

B.   QoS Characterization

The main types of QoS attributes in QoSockets are *throughput*, *delay (and jitter)*, and *reliability.* In addition, QoSockets introduces the *coerced flag*, to coerce compatible QoS requirements of the sender and the receiver of a traffic stream.

1)   Throughput

2

QoSockets defines four parameters to represent the network throughput.

- *min_rate*: Lower bound of transmission rate;
- *max_rate*: Upper bound of transmission rate;
- *peak_rate*: Upper bound of transmission peak rate;
- *size*: Maximal size of transmitted messages.

Each rate is number of messages conveyed per second. The throughput is the product of the rate (*min_rate*, *max_rate*, or *peak_rate*) multiplied by the message *size* (bytes). For the *i*th traffic stream, its throughput is computed as (in bytes/s):

$$\text{Minimal:} \quad t^i_m = min\_rate^i \times size^i \qquad (1a)$$
$$\text{Maximal:} \quad t^i_M = max\_rate^i \times size^i \qquad (1b)$$
$$\text{Peak:} \quad t^i_p = peak\_rate^i \times size^I \qquad (1c)$$

2) Delay & Jitter

QoSockets defines four parameters related to the transmission delay and jitter.

- *min_delay*: Lower bound of transmission delay;
- *max_delay*: Upper bound of transmission delay;
- *int_delay*: Maximal time elapsing between two received messages;
- *jitter*: Maximal delay variance of two consecutive messages

These parameters are metered in milliseconds.

3) Reliability

QoSockets defines the reliability using three major parameters.

- *loss*: Percentage of messages lost;
- *rec_time*: Maximal time elapsed for recovering a disrupted connection;
- *permt*: Permutable flag indicating if messages can be delivered out of order.

QoSockets also provides other parameters (e.g., connection failures) used for monitoring network reliability.

4) Coerced flags

QoSockets allows both the sender and the receiver of a stream to define their own QoS parameters. Sometimes, the QoS parameters at each end conflict with each other and need to be coerced (downgraded) to a commonly accepted level. Coerced flags are therefore used to indicate which parameters should be coerced. If no coercion is requested, both sender and receiver use their own parameters to request QoS, which may cause resource allocation failure in case of incompatibility.

For example, suppose two ends of a traffic stream want to coerce their peak rates (by setting *coerce_peak_rate* = True), and the rates of the sender and the receiver are 64 and 60 KBps (kilobytes per second) respectively. QoSockets coerces them to the minimal common rate of 60 KBps, and notifies the new rate to both sender and receiver. The sender effectively downgrades its peak rate to 60 KBps as a result.

C. QoS Provisioning

QoSockets provides application QoS by requesting resource allocations of the underlying service providers such as Intserv, Diffserv, and ATM. The current implementation includes ATM and RSVP. *RSVP*, also known as "soft mode", is a reservation protocol of Intserv and available for TCP and UDP traffic (referred here as *R-TCP* and *R-UDP* respectively).

In the soft mode, QoSockets maps the application QoS requirements to the Intserv QoS, and requests the reservation to the RSVP daemons at the end hosts of senders and receivers. The daemons propagate the QoS request to the resources (hosts and routers) along the flow route. If a resource reservation succeeds, the application network communication associated with the reservation may meet its QoS demands. When a resource reservation fails, QoSockets returns a message to the application. Combining this message with the QoS MIB contents, an application can change its QoS requirement to adapt its reservation to available resources.

Experience with QoSockets shows that, even when the reservation succeeds, the end-end effective QoS may drift from the original negotiated QoS. There are several reasons for this. (1) Not every intermediate equipment involved support reservations. For example, it is common that a workstation requesting an RSVP reservation is in fact connected to a shared best-effort Ethernet hub and the hub connected to an RSVP router. (2) Not all applications comply with their reservations. The application may actually send more packets than the reservation it requested and incur possibly large packet delays and losses. (3) Equipment may fail. Applications have to see disruptions of QoS and need to choose alternative routes.

III. QoS APPLICATIONS AND TESTBEDS

This section introduces two multimedia applications: the real-time video tool NetVideo [10] and the resource management system DIRM [6]. Their core programs are respectively NV and IIOPGW (IIOP GateWay, the resource manager of DIRM), in which the socket application program interface (API) has been replaced for the QoSockets API.

To investigate issues facing provisioning protocols in providing QoS, each applications uses a different transport protocol. NetVideo uses UDP, and DIRM uses TCP. Moreover, the authors developed the traffic generation program TG (Traffic Generator) to generate the reference traffic of TCP or UDP for the tests.

The testbeds of NetVideo and DIRM are similar in network layout, but very different in how they are used.

A. *Similarities of the Two Testbeds*

The two testbeds are shown respectively in Figure 4 (NetVideo) and Figure 5 (DIRM). Each testbed is not isolated butrather constructed to be a part of the Columbia University Computer Science Department network.

1) *Network Layout*

Each testbed consists of two local sub-networks: 128.59.10.0/24 (Subnet 10) and 128.59.11.0/24 (Subnet 11). Two routers are connected using a serial line (using another sub-network 192.168.1.0/24) which has a "bottleneck" bandwidth (1.5M) between Subnets 10 and 11.

2) *Hosts and Routers*

Two hubs connect the hosts, and constituting two separate sub-networks. The two workstation hosts are a Sun SPARCstation 20 (*ws0*) and a Sun SPARCstation 5 (*ws1*), equipped with the Class Based Queuing (CBQ) patch to boost their Solaris 2.5.1 kernels with traffic control support. The Sun RSVP package *SolarisRSVP 0.5.0* [8] is also installed. The two PC hosts are used only in the DIRM testbed: *pc0*, which is an IBM Thinkpad 760 (Pentium 166MHz), and *pc1*, which is a DELL Dimension XPS R400 (Pentium II 400MHz). Both PCs are equipped with Linux 2.0.36 and the Linux port of the RSVP r4.2a3 package [9]. (Although these hosts are not the latest devices, they are fast enough to manage the traffic and to congest the routers connected by the low bandwidth serial line).

Two Cisco 2514 routers are equipped with Cisco IOS 11.2 and provide RSVP support by Weighted Fair Queuing (WFQ).

3) *Traffic Flows*

In each experiment, two kinds of tunable traffic flows are generated between the two subnets for comparisons. The *main traffic flow* is generated by the pair of NV or IIOPGW programs while the *reference traffic flow* is generated by the pair of TG programs.

The *main traffic flow* may be reserved (with QoS) or unreserved (without QoS) and use UDP or TCP transport, while the *reference traffic flow* is always unreserved (either UDP or TCP).

4) *Test Cases*

A typical test case of an experiment is composed of: (1) reserved and/or unreserved *main traffic flows*; (2) normal, heavy, or overloaded traffic condition; (3) single or multiple flows; and (4) bi-directional TCP or unidirectional UDP. (Table 1 in next section lists all the test combinations.)

The Control-Load (CL) service of Intserv is used to provide applications with QoS (resource reservation). Otherwise, applications tested without QoS provisioning use the Best-Effort (BE) service.

5) *QoS Performance Monitoring*

The NV and IIOPGW applications are monitored by the QoSockets instrumentation in real time. All of the performance parameters (including throughput, delay, jitter, and loss) are sampled at the receiver end of a flow, while the throughput is sampled at the sender end. Throughput and loss are computed from total numbers of messages sent and received, which are sampled and reset every 0.5 second. The delay and jitter are sampled per message transmitted.

B. *NetVideo Testbed*

NetVideo [10] is a multimedia tool for the Internet that captures, transfers, and receives real-time video pictures using *UDP*. The proposed version employs the QoSockets API and requests QoS for UDP transport (*R-UDP*).

The two NV programs (sender and receiver) run on two workstations: *ws0* and *ws1* (see fig 4). *Ws1* acts as a video sender, is equipped with a video camera, and captures real-time pictures at 30 frames per second, while *ws0* acts as a video receiver and displays those pictures received from *ws1* on the screen. The *main traffic flow* of NetVideo, is depicted in Figure 4 along the route marked as "*UDP Traffic*".

Because the NetVideo sender can use bandwidth up to 1024 kilobits per second (kbps), its transmitting rate can be
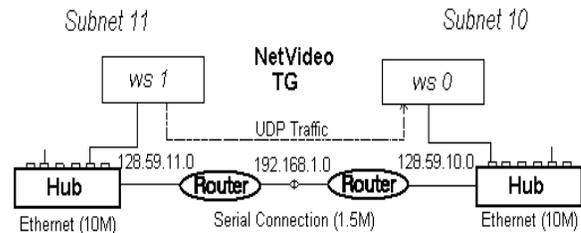
bigger than 30 frames/s (each frame is roughly 1280 bytes). In this test, it sends up to 80 frames/s when its bandwidth is set to 640 kbps.

Figure 4: NetVideo testbed

The two TGs run on the same hosts as the NV sender and receiver, and create a UDP *reference traffic flow* in the same direction as the *main traffic flow*. The TG sender sends a 1024-byte message at an approximate rate of 530kbps, but the receiving rate of the TG receiver may vary under different traffic condition.

1) *QoS Requirements*
User requirements
    Rates: 60~80 frames/s **Delay**: 0~100 ms
    Jitter: <50 ms   **Loss**: <5%
    Max frame length: 1280 Byte **Recovery time**: 5000 ms
Mapped QoSockets parameters
    Throughput
min_rate= 60 **max_rate**=80 **peak_rate**=100 **size**=1280

Delay
min_delay=0 max_delay=100 int_delay=50
 Reliability
rec_time=5000 loss=5 permt=False
 Coerced flags
 All coerced flags are set to TRUE.

2) Traffic Profiles

NV

Protocol: UDP	**Network Service**: CL or BE
Rate (kbps): 614.4	**Peak** (kbps): 1024
Message size (B): 1280	

TG

| **Protocol**: UDP | **Network Service**: BE |
| **Rate** (kbps): 540 | **Message size** (B): 1024 |

C. DIRM Testbed

DIRM [6] develops a high-level API that allows stream-based (socket) and object-based (CORBA [4]) applications to acquire QoS. Per application request, DIRM allocates and manages network resources (e.g., bandwidth) dynamically using the IIOPGW resource manager. IIOPGW uses the QoSockets over TCP (*R-TCP*) to request the resource allocation for its stream traffic.

Figure 5 is a typical scenario of DIRM, where Slideshow is a client-server sample Java application using CORBA. The Slideshow server, a CORBA object service implementation on *pc1*, manages a repository of images. The Slideshow client, a CORBA client application on *pc0*, requests the images through its ORB (Object Request Broker) and then displays them on the screen. Two IIOPGW programs on *ws0* and *ws1* run as IIOP gateways and establish a "bridge" between the ORBs of *pc0* and *pc1*. The bridge provides QoS guarantee to the traffic between Slideshow server and client.
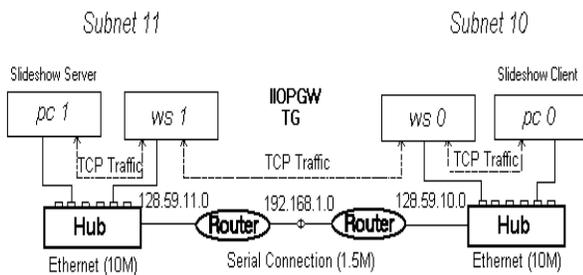

Figure 5: DIRM testbed

When starting, the *pc0* client makes an object request for the image service to its local ORB, which forwards the request to the *ws0* (local) IIOPGW at *ws0*. *Ws0* processes and transfers it to the *ws1* (remote) IIOPGW. *Ws1* processes and transfers it to the *pc1* server. *Pc1* processes this request and then requested image to *pc0* along the reserve path of the client request.

The IIOPGWs at *ws0* and *ws1* make two bandwidth reservations for the IIOP connections between them. The *main traffic flow* between two IIOPGWs is thus protected in the center of the path transferring images from *pc1* (the data sender) to *pc0* (the data receiver). The whole path is labeled with "*TCP Traffic*" in Fig. 5 and passes two-way traffic.

Two TG programs also run on *ws1* and *ws0*, and create a competing TCP stream along the same route as the *main traffic flow*. This is the *reference traffic flow*, without reservation.

One thing to be mentioned here is that IIOPGW transmits a whole image each time, from 18 to 58 kilobytes (KB), and resulting in big bursting rates for the *main traffic flow*. Its message size is consequently much bigger than that of NV (1280 bytes) and TG (1024 bytes).

1) QoS Requirements

User requirements

Slides: 1~3 images/s	**Delay**: 100~500 ms
Jitter: <250 ms	**Loss**: 0
Max message size: 60000 bytes	**Recovery time**: 5000 ms

Mapped QoS parameters
 Throughput
min_rate= 1 max_rate=1 peak_rate=3 size= 60KB
 Delay
min_delay=100 max_delay=500 int_delay=250
 Reliability
rec_time=5000 loss=0 permt=False
 Coerced flags
 All coerced flags are set to TRUE.

2) Traffic Profiles

IIOPGW

Protocol: TCP	**Network Service**: CL or BE
Rate (kbps): 480	**Peak** (kbps): 1440
Message size (B): 60000	

TG

| **Protocol**: TCP | **Network Service**: BE |
| **Rate** (kbps): 540 | **Message size** (B): 1024 |

IV. RESULTS AND ANALYSIS

The test performance is monitored in real-time inside applications, using the QoSockets MIB management of NV and IIOPGW and the TG monitoring module. The main parameters studied are *throughput*, *delay, jitter*, and *loss*. *Throughput* and *loss* are computed over time (t), and *delay* and *jitter* are computed per message (m). These measures are defined in Equations (2)-(5).

- *Throughput*

$$T_i(t) = \sum P_i(t) / t_i \qquad (2)$$

Where T_i is the throughput (bits/s or bps) during the *i*th sampling interval, $\sum P_i(t)$ is the total bits of all received messages within the *i*th interval, and t_i is the time duration of the *i*th interval.

- *Loss*

$$L_i(t) = 100 * (1 - \sum R_i(t) / \sum S_i(t)) \qquad (3)$$

Where L_i is the loss rate (%) during the ith interval, and $\sum R_i(t)$ and $\sum S_i(t)$ are respectively the total numbers of received and sent messages within the ith interval.

- *Delay*

$$D_i(m) = r_i(m)- s_i(m) \qquad (4)$$

Where D_i is the delay (millisecond) of the ith message arrived, and r_i and s_i are the arrival and sending timestamps of the ith message.

- *Jitter*

$$J_i(m) = |D_i(m) – D_{i-1}(m)|, \text{ while } i>0 \qquad (5)$$

Where J_i is the jitter (absolute value in millisecond) of the ith message, D_i and D_{i-1} are the delays of two consecutive messages computed from Equation (4).

Each testbed executes three experiments with different traffic conditions, and is also subject to the background traffic within the departmental network. (A) *Normal*, involving a single flow of NV, IIOPGW or TG, with total traffic close to 50% of the bottleneck bandwidth (1.5Mbps). (B) *Heavy,* involving two flows: one NV or IIOPGW and one TG, with total traffic close to 80% of the bottleneck. (C) *Overloaded*, involving three flows: one NV or IIOPGW and two TG, with total traffic beyond the bottleneck.

Test	*NetVideo (NV)*	*DIRM(IIOPGW)*
A. One-flow: Normal		
A1	**NV w/o QoS: UDP**	**IIOPGW w/o QoS: TCP**
A2	NV w/ QoS: R-UDP	IIOPGW w/ QoS: R-TCP
A3	TG: UDP	TG: TCP
B. Two-flow: Heavy		
B1	**NV w/o QoS and TG**	**IIOPGW w/o QoS and TG**
B1a	NV w/o QoS: UDP	IIOPGW w/o QoS: TCP
B1b	TG: UDP	TG: TCP
B2	**NV w/ QoS and TG**	**IIOPGW w/ QoS and TG**
B2a	NV w/ QoS: R-UDP	IIOPGW w/ QoS:R-TCP
B2b	TG: UDP	TG: TCP
C. Three-flow: Overloaded		
C1	**NV w/o QoS and 2 TG**	**IIOPGW w/o QoS and 2 TG**
C1a	NV w/o QoS: UDP	IIOPGW w/o QoS: TCP
C1b	TG 1: UDP	TG 1: TCP
C1c	TG 2: UDP	TG 2: TCP
C2	**NV w/ QoS and 2 TG**	**IIOPGW w/ QoS and 2 TG**
C2a	NV w/ QoS: R-UDP	IIOPGW w/ QoS:R-TCP
C2b	TG 1: UDP	TG 1: TCP
C2c	TG 2: UDP	TG 2: TCP

Table 1 Test cases of NetVideo and IIOPGW experiments

For each traffic condition, each experiment performs 2–3 tests, as listed in Table 1. For example, two tests, *B1* and *B2*, study *heavy* traffic condition. Both *B1* and *B2* have two flows (e.g., *B1a* and *B1b*). For NetVideo, *B1a* is the unreserved main UDP flow generated by NV (without QoS), *B2a* is the reserved one (with QoS), while both *B1b* and *B2b* are the reference UDP flows generated by TG (without QoS).

The measurements of the two experiments are presented in this section, followed by analysis and discussions. The figures in this section depict the average experimental data sampled by NV, IIOPGW and TG. In the throughput figures (Figures 6 and 9), the light gray column represents an average value of the flow sender whilst the dark gray column represents the one of the flow receiver. Two columns drawn together, one light and the other dark, reflect the throughput rate difference of a flow in one test. In other figures (of loss, delay and jitter), only dark columns are drawn (from the measurements at the receivers).

A. NetVideo

In this experiment (for NV and TG) at least 100 samples of throughput and loss are computed, while 2000–4000 samples of delay and jitter (per message) are computed (varying for each test).

1) Throughput

Figure 6 shows the average throughput rates (sending and receiving) for all tested flows. Looking at these rate columns, the following characteristics about throughput are concluded.

- For reserved NV flows (A2, B2a and C2a), the sending and receiving rates match. For unreserved flows of both NV (A1, B1a and C1a) and TG, their rates do not match and do show considerable disparity particularly under overloaded traffic.
- For NV flows, the rates of reserved flows (A2 and B2a) under normal and heavy traffic conditions are a bit less than those of unreserved flows (A1 and B1a), due to a tiny overhead by the Solaris traffic-control kernel scheduling reserved flows. As expected, under overloaded traffic condition, the receiving rate of reserved flow (C2a, 520kbps) is twice that of the unreserved one (C1a, 250kbps).
- As the traffic condition varies from normal (A), heavy (B) and overloaded (C), reserved NV flows have steady throughput rates close to 530kbps, whereas unreserved NV and TG flows reduce their receiving rates from 570 (A1) to 250 kbps (C1a).
- Under the overloaded traffic condition, the reserved C2a flow has similar sending and receiving rates (520kbps), while the unreserved C1a and TG flows (C1b and C1c, C2b and C2c) do experience significant disparity between sending and receiving rates. Moreover, TG flows C2b and C2c become worse and even experience -200kbps disparity when compared to flows C1b and C1c, which experience only -120kbps disparity.
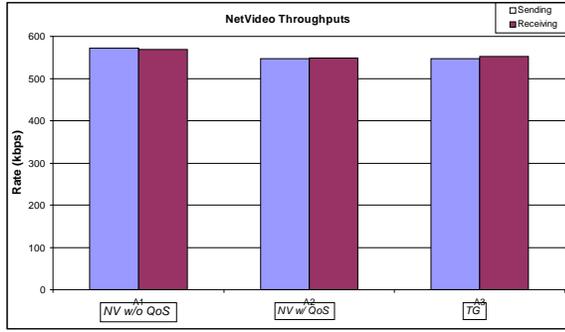
(A) Single flow tests (normal), each column group (gray and dark) represents the sending and receiving rates of one tested flow.
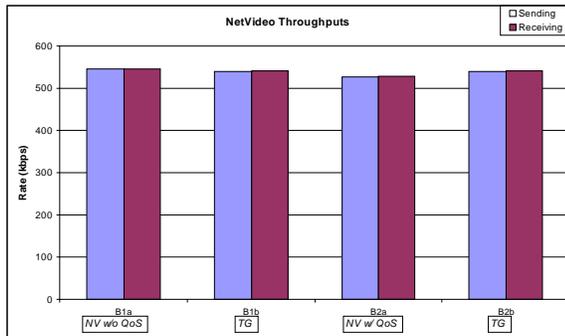
(B) Two-flow tests (heavy), the left 2 column groups represent one NV and one TG flow rates when NV is tested without QoS whilst the right 2 groups represent their rates when NV is with QoS.
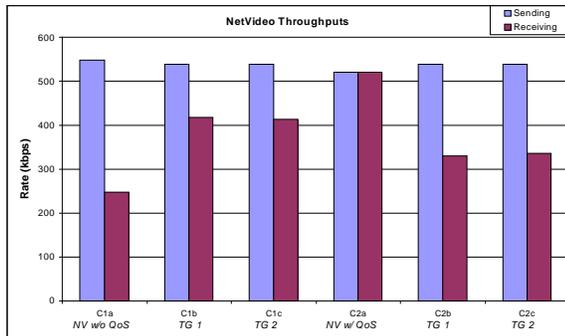
(C) Three-flow tests (overloaded), the left 3 column groups represent one NV and two TG flow rates when NV is tested without QoS whilst the right 3 groups represent their rates when NV is with QoS

Figure 6: Throughput rates of NetVideo flows

2) Loss

Message loss is very dependent on the throughput, and increases as the gap between sending and receiving rates of a flow increases. Figure 7 shows the average loss rates for all tested flows, as sampled at the receiving ends.

- Under normal and heavy traffic conditions, both reserved and unreserved flows (except A1) do not lose messages.
- Under the overloaded traffic condition, the reserved C2a flow has zero loss, while the unreserved C1a gets a big loss rate (47%) and TG flows have loss rates 25% (C1b and C1c) and 40% (C2b and C2c).
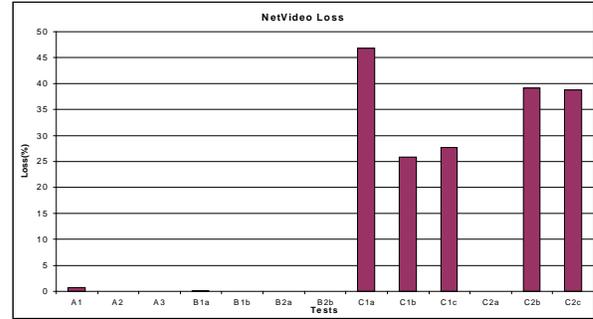

Figure 7: Message loss rates of NetVideo

3) Delay

Figure 8 shows the average delay values for all flows, as sampled per message at the receiving ends. From this figure, one concludes the following.
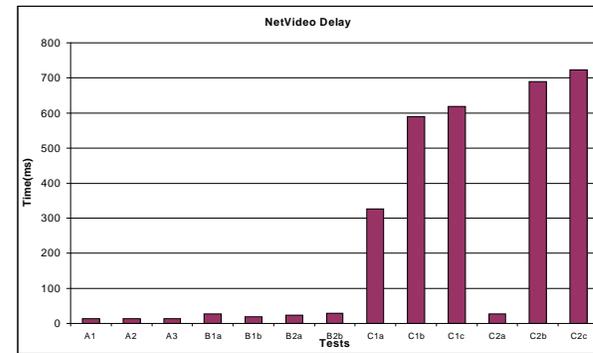

Figure 8: Message delays of NetVideo

- As the traffic condition varies from normal (A) to heavy (B) and overloaded (C), reserved NV flows have steady delays (<30ms), whereas unreserved NV and TG flows increase sharply their delays.
- Under the overloaded traffic condition, the reserved C2a has still a low delay (25ms), but the unreserved C1a and TG (C1b and C1c, C2b and C2c) flows have delays 10–30 times higher. TG flows C2b and C2c have delays up to 720 ms, larger than flows C1b and C1c do (600ms).

4) Jitter

Figure 9 shows the average jitter values for all tested flows, as computed from message delays at the receiving ends. Similar to delay, one concludes the following about jitter.

- As the traffic condition varies from normal (A) to heavy (B) and overloaded (C), reserved NV flows have bound jitter (<10ms), whereas unreserved NV and TG flows increase largely their jitter (e.g., C1a and C1c).
- Under the overloaded traffic condition, similar to the delay, the reserved C2a has jitter smaller than the unreserved C1a has. Moreover, different from what happens for the delay, TG flows (C2b, C2c) have also

lower jitter for the reserved NV (C2a), when compared to the flows C1b and C1c. The reserved flow (C2a) jitter (5ms) is much smaller than that of the unreserved C1a (115ms), indicating that traffic control for the main traffic may help in jitter reduction even for the reference traffic.
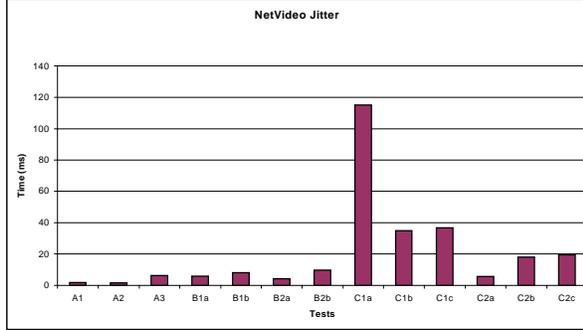

Figure 9: Message jitters of NetVideo

B. DIRM

DIRM tests are similar to NetVideo. At least 200 samples of throughput and loss are computed for IIOPGW and TG while 1000–10000 samples are computed for delay and jitter. It is noted that no message is lost in all tests because all flows here are TCP-based.

1) Throughput

The average throughput rates of DIRM tested flows are shown in Figure 10, and each flow has similar sending and receiving rates due to TCP control.

- For the reserved IIOPGW flows (A2, B2a and C2a), the sending and receiving rates match. For unreserved IIOPGW (A1, B1a and C1a) and TG flows, their rates do not match completely without QoS provisioning.
- Under normal traffic condition, there is no obvious difference of throughput rate between the reserved (A2) and unreserved (A1) IIOPGW flows. But, under heavy and overloaded traffic conditions, the reserved B2a and C2a flows have rates higher (20%) than those of the unreserved B1a and C1a.
- As the traffic condition varies from normal (A) to heavy (B) and overloaded (C), all of reserved IIOPGW, unreserved IIOPGW and TG flows reduce somewhat their throughput rates.

Here one notes that the throughput decreases as the traffic condition varies from normal to heavy and overloaded. It is natural that, because of no reservation, TG flows reduce their throughputs as the network traffic increases.

But, why do the reserved IIOPGW flows (B2a and C2a) have their throughput reduced as well? The reason is a bit complicated, and deferred until the next section "Discussion".
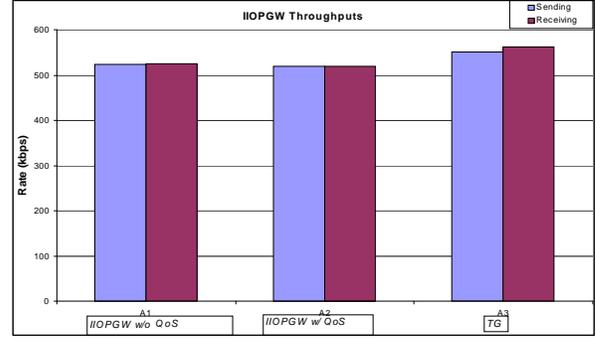
(A) Single flow tests (normal), each column group (gray and dark) represents the sending and receiving rates of one tested flow.
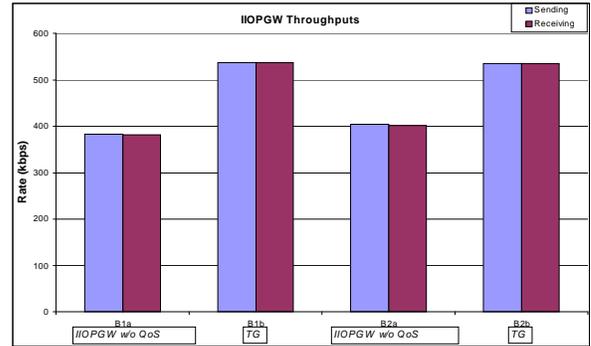
(B) Two-flow tests (heavy), the left 2 column groups represent one IIOPGW and one TG flow rates when IIOPGW is without QoS whilst the right 2 groups represent their rates when IIOPGW is with QoS.
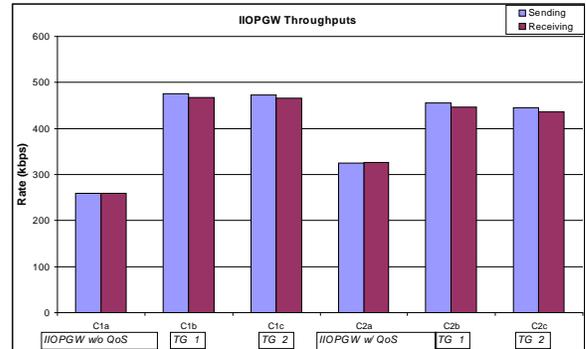
(C) Three-flow tests (overloaded), the left 3 column groups represent one IIOPGW and two TG flow rates when IIOPGW is without QoS whilst the right 3 groups represent their rates when IIOPGW is with QoS.

Figure 10: Throughput rates of DIRM flows

2) Delay

Figure 11 shows the average delay values sampled from all tested flows in the DIRM experiment.

- As the traffic condition varies from normal (A) to heavy (B) and overloaded (C), reserved IIOPGW, unreserved IIOPGW, and TG flows increase their delays.
- Under heavy and overloaded traffic conditions, the reserved IIOPGW (C2a) flow has smaller delay than the unreserved IIOPGW (C1a). The TG flows have similar delays (B1b vs. B2b, C1b vs. C2b and C1c vs. C2c).
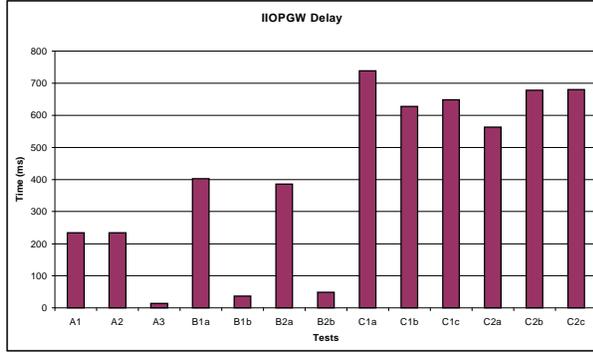
Figure 11: Message delays of DIRM

It is reasonable that both unreserved IIOPGW and TG increase their delays, as the traffic condition becomes heavy or overloaded. Why do the reserved IIOPGW flows (B2a and C2a) have big delays? The reason is that the average size of IIOPGW messages is 38 KB, much bigger than the TG message size of 1 KB. One can verify this statement by noting that, under normal condition, the delays of both unreserved (A1) and reserved (A2) IIOPGW flows are far bigger than that of the TG flow (A3). With a reservation the C2a flow (IIOPGW) has a smaller delay than TG flows C2b and C2c as expected. Also as expected, TG flows increase sharply their delays from B2b to C2b and C2c.
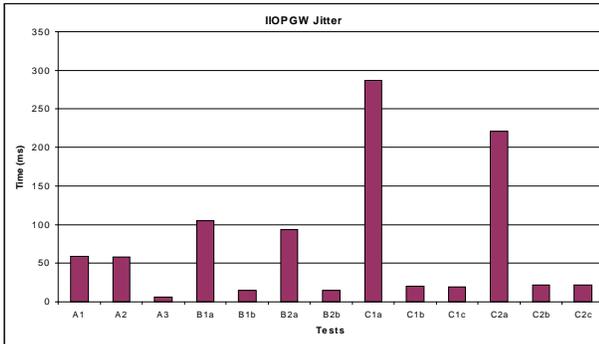

Figure 12: Message jitters of DIRM

3) Jitter

Figure 12 shows the average jitter values for all tested flows.

- As the traffic condition varies from normal (A), heavy (B) to overloaded (C), reserved IIOPGW, unreserved IIOPGW, and TG flows increase their jitter.
- Under heavy and overloaded traffic conditions, the reserved C2a has smaller jitter than the unreserved C1a. But, unreserved TG flows has similar and small jitter.

Both reserved and unreserved IIOPGW flows have bigger jitter than TG flows. While TG flow has a fixed message size (1 KB), an IIOPGW flow message size is not only bigger (38 KB at average) but also varies from 18 to 58 KB. As a consequence, the transmitting time of an IIOPGW message is variable and longer than TG, resulting in a larger jitter. The jitter under normal load shows this fact because both unreserved (A1) and reserved (A2) IIOPGW flows have much bigger jitter than TG (A3).

C. Discussions

1) NetVideo

The NetVideo experiments show that the reserved NV flows have obtained their requested QoS. Even when the traffic condition shifts from normal to overloaded, the reserved flows behave steady throughput, with low delay and jitter, and without message loss. In contrast, the receiver of the unreserved NV flow C1a receives only 50% of the sending rate, resulting in 47% message loss.

2) DIRM

The DIRM experiments show a different set of results. The Intserv reservations improve but do not guarantee the IIOPGW performances. All traffic flows (reserved or unreserved IIOPGW and TG) do not experience any message loss, but their TCP segments may be internally dropped (and re-transmitted). The drops are used to adjust the congestion window to reduce the flow throughput as the traffic load increases.

It is very important to notice, however, that a reserved DIRM/IIOPGW flow experiences higher throughput and lower delay and jitter than an unreserved flow, as observed previously.

DIRM has several aspects that contribute to its worse QoS performance. (1) DIRM generates bi-directional TCP traffic whereas NetVideo has only unidirectional UDP. (2) DIRM/IIOPGW transmits large and variable-size messages (from 18 to 58 KB) (may cause big burst rate and heavy IP packet fragmentation), whereas NetVideo transmits similar-size messages (roughly 1280 bytes). (3) DIRM reservations cover only the *main traffic flow* portion and not the entire traffic route, while NetVideo reservations are all end-end.

Finally, DIRM is a bit more complex than NetVideo. DIRM integrates a group of programs running on different platforms: IIOPGW and TG (C/C++) programs on Solaris, and Slideshow (Java) programs on Linux, while NetVideo has NV and TG (C/C++) on Solaris.

3) TCP, UDP and QoS Provisioning

There are important reasons why TCP and UDP protocols affect the QoS of their flows differently. UDP creates a unidirectional data flow, while TCP creates a bi-directional flow, one direction for data (originated from the sender) and the other for ACKs (originated from the receiver). In fact, the TCP slow-start and congestion avoidance mechanism [15-17] at the sender end monitors ACK packets for traffic congestion control, and use this information to decide the data transmission rate.

Current QoS provisioning services, such as Diffserv and Intserv/RSVP, protect unidirectional streams. That is the

main reason why UDP applications like NetVideo get better QoS. For a TCP application, this one-way resource provisioning guarantees only the data packets, while the ACKs are not guaranteed and thus may be delayed or even lost.

Once ACKs do not arrive in time, the sender slows or stops transmitting data packets, and even restarts the slow-start mechanism if delays are larger than the timeout. The net result is the reduction of the TCP throughput, as observed in the IIOPGW flow.

V. CONCLUSIONS

This paper describes two sets of experiments in which two different applications have been extended to support QoS using QoSockets and tested in a real network environment. Their performances provide us an insight of current Internet QoS behavior and challenges.

- Both UDP and TCP applications benefit from QoS (e.g., resource reservations) and experience significant improvement in their performances. Non-QoS flows may get better performance during light traffic, because there are no traffic control overheads, but suffer much worse behavior under heavy or overloaded traffic.
- QoSockets is able to map the generic QoS requirements of applications, very effectively, onto specific QoS provisioning mechanisms in a manner transparent to the applications. In addition, QoSockets generates QoS-monitoring instrumentation of real-time network performances, which is very valuable for QoS assurance, adaptation and management.
- TCP applications demanding QoS need more attention of both end users and QoS provisioning mechanisms, because these mechanisms do not guarantee bi-directional traffic flows. The ACK stream needs guarantee as the data stream does, otherwise, in case of ACK delay or loss, the application QoS degrades.
- The DIRM experiment shows that the QoS of an application is dependent not only on a particular service but also on its own architecture. If the application creates bi-directional traffic flows, transmits big size messages, or includes complex software and hardware components, the interactions with the QoS provisioning mechanics have to be carefully designed. Otherwise, they may impact the overall QoS performance.

Intserv and Diffserv are presently developed as central QoS provisioning services in current networks. In order for these mechanisms to become available for network applications, it is necessary to create appropriate middleware that can bridge the needs of applications with network QoS services. QoSockets provides this function by keeping processing overheads to a minimum (under 1%) and enabling simple incorporation of access to QoS delivery within applications, through minimal extensions of common socket API.

REFERENCES

[1] Florissi, P., "QuAL: Quality Assurance Language", Ph.D. Thesis, Columbia University, 1996
[2] Zhang, L., Berson, S., Herzog, S. and Jamin, S., "Resource ReSerVation Protocol (RSVP) – Version 1 Function Specification", Internet RFC-2205, 1997
[3] Braden, R., Clark, D. and Shenker, S., "Integrated Services in the Internet Architecture: Overview", Internet RFC 1633, June 1994
[4] Object Management Group, "The Common Object Request Broker: Architecture and Specification", Rev. 2.2, Feb. 1998
[5] ATM Forum, "ATM User-Network Interface Specification", Version 3.1, 1994
[6] Zinky, J., Bakken, D. and Schantz R., "Architectural Support for Quality of Service for CORBA Objects", Theory and Practice of Object Systems, January 1997.
[7] Blake, S., Black D., Carlson, M. Davies, E., Wang, Z. and Weiss, W., "An Architecture for Differentiated Services", Internet RFC-2475, Dec. 1998
[8] Sun, Solaris RSVP/CBQ, ftp://playground.sun.com/pub/rsvp/SolarisRSVP.0.5.0.tar.Z, Mar. 1998
[9] Wang, P.Y., Linux Port Of RSVP R4.2a3, http://www.cs.columbia.edu/~yhwang/ftp/qos/rsvp, Aug. 1998
[10] Xerox Corporation, NetVideo, Version 3.3, 1994
[11] Demers,A., Keshav, S. and Shenker, S., "Analysis and simulation of a fair queuing algorithm", Proc. Of ACM SIGCOMM, Austin, Texas, September 1989
[12] Floyd, S. and Jacobson, V., "Link-sharing and Resource Management Models for Packet Networks", Transaction on Networking, V.3, N.4, August 1995
[13] SNMPv2 Working Group, "Protocol Operations for Versions 2 of the Simple Network Management Protocol (SNMPv2)", Internet RFC-1905, January 1990
[14] SNMPv2 Working Group, "Management Information Base for Versions 2 of the Simple Network Management Protocol (SNMPv2)", Internet RFC-1907, January 1990
[15] Stevens, W., "TCP slow start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", Internet RFC 2001, January 1997
[16] Nagle, John, "Congestion Control in IP/TCP Internetworks", Internet RFC 896, Januray 1984
[17] Allman, M., Paxson, V. and Stevens, W., "TCP Congestion Control', Internet RFC 2581, April 1999