Experimental QoS Performances of Multimedia Applications

Phil Yonghui Wang, Yechiam Yemini, Danilo Florissi
{yhwang, yemini, df }@cs.columbia.edu
Computer Science Dept
Columbia University, New York, NY10027
Tel: (212) 939-7000 Fax: (212) 939-7181

Patricia Florissi
patricia@smarts.com
SMARTS
14 Mamaroneck Avenue
White Plains, NY10601

John Zinky
jzinky@bbn.com
BBN Technologies
10 Moulton Street
Cambridge, MA 0213

***Abstract* – To bring QoS to the Internet, several better-than-best-effort network services have been recently devised with significant efforts. The goal of this paper is to measure the effective performances of existing applications after incorporating QoS, and to understand what are the effects of provisioning QoS in the applications.**

Our experiments use two kinds of existing multimedia applications: UDP-based NetVideo and TCP-based DIRM. QoSockets is the Berkeley socket extension for the specification of QoS requirements and the management of QoS performances in a distributed environment. We replace their socket APIs with QoSockets and test them in real network environments that include programs written in C/C++ and Java on Solaris/SPARC and Linux/X86 platforms and two sub-networks connected by two routers. We also introduce an additional program TG to create UDP or TCP reference traffic for comparisons.

We use the integrated service (IS) to provide QoS for both applications, and examine their QoS performances (throughput, loss, delay and jitter) in a number of cases: (1) single and multiple, reserved and unreserved flows; (2) normal, heavy and overloaded traffic; (3) one- and two-way streams; and (4) TCP and UDP communications.

The experimental data shows that QoS performances of the two applications through resource reservations are significantly improved but slightly different for DIRM (which suffers the complexities of TCP and its own system). We summarize the experimental results and derive some generic conclusions on providing QoS for existing distributed applications.

I. INTRODUCTION

Today's Internet is not yet ready for Quality of Service (QoS) since the Best Effort (BE) network service is still widely used. BE employs FIFOQ (first-in-first-out queuing) which provides a single service level for all data packets, results in unpredictable packet delays and losses upon network congestion.

Targeting more service levels, Better-than-Best-Effort (BBE) network services employs multiple queues or classes scheduling such as WFQ (weighted fair queuing) and CBQ (class-based queuing) to enable traffic control, such as packet admission and classification. Differentiated Service (DS) [8] and Integrated Service (IS) [9] are two major BBE services proposed by the IETF, both offering multiple levels of service quality. DS is a packet-based priority service, and provides Premium and Assured services to meet differentiated requirements of network applications. IS is a flow-based reservation service, provides Controlled-Load and Guaranteed services to support mission-critical services such as real-time service. In addition, ATM [5] is a VC(virtual circuit)-based BBE service, and provides rate-based QoS levels. But, ATM is not built directly with the TCP/IP suite.

This paper takes the viewpoint that current Internet is a mesh of QoS-enabled (e.g., DS, IS and ATM) and non-QoS (BE) islands. Applications in this environment have to predict and adapt to changing assumptions on the underlying infrastructure. They need a QoS establishment to be able to (1) specify and request the needed QoS and (2) monitor and adapt to the actual QoS behavior offered by the network. QoSockets (Quality of Service Sockets) [1] is a runtime environment to meet the above challenges, extends Berkeley sockets with parameters for QoS specification which are then used to allocate network resources when possible, and automatically generates real-time instrumentation and monitoring for QoS management via SNMP.

Our work in this paper concentrates on QoS performances of multimedia application experiments and investigates various issues in QoS provision. We select two interactive applications, change their sockets APIs for QoSockets and then test them respectively in a real network. The first is NetVideo [10], a UDP-based real-time video tool; and the second is DIRM [6], a TCP-based resource management system for socket- and CORBA-based applications. These applications show that existing applications that use sockets can be easily upgraded to use QoSockets and take advantage of its powerful infrastructure. We also use a special program TG to create UDP or TCP reference traffic for comparison.

Their testbeds consist of two sub-networks between which we places a "bottleneck" link through two RSVP-capable routers, and include heterogeneous system environments and platforms. Integrated Service (IS) is used to provide QoS for both applications, and three traffic conditions are designed with different numbers of flows.

The experimental results show that both applications obtain significant improvements of their QoS performances. NetVideo is relatively steady, but DIRM is a bit different since it includes complex programs and platforms, generates two-way traffic with big and variable packet sizes, and does not provide reservations for all of its flows. Moreover, we also notice that a TCP application may not sustain the same level of QoS as UDP because BBE services offer one-way guarantees while TCP generates a reverse acknowledgement (ACK) flow in addition to a data flow.

This paper is organized as follows. QoSockets is introduced in section 2 with its structure, QoS characterization, provisioning and management. Section 3 describes two multimedia applications: NetVideo and DIRM, their QoS requirements and experimental environments (testbeds). The experimental data (throughput, delay/jitter and loss) are detailed in section 4 and accompanied with analysis and discussion. Finally, some conclusions are presented about QoS-related issues.

II. QoSockets

Berkeley socket is extensively used in network programming, but does not comply with QoS. QoSockets [1] extend Berkeley sockets in order to enable applications to specify and acquire QoS (if the underlying network transport supports QoS). During runtime (Fig. 1), they enable distributed applications with end-to-end QoS requirements and monitor real-time network performances for QoS management.

A. Runtime environment

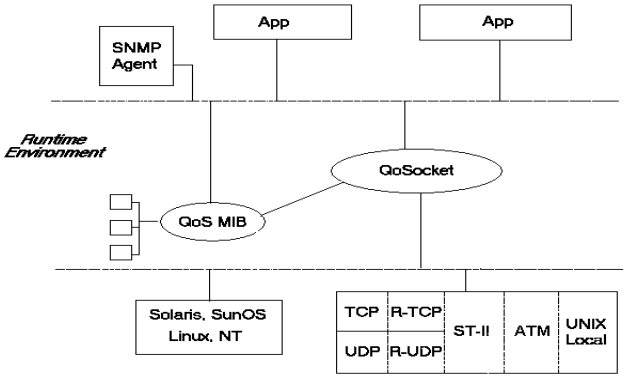

Figure 1: QoSockets Structure

Inside QoSockets, the transport layer is abstracted and unified for QoS-enabled functionality, various transport protocols are thereafter supported with real-time access. In addition, QoSockets generates QoS MIBs, provides access to multiple network protocols including *TCP, UDP, RSVP, ST-II,* and *ATM*. UNIX native protocol (*Local*) is also provided for testing an applications locally (debug use).

QoSockets consists of newly defined socket interfaces, a routine library, and auxiliary resources. The main API types provided by QoSockets are as follows.

- *Connection Establishment*: initialize and establish a QoS connection and map a QoS.
- *Collection of Communication Management Information*: collect information about network status and store into QoS MIBs.
- *Selection of Protocol & Address*: select a specific protocol, bind a user address (IP and port number) with a QoS socket.
- *Connection Management*: check the status of an input or output port in a connection, or free a connection.
- *MIB Access*: offer a set of MIB access interfaces to obtain a real-time object instance values directly inside a QoS application.

One major function is that QoSockets offers a network service to a QoS application, Fig. 2 shows that how QoSockets works with IS/RSVP [2] for a resource reservation.

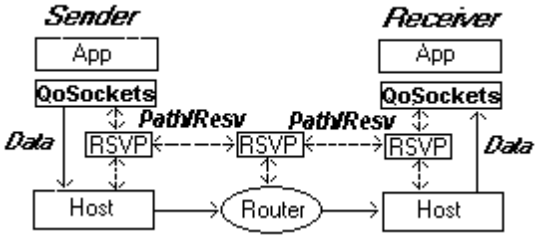

Figure 2: QoSockets and IS/RSVP

Another major function is that QoSockets offers QoS management with SNMP agent. Fig. 3 shows that how QoS MIB works with QoSockets.

When an application establishes a network traffic stream, QoSockets starts collecting its performance. It collects data from each traffic stream, including QoS specifications, connection times, transmission rates and delays, and calculates QoS parameter to determine if there is any QoS violation. All these data are stored into SNMP MIBs (Management Information Base), and can be accessible from a

QoSockets application or an SNMP agent with a remote SNMP manager.

With MIBs, QoSockets is able to control and adapt the QoS requirements for different applications, and offers an SNMP agent to access and update these MIBs, which enables an SNMP network manager (management system) to monitor network status and to adapt QoS guarantees at a remote terminal.

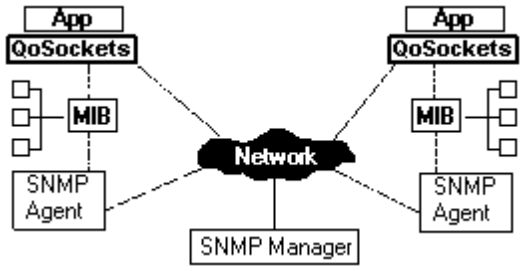

Figure 3: QoSockets and MIB

B. QoS Characterization

In most research works, major parameters for QoS requirements are *throughput*, *delay*, *reliability*, and/or *price*. In QoSockets, the first three types are included and, in addition, *coerced flag*, is introduced to coordinate QoS requirements between a sender and a receiver of an application.

1) Throughput

QoSockets defines four parameters to represent a 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*: Maximum size of communicated messages

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

Minimal: $t^i_m = \text{min\_rate}^i \times \text{size}^i$

Maximal: $t^i_M = \text{max\_rate}^i \times \text{size}^i$

Peak: $t^i_p = \text{peak\_rate}^i \times \text{size}^I$

2) Delay

Three parameters related with transmission delay are defined in QoSockets.

- *min_delay*: Lower bound of transmission delay
- *max_delay*: Upper bound of transmission delay
- *int_delay*: Maximum of inter-message delay or jitter

The above parameters are metered in milliseconds.

3) Reliability

Network reliability is defined as three parameters.

- *loss*: Percentage of message loss
- *rec_time*: Maximum time of recovering disrupted transmissions
- *permt*: Boolean flag indicates if messages may be delivered out of order

Some other parameters such as connection fails are also used for monitoring network reliability.

4) Coerced flags

QoSockets allows that both sender and receiver of a traffic stream define their own QoS parameters. Sometimes QoS parameters at two ends are not the same and need to be coerced (downgraded to a commonly accepted level). The coerced flag indicates if the coercion is allowed for each of the above throughput, delay and reliability parameters. QoSockets does this coercion only when the sender and the receiver have different QoS requirements.

For example, suppose a traffic stream is set to coerce its peak rates (by setting *coerce_peak_rate* = True) and maximal peak rates of the sender and the receiver are 64 and 60 kbps respectively, QoSockets will match two peak rates to 60 kbps, and notifies both sender and receiver this new rate. That QoS is downgraded, the sender thus bounds its peak rate to 60 kbps.

C. QoS Provisioning

QoSockets tries to allocate user specified QoS based on BBE (better than best-effort) network services such as Integrated Service (IS) [3], Differentiated Service (DS) [7] and ATM. The current implementation supports ATM and RSVP [2]. *ATM*, also known as "hard mode", is available for ATM-capable network. The QoS requirements of an application are mapped by QoSockets to an ATM service level directly.

RSVP, also known as "soft mode", is a reservation protocol of Integrated Service (IS) and available in QoSockets for TCP and UDP transport protocols (called as *R-TCP* and *R-UDP* respectively).

In the soft mode, QoSockets tells the QoS requirement of an application to an RSVP daemon, which propagates the requirements onto resource systems (hosts and routers) along the flow route, and requests them to make the resource reservation. If a

resource reservation succeeds, the application's network communication associated with this reservation may meet its QoS. When a resource reservation fails, QoSockets returns a message to the application. Combining this message with QoSockets MIB, an application knows the current status of available network resource, and then changes its QoS requirement and lets QoSockets to adapt its reservation. This paper concentrates on IS/RSVP reservations and leaves the study of ATM performance for later publications.

Our experience with QoSockets shows, even when the reservation succeeds, the end-to-end effective QoS may drift from the original negotiated QoS. There are several reasons for this. (1) Not all 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 reserving application may in fact send more messages than the reservation it requested and incur a large delay or loss. (3) Equipment may fail. The applications will have to see disruption of QoS and need to chose alternative routes.

III. QoS APPLICATIONS AND TESTBEDS

A QoS application needs at least two QoS-related services: one is to map its requirements onto the underlying system to obtain a resource assurance service; the other is to monitor real-time system performances to ensure this service. QoSockets is capable of providing these services.

In this section, we introduce two multimedia applications which are NetVideo [10], a real-time video tool, and DIRM [6], a resource management system. Their functional programs are respectively NV and IIOPGW (IIOP GateWay, the resource manager of DIRM), and replaced the socket APIs for QoSockets.

We present applications that use two different transport protocols, UDP (NetVideo), and TCP (DIRM) to investigate the different issues facing these two protocols in providing QoS. Each application is experimented with individual testbed. The next section reports on their experimental results.

A. Commons of Two Testbeds

We have set up two testbeds respectively for NetVideo (Fig. 4) and DIRM (Fig. 5) in a real network environment. Each testbed is not isolated but constructed to be a part of the Columbia Computer Science Department network. Each consists of two sub-networks: 128.59.10.0 (subnet 10) and 128.59.11.0 (subnet 11), and between them are two Cisco 2514 routers which are equipped with Cisco OS 11.2 and provide RSVP support by WFQ. Inside the two routers is an internal sub-network 192.168.1.0 over a serial cable connection in order to create a "bottleneck" bandwidth (1.5M) between the two subnets.

Hosts are connected by two hubs, each implementing a separate sub-network. Two hosts are Sun SPARCstation 20 (named *qos0*) and SPARCstation 5 (*qos1*), the CBQ patch enables their Solaris 2.5.1 kernels with traffic control support, Sun RSVP package *SolarisRSVP 0.5.0* [8] is also installed. Two PC hosts are used in the DIRM testbed: *qos10*, which is an IBM Thinkpad 760 (Pentium 166M), and *qos11*, which is a DELL Dimension XPS R400 (Pentium II 400M). Both PCs are installed with Linux 2.0.36 and Linux port of RSVP r4.2a3 package [9]. (Although these hosts are not latest devices, they are fast enough to congest the routers connected through the low bandwidth serial line).

In each experiment, two kinds of traffic flows are generated and tuned. A pair of NV or IIOPGW programs of which one is the sender and the other is the receiver, run on one host of each subnet, and generates the *main traffic flow* between two subnets. Another pair of TGs (Traffic Generator) programs run also in each subnet, and generates the *reference traffic flow*. The *main traffic* (NV or IIOPGW) includes a reserved or unreserved, UDP or TCP flow, whereas the *reference traffic* (TG) includes an unreserved UDP or TCP flows.

A test case of an experiment is composed or a combination of the following parameters: (1) reserved (with QoS) and/or unreserved (without QoS) flows; (2) under normal, heavy, or overloaded traffic condition; (3) single or multiple flows; and (4) TCP or UDP. Table 1 in next section lists all the test combinations.

NV, IIOPGW, and TG are monitored with the same parameters and sampling intervals, which are listed in the "QoS Monitoring" section of each testbed. TG is also instrumented with a monitoring module similar to QoSockets MIB management

functionality in order to monitor in real-time the network performance related to its traffic.

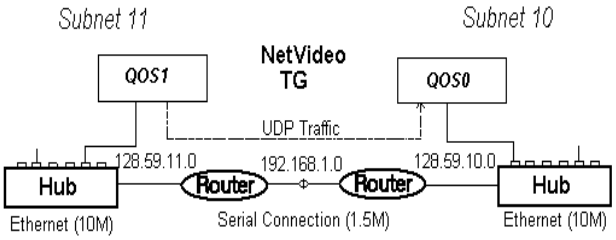

Figure 4: NetVideo testbed

B. *NetVideo Testbed*

NetVideo [10] is a multimedia tool for the Internet, captures, transfers, and receives real-time video pictures through *UDP* communication. With QoSockets, it requests QoS of its UDP traffic through *R-UDP*. It also uses QoSockets management function to monitor the network performance. We should point out that the modification is very simple replacement of traditional sockets with QoSockets.

As shown in Fig. 4, two NV programs run on two Solaris 2.5.1 workstations: *qos0* and *qos1*. *Qos1* acts as a video sender, is equipped with a video camera and captures real-time pictures up to 30 frames per second, whereas *qos0* acts as a video receiver and displays those pictures received from *qos1* onto screen. This is the *main traffic flow* of NetVideo, with or without reservation. Because the NetVideo sender can use more bandwidth than it needs, the transmitting rate of the sender can be bigger than 30 frames/s. In this test, it sends up to 80 frames/s after its bandwidth reaches 640 kbps.

Two TGs run on the same hosts as NVs, create a UDP flow in the same direction as the *main traffic flow*. This is the *reference traffic flow* without reservation. The TG sender sends a 1024-byte packet at a close rate (~ 530kbps), but the receiving rate of the TG receiver is largely dependent on different traffic condition.

Both *main traffic* and *reference traffic* flows have the same traffic route as marked "*UDP Traffic*" in Fig. 4. In addition, the sender and the receiver share the same QoS parameters since they use the same program NV.

1) QoS Requirements

User requirements

Rates: 40~80 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 **Service Type**: control-load
Rate (kbps): 614.4 **Peak** (kbps): 1024
Packet size (B): 1280

TG

Protocol: UDP **Service Type**: none
Rate (kbps): 540 **Packet size** (B): 1024

3) QoS Monitoring (Sampling interval)

Sender End

Throughput: 0.5 s

Receiver End

Throughput: 0.5 s **Loss:** 0.5 s
Delay: per packet **Jitter:** per packet

C. *DIRM Testbed*

DIRM is part of QuO project [6], and develops a high-level API that allows stream-based (socket) and object-based (CORBA[4]) applications to control QoS for their communications. Upon an application request, DIRM allocates and manages network resources dynamically, and IIOPGW is its resource manager program built using QoSockets (*R-TCP*).

Fig. 5 is a typical scenario of DIRM, where Slideshow is a client-server Java application using CORBA. The server *QoS11*, which is a CORBA object service implementation, manages a repository of image service, and the client *QoS10*, which is a CORBA client application, requests the image service through an ORB and then displays onto screen. Two IIOPGW programs run as IIOP gateways and establish a bridge between the ORBs of *QoS10* and *QoS11*, and provide QoS to the traffic from Slideshow server to client.

During the experiment, Slideshow client at *qos10* passes a object request of the image service to its local ORB, which forwards it to the local IIOP gateway IIOPGW at *qos0*. *Qos0* processes and transfers it to the remote IIOPGW at *qos1*. *Qos1* locates the object implementation repository to the Slideshow server at *qos11*. *Qos11* processes this request and then returns its object reference or

requested image to *qos10* along the reserve path of a client request.

After a client request is accepted and returned, a reservations along the path transferring images from the server (*qos11*) to the client (*qos10*) is also made at the same time by two IIOPGWs, the path is marked with "*TCP Traffic*" in Fig 5. The *main traffic flow* with or without reservation is the central part of the "*TCP Traffic*" path between *qos1* and *qos0*. Two TGs run on the same hosts as IIOPGWs, and create a TCP flow in the same traffic route as the *main traffic flow*. This is the *reference traffic flow* without reservation. TG sends or receives 1024-byte packets at a varied rate under different traffic condition.

In addition to these two flows, communications between Slideshow and IIOPGW programs generate traffic, but do not make reservations since they are in local networks. This means that, apart from the *main traffic flow*, other parts of "*TCP Traffic*" are unreserved.

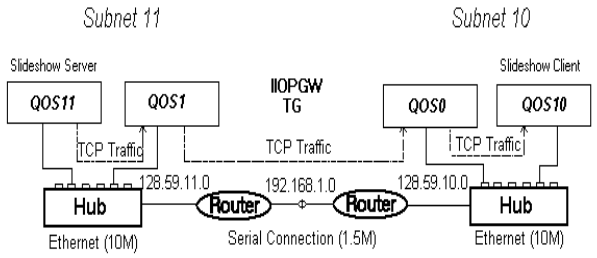

Figure 5: DIRM testbed

One thing to be mentioned here is that the packet size of IIOPGW is bigger than that of NV (1280 bytes) and TG (1024 bytes) since the JPEG images in the server repository vary from 18K to 58K bytes, so the IIOPGW flow has a much bigger burst rate up to 1440K at peak.

1) QoS Requirements

User requirements

Slides: 1~3 images/s **Delay**: 100~500 ms
Jitter: <250 ms **Loss**: 0
Max packet 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 **Service Type**: control-load
Rate (kbps): 480 **Peak** (kbps): 1440
Packet size (B): 60000

TG

Protocol: UDP **Service Type**: none
Rate (kbps): 540 **Packet size** (B): 1024

3) QoS Monitor (Sampling interval)

Same as NetVideo.

IV. RESULTS AND ANALYSIS

The major performance parameters investigated are monitored in real-time by the QoSockets MIB management component and by the TG monitoring module. They include *throughput*, *delay* & *jitter*, and *loss* sampled according to monitoring rules defined in each testbed.

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 flows: 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 flows: Overloaded	
C1	**NV w/o QoS and 2 TGs**	**IIOPGW w/o QoS and 2 TGs**
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 TGs**	**IIOPGW w/ QoS and 2 TGs**
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

Each testbed is experimented with three traffic conditions: (A) *normal*, involving a single flow of NV, IIOPGW or TG with traffic less than 50% of the bottleneck bandwidth (1.5Mbps) between two subnets; (B) *heavy*, involving two flows: one NV or IIOPGW and one TG, with traffic close to the whole bottleneck bandwidth; and (C) *overloaded*, involving

three flows: one NV or IIOPGW and two TGs, with traffic exceeding the bottleneck bandwidth.

Each condition performs 2~3 tests, all the tests are listed in Table 1. For example, under *heavy* traffic condition, two tests *B1* and *B2* are made, and *B1* generates two flows *B1a* and *B1b*. For NetVideo, *B1a* is an unreserved UDP flow generated by NV without QoS, *B2a* is a reserved UDP flow generated by NV with QoS, both *B1b and B2b* are UDP flows by TG without a QoS reservation.

This section presents statistic data, results and analyses for two experiments respectively, and follows a discussion of their results to conclude QoS with TCP and UDP later.

A. NetVideo

The figures in this section are derived from the experimental data sampled by NV and TG. For both NV and TG, 100 samples for throughput and loss every 0.5s, since the testing time of two programs varies under different traffic condition, 2000~4000 packets are transmitted and sampled for delay and jitter.

In these figures, a light gray column is a statistic value of sampling data from a sender, a dark gray column is from a receiver. Two gray columns drawn together, one light and one dark, express a flow performance in one test.

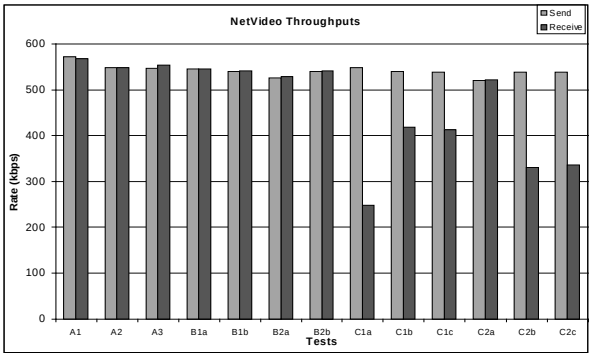

Figure 6: Throughput rates of NetVideo flows

1) Throughput

Fig. 6 shows the average throughput rates for all tested flows at the senders and receivers. Looking at these rate columns, we can conclude the following characteristics about throughput.

- For reserved NV flows (A2, B2a and C2a), their sending and receiving rates match because traffic control is applied at both sender and receiver hosts. For other unreserved NV (A1, B1a and C1a) and TG flows, their rates do not match and show considerable disparity between senders and receivers.
- For NV flows, the receiving rates of reserved flows (A2 and B2a) under normal and heavy traffic conditions are a bit less than that of unreserved flows (A1 and B1a). It is reasonable due to a tiny overhead caused by Solaris traffic-control kernel scheduling reserved flows. As expected, under overloaded traffic condition, the reserved receiving rate (C2a, 520kbps) is twice higher than the unreserved-enabled (C1a, 250kbps).
- As the traffic condition varies from normal (A), heavy (B) and overloaded (C), reserved NV flows (A2, B2a and C2a) have steady throughput rates close to 530kbps, whereas unreserved NV (A1, B1a and C1a) and TG flows reduce their throughputs approximately from 570 (A1) to 250 kbps (C1a).
- Under the overloaded traffic condition, the reserved NV flow (C2a) has close sending and receiving rates (520kbps), but the unreserved NV (C1a) and TG flows (C1b and C1c, C2b and C2c) have big disparities between their sending and receiving rates. TG flows (C2b and C2c) tested with the reserved NV flow (C2a) have rate differences (200kbps) bigger than those (C1b and C1c, 120kbps) tested with the unreserved NV (C1a).

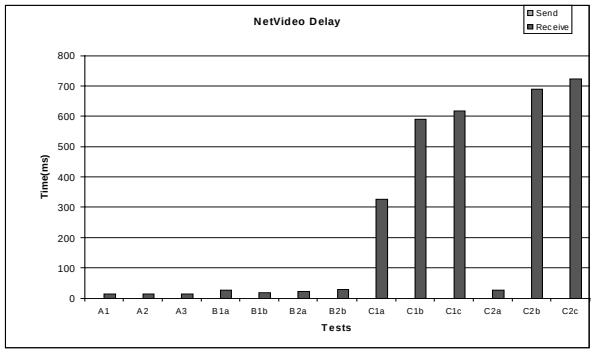

Figure 7: Packet delays of NetVideo

2) Delay

Fig. 7 shows the average delay values of all flows, which are sampled per packet arrived at the receiving ends. From this figure, we conclude about delay.

- As the traffic condition varies from normal (A) to heavy (B) and overloaded (C), reserved NV flows (A2, B2a and C2a) have steady delays (<30ms), whereas unreserved NV (A1, B1a and C1a) and TG flows increase sharply their delays.
- Under the overloaded traffic condition, the reserved NV (C2a) flow has still a low delay (25ms), but the

unreserved NV (C1a) and TG (C1b and C1c, C2b and C2c) flows have big delays (300~720ms). TG flows (C2b and C2c) tested with the reserved NV flow (C2a) jump to 700 ms and bigger than those (C1b and C1c, 600ms) tested with the unreserved NV (C1a).

3) Jitter

Fig. 8 shows the average jitter values for all tested flows, which are computed from packet delays at the receiving ends. Similar to delay, we conclude about jitter.

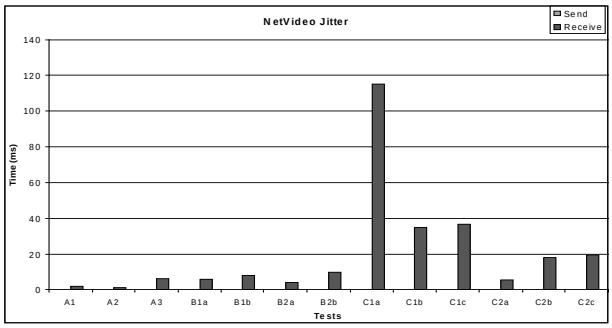

Figure 8: Packet jitters of NetVideo

- As the traffic condition varies from normal (A) to heavy (B) and overloaded (C), reserved NV flows (A2, B2a and C2a) have steady jitters (<10ms), whereas unreserved NV and TG flows increase largely their jitters.
- For unreserved TG flows, their jitters do not increase from heavy to overloaded traffic conditions. Instead, their jitters cut down, flows C1b and C1c are close to 40ms (*heavy*), C2b and C2c are 20ms (*overloaded*). This is dissimilar to the delay because the reserved NV flow (C2a) has a much smaller jitter (5ms) than the unreserved (C1a, 115ms) and results in traffic congestion reduction.

4) Loss

Packet loss is very related to throughput, and becomes bigger as the difference of sending and receiving rates of a flow increases. Fig. 9 shows the average loss rates for all tested flows, which are sampled at the receiving ends.

- Under normal and heavy traffic conditions, both reserved and unreserved flows (except A1) do not loses packets.
- Under the overloaded traffic condition, better than expectation is that the reserved NV flow (C2a) has no loss, as opposed to that the unreserved NV (C1a) gets a big loss rate (47%) and TG flows have loss rates: 25% (C1b and C1c) and 40% (C2b and C2c).

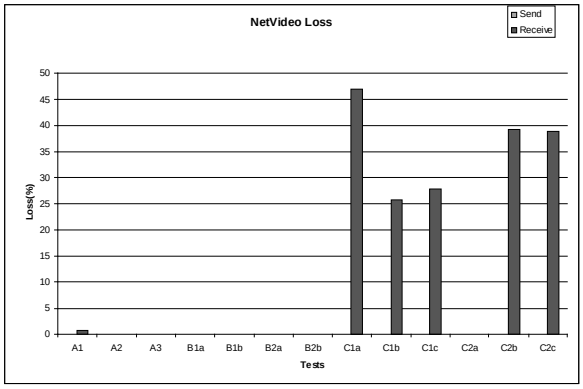

Figure 9: Packet loss rates of NetVideo

B. DIRM

DIRM testbed and testing are similar to NetVideo. 200 samples of their throughputs and losses are done to both IIOPGW and TG, 1000~10000 packets of a flow are sampled for delay and jitter for the testing time of these programs varies sharply under different traffic condition.

There are two major differences between DIRM and NetVideo testbeds. The first is *traffic*. In NetVideo, both NV and TG create one-way UDP traffic of similar size packets from subnet 11 to 10. However in DIRM, two IIOPGWs create two-way TCP traffic of varied-size packets from and to the two subnets while TG creates a one-way TCP traffic of same-size packets. The second is *loss*, since it's TCP-based there is no loss.

1) Throughput

The average throughput rates of DIRM tested flows are shown in Fig. 10, and each flow has two close columns for its sending and receiving rates due to TCP.

- 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 because of no traffic control.
- Under normal traffic condition, no obvious difference of throughput rate is between the reserved (A2) and unreserved (A1) IIOPGW. But, under heavy and overloaded traffic conditions, the reserved rates (B2a and C2a) are higher (20%) than 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 their throughput rates more or less.

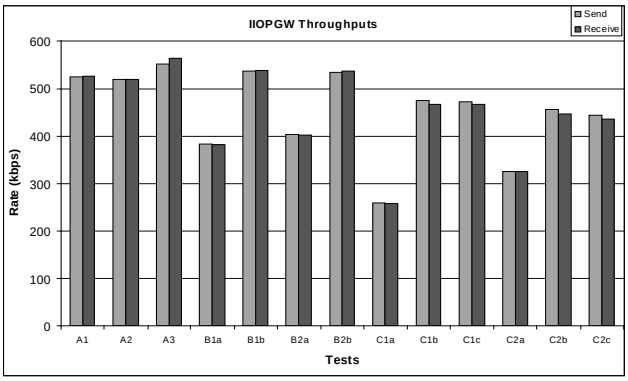

Figure 10: Throughput rates of DIRM flows

Here we note 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) reduce too? Looking back at Section IIIC "DIRM Testbed" and Fig. 5, we know that DIRM testbed has two-way traffic and not reserved all the "TCP-Traffic" path. IIOPGW flow is the main *traffic flow* and a core part of DIRM communication. Except for this flow, other flows between Slideshow client, server and their respective IIOPGWs are not reserved and take longer to transfer a whole image as traffic increases. Consequently, the reserved IIOPGW flows have to wait more time and reduce thus their throughputs too.

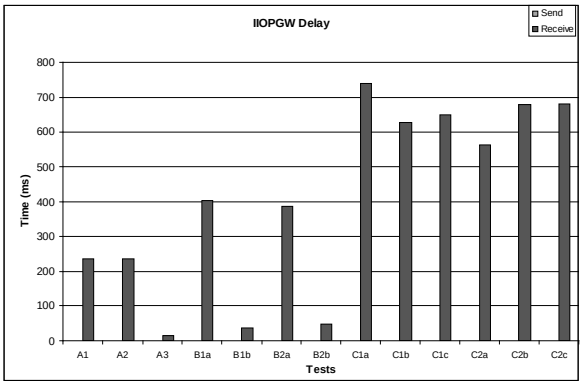

Figure 11: Packet delays of DIRM

2) Delay

Fig. 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 corresponding *reference traffic flows* (TG) have close delays (B1b vs. B2b, C1b vs. C2b and C1c vs. C2c).

It is reasonable that both IIOPGW and TG increase their delays, as the traffic condition becomes heavy or overloaded. But, why do IIOPGW flows have big delays (bigger than TG)? The reason is simple, the packet size of IIOPGW is much bigger than that of TG. The average size of IIOPGW packet is 38KB, as compared to the 1KB TG packet size, an IIOPGW packet certainly is transmitted longer than a TG packet is, and as a result its delay is bigger. These delays under normal condition prove this because both unreserved (A1) and reserved (A2) IIOPGW flows are much bigger than the TG (A3).

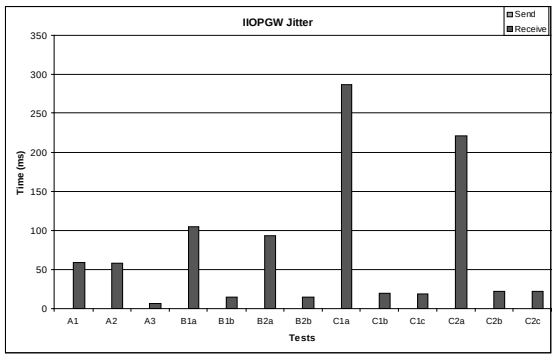

Figure 12: Packet jitters of DIRM

3) Jitter

Fig. 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 jitters.
- Under heavy and overloaded traffic conditions, the reserved IIOPGW (C2a) flow has bigger jitter than the unreserved IIOPGW (C1a). Similar to their delays, jitters of unreserved TG flows are close but small.

Similar the delay, both reserved and unreserved IIOPGW flows have bigger jitters than TG flows. As opposed to that TG has a fixed packet size (1Kb), an IIOPGW packet size is not only big (38 KB at

average) but also varies from 18 to 58 KB. So, the time difference of transmitting an IIOPGW packet is much bigger than TG, especially under the heavy or overloaded traffic condition. Also, these jitters under normal condition prove this because both unreserved (A1) and reserved (A2) IIOPGW flows are much bigger than the TG (A3).

C. Discussion

In the above experiments, two kinds of traffic flows are tested, one is the *main traffic flow* created by NetVideo/NV (UDP) or DIRM/IIOPGW (TCP), while the another is the *reference traffic flow* by TG (UDP or TCP).

In NetVideo, due to resource reservation, we get a better-than-expectation result: a reserved NV flow is able to obtain its QoS requirements of throughput, delay, jitter, and loss. Even when the traffic condition shifts from normal to overloaded, this reserved flow still gets steady throughput, low delay and jitter, and no packet loss, but the unreserved flow reduces throughput as much as 50% and loses 47% of the packets.

The DIRM experiment is more sophistical than the NetVideo one. (1) DIRM integrates a group of programs running on different platforms: IIOPGW and TG (C/C++) programs on Solaris, and Slideshow (Java) programs on Linux. NetVideo includes NV and TG (C/C++) on Solaris. (2) DIRM generates two-way TCP traffic whereas NetVideo does one-way UDP. (3) DIRM/IIOPGW transmits variable-size packets (from 18 to 58 kilobytes) with a big burst rate, whereas NetVideo/NV transmits roughly uniform-size packets (1280 bytes). (4) The *main traffic flow* of DIRM or NetVideo uses a reservation, does not cover the whole traffic path of DIRM (whereas it does that of NetVideo. Since it is TCP-based, all flows (reserved or unreserved IIOPGW and TG) do not experience any packet loss, but do reduce their throughput rates as traffic rates increase. It is of great significance that a reserved DIRM/IIOPGW flow experiences higher throughput and lower delay/jitter than an unreserved, as observed previously.

There are important reasons why the mechanisms of TCP and UDP protocols affect the QoS of their flows. While UDP creates a one-way traffic, TCP creates a two-way traffic, one for data and the other for ACKs. In fact, TCP uses ACK packets for traffic congestion control. The sender of a TCP flow waits for ACK packets returned from the receiver before sending next data packets. However, an RSVP reservation serves a one-way traffic, which is the main reason why UDP-based NetVideo gets better QoS. For TCP, the reservation guarantees only the data packets, and the ACKs are not guaranteed and thus may be delayed or even lost. When ACKs do not arrive in time, the sender blocks, which results also in reduced total throughput of the DIRM experiment. We can infer the similar QoS behavior of TCP-based applications from IS to DS because DS provides service priority to one-way packets too.

V. CONCLUSIONS

This paper describes two experiments in which existing multimedia applications have been extended to support QoS using QoSockets and tested in real network environments, their performances provide us an insight of current Internet QoS.
In summary, we conclude for UDP and TCP applications:

- Both UDP and TCP applications benefit from resource reservations and experience significant improvements in their QoS. Because of no traffic-control overhead, non-QoS flows may get better performance normally, but suffer much worse behavior under heavy or overloaded traffic conditions.
- QoSockets is able to map the generic QoS requirements of an application onto real transport systems, and helps it allocate its required QoS. In addition, QoSockets generates very important monitors of real-time network performances, which can be used by the applications for adaptation.
- TCP applications may not experience the same level of QoS as UDP applications because BBE services guarantee one-way traffic only (the TCP ACK stream may experience delay or loss). The DIRM experiment shows that the QoS of an application is dependent not only on a given service but also on its architecture. When an application creates two-way traffic or involves multiple programs and platforms, some parts involved in the whole traffic may not be able to reserve QoS and, therefore, impact the overall QoS performance.

At present, IS, DS, and ATM are the prevailing BBE services for the Internet. How to couple diverse existing applications with them is the “last mile” problem of QoS. QoSockets is a solution, and brings a

tiny (1%) additional traffic overhead. Of course, free of overhead is a right thing done for an application and, free of the "last mile" coupling is absolutely right for an existing application. We are currently investigating techniques in acquiring QoS for an existing application without its explicit participation. We are also extending the testbed to experiment with QoS of distributed applications across wide-area networks.

ACKNOWLEDGEMENTS

This work is partly sponsored by the US DARPA under Contract No. F30602-96-C-0315. We would like to thank Frank Bronzo at BBN Technologies for his contribution in the IIOPGW implementation.

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 Draft, dratf-ietf-diffserfv-arch-00.txt, August 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