QoS-Based Transport

Glenford Mapp and Steve Hodges
Olivetti & Oracle Research Laboratory
Old Addenbrooke's Site
24a Trumpington Street
Cambridge
CB2 1QA
UK
{gem,sh}@orl.co.uk

Abstract

This paper looks at transport services required by applications and proposes using a **QoS** vector to specify all transport requirements. These ideas have been pursued in the design and development of a new transport protocol called **A1**. Preliminary performance results for **A1** over ATM are presented and compared with an efficient kernel implementation of TCP/IP.

Keywords

Transport Protocols, Quality of Service, User-space Protocols

1 MOTIVATION

Support for quality of service (**QoS**) - though much talked about - has been rarely implemented in computing environments. Recent developments are changing this outlook. The first is the emergence of Asynchronous Transfer Mode (ATM) which can support QoS on a per-connection basis. The second is the adoption and deployment of RSVP [**Zhang93**] by the IP community which allows applications to specify QoS parameters on individual flows. The third impetus has been the WinSock2 application programming interface which supports QoS on socket connections. This paper examines the issue of providing realistic QoS support to applications at the transport layer. In contrast to the efforts above, a top-down approach is adopted in which application requirements form the main driving force behind the design.

2 QOS AND APPLICATIONS

Ideally an application programmer should not have to know the properties of a particular transport protocol and whether it is suitable for a given application. Instead it would be more appropriate if applications specified all their transport requirements as a QoS vector which is then mapped by the system to a specific transport protocol and appropriate operating parameters. This would be beneficial in many ways. Firstly, new transport protocols can be seamlessly added to the system without the need to inform application programmers and there would be no need to recompile or relink programs. Secondly, the transport requirements of an application may be mapped to several protocols on a given site. The system may choose a given transport protocol based on the requirements of the application, the types of networks to which it is connected and the transport protocols available at remote sites.

The ability to dynamically change the quality of service on a transport connection is becoming a much desired feature of transport services [**Pope97**]. However, it may be preferable not to involve the network every time there is a change in QoS. When a connection is being set up, the maximum QoS is determined and these parameters are submitted to the network layer. Once these parameters have been agreed the QoS can be changed at the transport layer without notifying the underlying network if the proposed change is less than or equal to the agreed maximum. A transport QoS vector is proposed below.

```
typedef struct qos {
    uchar     sort
    uchar     coptions
    ushort    max_alloc
    uint      max_data_r
    uint      max_data_t
    uint      max_data_inrate
    uint      max_data_inburst
    uint      max_data_outrate
    uint      max_data_outburst
    uint      max_latency
    uint      max_jitter
} MAXQOS, *pMAXQOS;
```

The **sort** field specifies the priority of this transport connection with regard to other connections in the system. The **coptions** field is a bit field that is used to set the properties of the connection. The properties are:

DATA_CHECK	:- checksum the data on this connection
RETRAN	:- ask for a retransmission if data is lost or corrupted
MBP	:- preserve message boundaries
RJ	:- restrict jitter: explained below
FASTCLOSE	:- close when the other end wants to close
UNI	:- data transfer in one direction only

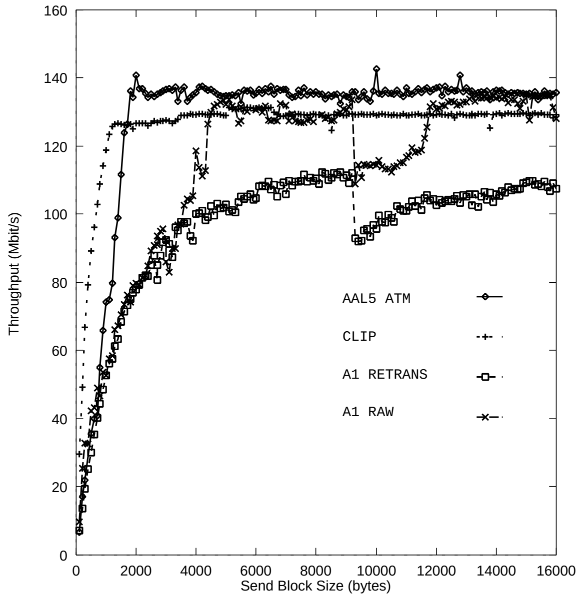

Figure 1 Throughput Measurements

For applications sending messages to each another, setting the *MBP* bit ensures that receivers get each individual message separately. The *FASTCLOSE* option is used to quickly terminate a connection once the other side has indicated that it would like to close the connection. The *UNI* option indicates that all data transfer will occur in one direction only. For some applications, getting the most recent data to the remote end as soon as possible is of paramount importance. In the *Restrict Jitter* mode, the message at the head of the receive buffer queue is deleted to provide space for incoming data when the queue is full. So the receiver gets the most current data even in overload conditions.

The **max_alloc** field specifies the maximum number of outstanding messages that the application would like to have pending while **max_alloc_r** and **max_alloc_t** specify the maximum receive and transmit message sizes respectively that the application expects to deal with. These parameters are used to allocate the necessary buffers for the connection and to set the flow control window size. The next four parameters set maximum values for the rate and burst in the inward and outward directions. For a normal connection, both rate-based and windowing flow controls are specified. In the **restrict_jitter** mode, there is no windowing flow control - though rate-based control is enforced.

3 THE A1 TRANSPORT PROTOCOL

To test some of the ideas presented in this paper, a user-level transport protocol, called **A1**, which supports different qualities of service was designed and implemented. Some preliminary performance results are given below. The tests were done using two Pentium Pros (199 bogomips/256M) connected via an ATML Virata Switch using Efficient Networks, ENI-155 Mbits/s PCI adaptor. Both machines were running Linux 2.0.25 with Werner Almesberger's Linux-ATM release 0.26. For the **A1** implementation 40-byte E164 addressing was used. The tests consist of sending a large amount of data but using block sizes up to 16K in length and measuring the throughput of the system.

Figure 1 shows the results for raw AAL5 SVC, classical TCP/IP or **CLIP** over ATM and **A1** with and without retransmission. For the AAL5 measurements, a switched virtual circuit was set up, using the ATM_UBR QoS traffic class. A 1Gbyte burst was sent and the throughput was measured for a 100 Kbyte sample. The plot shows a rather linear rise in throughput to 135 Mbits/s with the knee of the curve appearing at a block size of 1.7Kbyte. For CLIP a 10 Gbyte stream was used with the average throughput measured over the whole stream. This test program was compared with and gave identical results to *ttcp*. The same test program was used to drive **A1**. For CLIP, a throughput of 130 Mbits/second was achieved.

For **A1** with retransmission (A1 RETRANS) the throughput was measured over a 1Gbyte stream. A 9Kbyte network block size was selected to make a valid comparison with CLIP. The graph slowly climbs to about 111Mbits/s with a fall as the send block size exceeds the network block size. With **A1** without retransmission (A1 RAW), the throughput climbs on a similar curve to A1 RETRANS but continues upward to about 130 Mbits/s before falling once the send block size reaches the network block size and then rising again to a level close to the raw ATM throughput.

4 CONCLUSIONS

We think that the approach taken in this paper will lead to better transport services.

REFERENCES

[**Pope97**] S. Pope and P. Webster. Qos Support for Mobile Computing. In *this volume*, 1997.

[**Zhang93**] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, September 1993.