A Protocol for Relative Quality-of-Service in TCP/IP-based Internetworks *

Jörg Liebeherr *Alan Tai*

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
{jorg,act9m}@cs.virginia.edu

Abstract

A new protocol is presented that controls the quality-of-service of simplex end-to-end packet streams in a TCP/IP based internetwork. The protocol, called **TReg** for **T**ransport **Reg**ulation, limits the use of network resources by a rate control mechanism implemented at the traffic sources. The set of flows is partitioned into different traffic classes based on the application type, e.g., video traffic, audio traffic, or bulk data. The rate control mechanisms at the traffic sources adapt to the current traffic load such that flows within the same traffic class obtain the same quality-of-service *(Relative Quality-of-Service)*. Measurements of a prototype implementation in a network of workstations demonstrate the effectiveness of the **TReg** protocol.

1 Introduction

Today's networks must cope with a wide variety of traffic types, ranging from discrete media such as file transfers and electronic mail, to continuous media applications, such as digital audio and video. While traffic types that involve discrete media have only moderate sensitivity to the amount and variations of network delays, continuous media applications are extremely delay-sensitive and require performance guarantees for all transmitted data. The guarantees required by a single simplex end-to-end packet stream (**'flow'**), referred to as *Quality-of-Service* (*QoS*), are specified in terms of bounds on throughput, network delays, and delay variations ('jitter'). Traditional network protocols, such as TCP/IP, were designed with discrete media applications in mind and do not provide *QoS* for delay or throughput. With TCP/IP, a single flow can capture the entire capacity of a transmission channel and drastically degrade the service of all other flows on the channel.

There are several approaches to implement *QoS* for packet-switched networks. One approach, here referred to as *absolute QoS*, can guarantee *QoS* to individual flows, independent of all other flows that are active in the network. Absolute *QoS* requires the network to reserve resources for each flow, as well as use admission control functions to determine whether the network has sufficient resources to support the *QoS* of a new flow. Network architectures that provide absolute *QoS* include the Tenet Protocol Suite [4], RSVP [9], and the ISPN architecture in [2]. Note that a transition of ubiquitous TCP/IP-based networks to absolute *QoS* requires fundamental changes to the communication software of hosts and routers. In addition, since absolute *QoS* demands that resources be reserved on a per-flow basis, a network with absolute *QoS* can no longer be viewed as a shared resource that can be accessed by all users at all times.

In this study, we consider an alternative approach to *QoS* that does not reserve network resources to individual flows and, therefore, does not need admission control functions. This approach partitions flows into different traffic classes and gives identical *QoS* to all flows in the same class. The partitioning is based on the application type, i.e., video traffic, audio traffic, bulk data transfers, etc. Since a flow's *QoS* is relative to the *QoS* of all other flows in the same traffic class, we refer to this approach as *relative QoS*. Relative *QoS* does not provide admission control functions, hence, the number of flows in the network may grow arbitrarily large. As a consequence, the *QoS* of a flow can also degrade arbitrarily. Theoretical concepts for relative *QoS* have been studied before [1, 5, 6, 7]. For example, in the schemes investigated in [5, 6], all flows whose throughput is limited by the same network resource have identical throughput constraints. Obviously, the performance guarantees with relative *QoS* are weaker than those obtainable with absolute *QoS*. On the other hand, since rela-

*This is supported in part by the National Science Foundation under Grant No. NCR-9309224.

tive *QoS* does not restrict the number of active flows in the network, it maintains the notion of a network as uniformly accessible resource. Also, relative *QoS* can be implemented in the existing infrastructure of TCP/IP based internetworks without redesigning all network components.

We present the design and implementation of the **TReg** (**T**ransport **Reg**ulation) protocol, which provides relative *QoS* for existing internetworks. Rather than defining (yet another) new protocol suite, TReg is designed to easily integrate with the existing network infrastructure. Since TReg uses the standard TCP and UDP protocols for data transfer and control, it can be installed onto any network that uses the TCP/IP protocol suite. The application programming interface of TReg is almost identical to the widely used *BSD Socket* [3] interface. Thus, any application program that uses BSD sockets for communication can be easily converted to TReg with minimal changes to the source code. Finally, the implementation of TReg does not require any changes to the kernel of the hosts' operating systems. Thus, TReg is highly portable across different operating systems.

The rest of the paper is structured as follows. In Section 2 we outline the design principles of the TReg protocol. In Section 3 we discuss the implementation of the protocol and report on an ongoing prototype implementation of TReg. In Section 4, we describe the outcome of a measurement experiment in a network of workstations.

2 Design of the TReg Protocol

The TReg protocol views an end-to-end simplex traffic flow as an entity that consumes network resources on its route from the source to the destination. TReg assumes that the consumption of resources is proportional to the amount of transmitted data. Network resources include, but are not limited to, link bandwidth, CPU processing time, and buffer space. TReg controls the usage of network resources with a *rate-controlling* mechanism at the source of a flow. The maximum rate at which a flow can transmit is determined by a *rate-calculating* mechanism, referred to as the *TReg daemon*. There is one TReg daemon at each host and TReg daemons at different hosts communicate with each other to determine the maximum rate of each source. In a previous study [6] it was shown that these mechanisms are sufficient to effectively support relative *QoS* in a general packet-switched network.

Within this scope, TReg is designed with the following goals in mind:

Transparent Operation: Network applications whose flows are rate-controlled by TReg need not be aware of TReg's rate control mechanisms.

Implementation in User Space: TReg is implemented completely outside the kernel of an operating system. Therefore, existing transport and network protocols can coexist with TReg.

Multiple Traffic Classes: TReg can distinguish between a large number of traffic classes with fundamentally different characteristics (video, file transfer, http, etc.) and enforce different relative *QoS* for each traffic class.

Simple Transition: Since TReg is designed as an enhancement to the widely used TCP/IP protocol suite and the BSD Socket programming interface, existing network applications can be ported to TReg without significant changes to the applications themselves,

Currently, TReg provides a notion of relative QoS that is based on the concepts of (*intra–class fairness*) and *inter-class fairness* from [6]. Flows are assumed to be partitioned into flow classes. Intra-class fairness enforces that flows from the same class that consume identical resource types have identical throughput constraints (*max-min fairness* [1]). Inter-class fairness controls the resource consumption of entire flow classes by enforcing throughput bounds on the total traffic from all flows of a class. Each flow class has a bandwidth guarantee; if the flows from a class do not utilize the guarantee, the unused bandwidth is divided evenly among needy flow classes.

3 TReg Implementation

Here we describe the implementation of the two basic mechanisms provided by the TReg protocol: a *rate-controlling* mechanism at the sending applications (flow sources), and the *rate-calculating* mechanism at a special process, the TReg daemon.

The rate-controller at a flow source is a variation of the *leaky bucket* [8] that restricts the maximum traffic rate of a flow source. In our implementation, credits are added to the leaky bucket only up to a maximum number of credits. A packet is transmitted only if sufficient credits are available, and each packet transmission reduces the number of credits. If the leaky bucket does not contain sufficient credits, a packet is blocked until enough credits have accumulated.

The rate-controlling functions of a flow source are performed by a set of library functions, called the *TReg stub*, that is bound to an application program during compilation. In Figure 1 we sketch the rate-controlling functions of a sending application and the

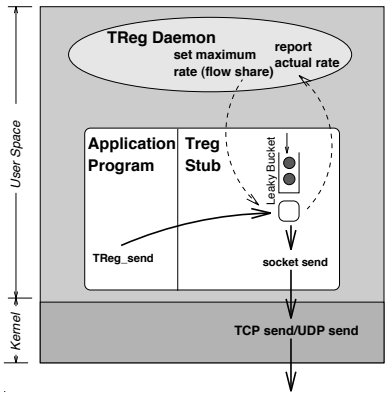

Figure 1: Operations at a *TReg Stub*.

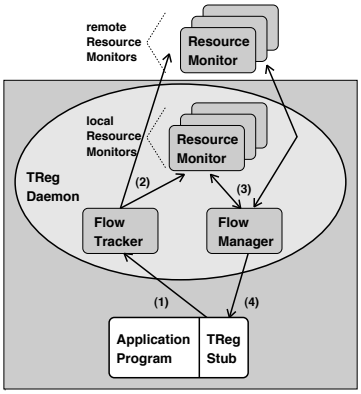

Figure 2: Operations at a *TReg Daemon*.

interaction with the functions of the TReg daemon. All send operations (*TReg_send*) of the application program are trapped by the TReg stub, which controls the maximum sending rate of the flow (*flow share*) with a leaky bucket. The credits of a leaky bucket are updated only during the execution of a *TReg_send* operation. This way, the leaky bucket does not incur overhead if an application is not transmitting. If the leaky bucket has sufficient credits, the TReg stub performs a *socket send* system call to the operating system kernel, which in turn performs a *TCP send* or *UDP send* operation. Periodically, the TReg stub reports the actual measured rate of data transmission to the TReg daemon, and the TReg daemon returns a new value for the flow share. Note that both daemon and stubs operate outside the kernel of the operating system.

Next we describe the operations of the TReg daemon to a greater detail. Recall that each TReg-aware host has exactly one TReg daemon. A TReg daemon is composed of three entities: a *flow manager*, a *flow tracker*, and a set of *resource monitors*.

The *flow manager* of a host maintains state information on all flows at this host. Whenever a new TReg stub becomes active it *registers* with the flow manager and receives a unique flow identifier. The state information kept at a flow manager includes the route and state of the flow. The route comprises all resources that a flow may consume on its way from the source to the destination, including CPUs, links, etc. The state of the flow is either *overloaded* or *underloaded*. Flows that are overloaded are prevented from transmitting at their desired rate.

The *flow tracker* of a host collects the reports of the actual transmission rates from the TReg stubs of the flows and relays this information to the resource monitors.

Each *resource monitor* collects information on a particular resource in the network. For example, for each transmission link there may be one resource monitor that collects information on this link. A network resource and its resource monitor need not be on the same host machine. The flow managers at all hosts inform the resource monitor on the number of flows that are *'overloaded'* at this resource. Periodically, a resource monitor calculates the maximum rate at which flows from each class can send data to the resource (*resource share*). The details of the calculations which are based on expressions derived in [6] are not given here. The flow managers collect the resource shares from the resource monitors and translate them into maximum transmission rates for each flow, called *flow shares*. The flow shares are sent to the TReg stubs of the sending applications to adjust the credit mechanism of the rate-controllers.

The interactions between the entities of a TReg daemon and a TReg stub are illustrated in Figure 2. The figure depicts an application program with its TReg stub and a TReg daemon. Also shown in the figure are remote resource monitors on other hosts. The TReg stub reports the actual measured transmission rate to the flow tracker of the TReg daemon (see **(1)** in Figure 2). The flow tracker collects the rate information on all flows separately for each network resource (The rate of flows that are *'overloaded'* at the resource is not included). The accumulated rate of a network resource is reported to the resource monitor for this resource (see **(2)** in Figure 2). Note that a resource monitor can be located at a different host. After fixed time intervals, a resource monitor calculates the *resource shares* for each flow class. Upon request, resource monitors report the calculated resource shares to a flow manager (see **(3)** in Figure 2). For each of its flows, the flow manager sets the flow share to the minimum resource share value that was reported for any resource on the route of the flow. Finally, the flow share value is reported to the TReg stub of the flow so that it can adjust its rate controlling mechanism (see **(4)** in Figure 2).

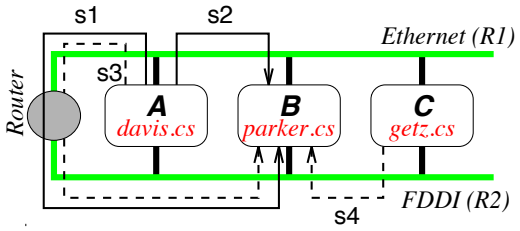

Figure 3: Testbed Network.

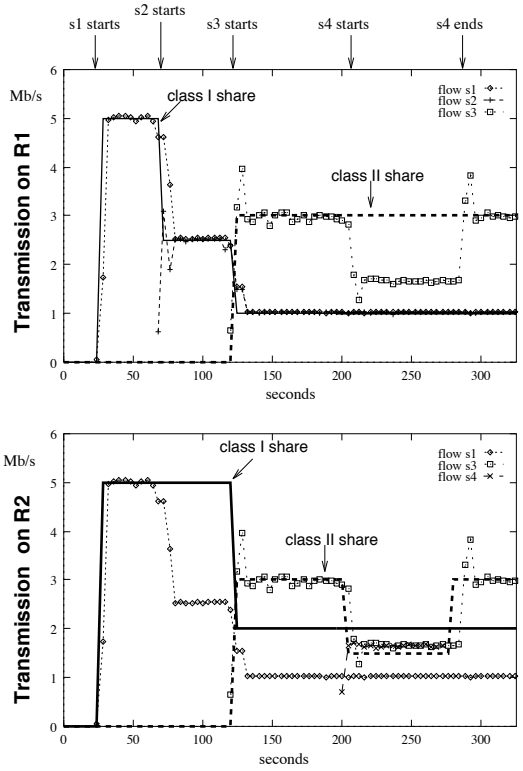

Figure 4: Experiment Results.

4 Experiment

A prototype version of TReg has been implemented in C++ and C on a network of three SGI Indy workstations shown in Figure 3. The workstations, `davis.cs` (*A*), `parker.cs` (*B*), and `getz.cs` (*C*) are connected to both an FDDI ring (*R1*) and and Ethernet bus (*R2*). Here we show an experiment to test the effectiveness of the *QoS* mechanism. The experiment involves four overloaded flows *s1...s4* from two flow classes and two resources *R1* and *R2*, representing the bandwidth made available to the flows on the FDDI and the Ethernet. The capacity of both resources is set to 5 Mb/s; 2 Mb/s are guaranteed to flow class *I* and 3 Mb/s to flow class *II*. The resource monitors are located at *getz.cs* for *R1* and *getz.cs* for *R2*. Shares are recalculated every 4 seconds. The parameters of the flows are as follows:

Flow	Route	Class	Start Time
s1	*(A, R1, R2, B)*	*I*	t = 24 s
s2	*(A, R1, B)*	*I*	t = 68 s
s3	*(A, R1, R2, B)*	*II*	t = 120 s
s4	*(C, R2, B)*	*II*	t = 200 s

Figure 4 graphs how TReg regulates bandwidth consumption on resources R1 and R2. The thick lines show the resource shares of the two flow classes at each resource, and the thin lines show the actual transmission rates of the flows. The maximum throughput of a flow is given by the smallest resource share value on the flow's route. We have verified that the throughput bounds satisfy both relative *QoS* criteria, i.e., inter-class fairness and and intra-class fairness, at all times. The graphs show that the actual transmission rates of the flows are always close to the throughput bounds. Note that the transmission rates quickly adapt after changes of the traffic load. We conclude that TReg is effective in controlling the use of resources and implementing relative *QoS*.

References

[1] D. Bertsekas and R. Gallagher. *Data Networks, 2nd Ed.* Prentice Hall, 1992.

[2] D. D. Clark, S. Shenker, and L. Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanisms. In *Proc. Sigcomm '92*, pages 14–26, August 1992.

[3] S. J. Leffler et. al. Networking implementation notes–4.3bsd edition. Technical report, Department of Computer Science, University of California, Berkeley, 1986.

[4] D. Ferrari, A. Banerjea, and H. Zhang. Network Support for Multimedia – A Discussion of the Tenet Approach. *Computer Networks and ISDN Systems*, 26(10):1167–1180, July 1994.

[5] J. M. Jaffe. Bottleneck Flow Control. *IEEE Transactions on Communications*, 29(7):954–962, July 1981.

[6] J. Liebeherr, I.F. Akyildiz, and D. Sarkar. Bandwidth Regulation of Real-Time Traffic Classes in Internetworks. Technical Report CS-94-24, University of Virginia, Department of Computer Science, June 1994.

[7] M. Steenstrup. Fair Share for Resource Allocation. Technical report, BBN Systems and Technologies, ftp://clynn.bbn.com/pub/docs/FairShare.draft9312.ps, December 1992.

[8] J. S. Turner. New Directions in Communications (or Which Way to the Information Age?). *IEEE Communications Magazine*, 25(8):8–15, October 1986.

[9] L. Zhang, S. E. Deering, D. Estrin, and S. Shenker. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 7(5):8–18, September 1993.