TCP Trunking: Design, Implementation and Performance

H. T. Kung and S. Y. Wang
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138, USA

Abstract

A TCP trunk is an aggregate traffic stream whose data packets are transported at a rate dynamically determined by TCPÕs congestion control. Typically such a trunk is implemented on top of a layer-2 virtual circuit or an MPLS label switched path. A management TCP connection is used to regulate the rate at which the trunk transmits its data packets. Setting up a TCP trunk over a circuit or a path is easy, involving only the two end nodes of a trunk to implement the management TCP connection. A TCP trunk can guarantee minimum bandwidth while being able to grab additional bandwidth when it is available. When carried by a TCP trunk, UDP flows will be constrained in their bandwidth usage, although they themselves do not perform congestion control. Experiments on testbed networks have validated these properties. TCP trunking can be an effective tool for network operators in managing bandwidth sharing between aggregates.

1. Introduction

Providing qualities of service (QoS) for the Internet has been an active area of study in recent years. For instance, QoS can be provided end-to-end at per-flow level using methods such as RSVP [18], or on a per-packet basis using methods such as Diff-Serv [12, 5].

This paper addresses the issue of providing QoS for aggregate traffic streams rather than individual flows. An aggregate stream is a collection of IP flows that are grouped together for the same routing and QoS treatment between two points in a network. It is important for carriers to differentiate service levels and provide contracted quality of service for aggregates, as their commitments to customers are likely to be at the aggregate level rather than for individual flows.

Traditionally, layer-2 circuits such as ATM virtual circuits are used to pin down network paths that will carry aggregate traffic streams. Recently, it has also been proposed that MPLS [2, 15] be used to achieve similar objectives. See [1] for discussions on requirements for traffic engineering over MPLS.

We study the use of *TCP trunks* [3, 4] as a means for providing congestion controlled aggregate streams. These streams are typically implemented on top of layer-2 virtual circuits or MPLS label switched paths. A TCP trunk is a circuit or path which transports data packets at a rate dynamically determined by TCPÕs congestion control. Unlike fixed-bandwidth circuits, a TCP trunk is elastic in the sense that it will adjust its bandwidth to adapt to changing load conditions of the network, using TCPÕs congestion control algorithms.

This paper is organized as follows: Section 2 gives an overview of TCP trunks and their properties. Section 3 describes our implementation of TCP trunks, based on a novel approach that decouples control from data in TCP congestion control [16]. The implementation will allow TCP trunks to have a guaranteed minimum bandwidth (GMB). Buffer management algorithms for routers on the path of a TCP trunk and at its sender are discussed in Sections 4 and 5. With these buffer management algorithms, a TCP trunk can assure strong properties such as no loss of user packets due to congestion, and can improve its performance regarding utilization and fairness. In Section 6, we describe TCP trunking experiments on laboratory testbed networks, and report measured performance results. In particular, we show that TCP trunks can guarantee minimum bandwidths, while being able to acquire additional bandwidths when they are available, and can protect interactive Web traffic from competing UDP or large TCP flows. Section 7 summarizes and concludes the paper.

2. Overview of TCP trunks and their properties

This section gives an overview of TCP trunks and their properties. A TCP trunk is an aggregate traffic stream on a layer-2 circuit or an MPLS path, where data packets are transported at a rate dynamically determined by TCPÕs congestion control. Figure 1 (a) depicts an IP network with

four router nodes. Figure 1 (b) shows two TCP trunks: tcp-trunk-1 from A to C and tcp-trunk-2 from D to B.

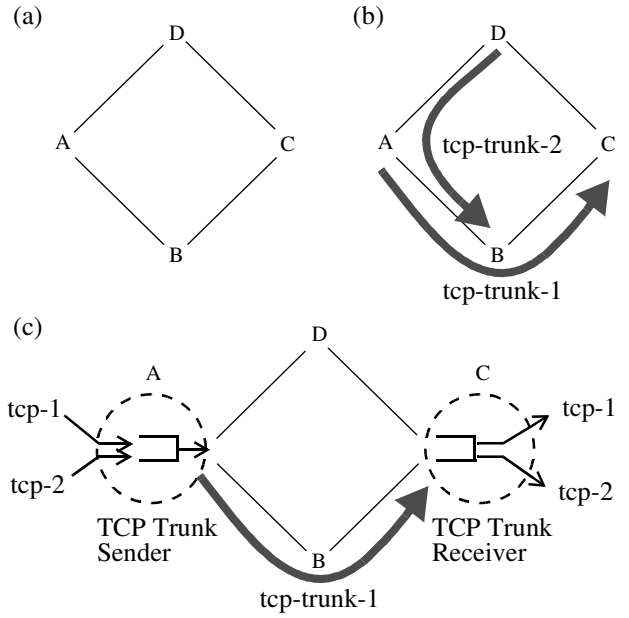

Figure 1. (a) An IP network; (b) two TCP trunks over the network; and (c) two user flows (tcp-1 and tcp-2) over tcp-trunk-1.

The layer-2 circuit or MPLS label switched path associated with a TCP trunk is called the *path* of the trunk. For example, the path of tcp-trunk-1 is from A to B and to C. The *sender* and *receiver* of a TCP trunk are, respectively, the source and destination of the trunk path. For example, tcp-trunk-1Õs sender and receiver are A and C, respectively.

Like a conventional leased line or a layer-2 virtual circuit, a TCP trunk may carry a number of *user flows*, which are host-to-host TCP or UDP flows. Packets of user flows are called *user packets*. Figure 1 (c) depicts that tcp-trunk-1 carries two user TCP flows, tcp-1 and tcp-2.

To implement TCPÕs congestion control for a TCP trunk, we use a *management TCP connection*. Over this management TCP connection, management packets will be injected by the trunk sender into the network to sample its congestion level. As in a normal TCP connection, based on arrivals of acknowledgment packets or their absence, the sender will determine the rate at which management packets will be sent. The sending rate of management packets will, in turn, determine the rate of the trunk in transmitting data packets.

The use of management packets for TCP trunks is similar to that of resource management cells [19] for ATM virtual circuits. These management packets or cells are independent of user data in the sense that they are injected into and removed from the network without intruding on the user traffic, and do not have to be aware of the user data protocols.

Under this approach, data packets are transmitted at rates determined by TCPÕs congestion control, but are not subject to retransmission and associated delays. We call this approach Òdecoupling control from data for TCP congestion controlÓ [16].

A TCP trunk can simultaneously satisfy a number of properties of interest to various applications. These include:

P1. *Guaranteed and elastic bandwidth.* (See Section 3 for implementation methods, and Figures 4 and 5 for experimental results.)

- Guaranteed minimum bandwidth (GMB): A TCP trunk can guarantee that it will deliver at least some number of bytes of data over a period of a time when there are data to be sent.
- Elastic bandwidth: Beyond GMB, a TCP trunk can grab additional network bandwidth when it is available. A TCP trunk can share the available bandwidth with other competing TCP trunks in a fair way, in proportion to the trunkÕs GMB, or in any other desired proportion.

P2. *Immediate and in-sequence forwarding.* (See Section 3.)

- At the trunk sender, arriving user packets will be immediately forwarded to the trunk, provided this is allowed by the trunkÕs management TCP. Similarly, at the trunk receiver, arriving user packets will be immediately forwarded to output network interfaces. In particular, the receiver will not wait for the arrivals of other user packets which may be delayed due to retransmission or other reasons.
- User packets arriving at the trunk sender or receiver will be forwarded out in-sequence, that is, in the order of their arrivals.

P3. *Lossless delivery.* (See Section 4 for router buffer requirements, and Figure 6 for experimental results.)

- Suppose that routers on the path of the trunk can differentiate management and user packets by packet marking, and during congestion, they will drop management packets before user packets. (This is similar to what routers supporting diff-serv [12] can do.) Then the TCP trunk can guarantee that user packets will not overflow buffers in these routers, while being able to adapt its bandwidth to network congestion using TCPÕs congestion control. (Note, however, that user packets may still be dropped at the trunk sender if they arrive at a rate higher than the available bandwidth of the trunk. This is similar to the situation when data arrive at the sender of a fixed-bandwidth leased line at a rate higher than the bandwidth of the leased line. See Section 5 for discussions on buffer management at the trunk sender.)

P4. *Aggregation and isolation.* (See Figure 9 for experimental results.)

- By aggregating a number of user flows into a single TCP flow, a trunk will reduce the number of flows

routers on the trunk path will need to handle. This will decrease packet drop rates, for the same router buffer size [11].

- By using a TCP trunk to carry UDP flows, which are not congestion controlled or not ÒTCP-friendlyÓ [20], these UDP flows can no longer starve competing TCP connections.
- By aggregating TCP flows from various user sites into dedicated TCP trunks, sites with different numbers of flows can share the network bandwidth fairly.

We note that setting up a TCP trunk involves only the two end nodes of the trunk to implement the associated management TCP. Except some diff-serv-like settings needed for assuring lossless delivery of user packets as noted above, routers on the trunk path do not require additional set up.

Using the approach of Section 3, we have implemented the TCP trunk sender and receiver on FreeBSD 2.2.7 machines. The rest of this paper describes our TCP trunking implementations, discusses their design considerations, and reports experimental results on our laboratory testbed networks.

Before experimenting with TCP trunking on the network testbeds, we used the Harvard TCP/IP network simulator [17] to simulate various properties of TCP trunks. Our results from the simulator were consistent with those from the network testbeds reported in Section 6.

3. TCP trunking implementation

This section describes our implementation of TCP trunks, for which Figure 2 provides an overview.

3.1. Management TCP

To implement TCPÕs congestion control, a TCP trunk is associated with a TCP connection, called management TCP. The management TCP is a normal TCP connection from the trunk sender to the trunk receiver but does not actually transmit any real data. The management TCP works on a Òvirtual byte stream,Ó which does not physically exist.

Each packet transmitted by the sender of the management TCP is a *management packet*, consisting of only the TCP/IP header, and no TCP payload. Thus the management packet identifies some bytes of the virtual byte stream that the packet is supposed to carry, but does not actually carry any data in the packetÕs TCP payload.

When the receiver of the management TCP receives a management packet, it performs the normal TCP receiving operations such as generation of an *ACK packet*, but appends no data to the TCP receive buffer.

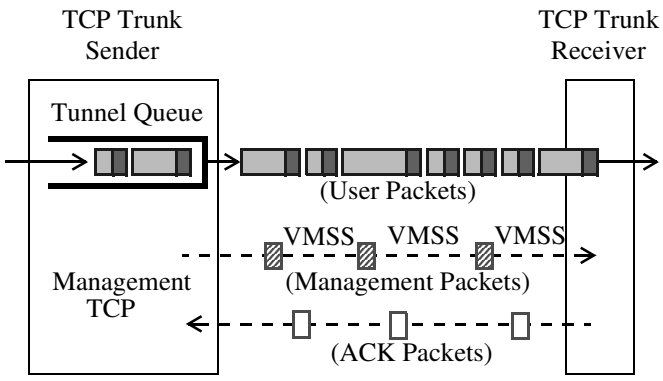

Figure 2. TCP trunking implementation. At the trunk sender, arriving packets are redirected to the tunnel queue. The tunnel queue plays the same role as a TCP socket send buffer for the management TCP, except that once a user packet has been forwarded by the management TCP, its occupied buffer space is immediately reclaimed. (Since there will be no retransmission of lost user packets, there is no need to hold transmitted user packets in the buffer.) The management TCP sends out a management packet without containing any TCP payload, each time after user packets totaling, on the average, VMSS bytes have been forwarded out. At the trunk receiver, arriving user packets are immediately forwarded out based on their headers. All operations are done in the kernel; no user level operations are involved.

The purpose of this management TCP is to control the rate at which the trunk sender sends user packets. More precisely, each management packet transmitted by the management TCP is followed by at most *VMSS* (*virtual maximum segment size*) bytes of user packets, where VMSS is a parameter to be set for the trunk. Typically, VMSS = 1500. This implies that the rate at which user packets are sent will be at most VMSS/HP_Sz times the rate at which management packets are transmitted, where *HP_Sz* is the size of management packets in bytes. Since transmission rate of management packets is regulated by the management TCPÕs congestion window, so is the transmission rate of user packets.

To smooth its bandwidth change, the trunk can employ multiple management TCPs. Suppose that there are M management TCPs. Then a 50% bandwidth reduction from any of them after TCP fast retransmit is triggered will only result in a reduction of the total trunk bandwidth by a factor of (1/2)/M. Experiment suite TT1 of Section 6.1 demonstrates smooth bandwidth transitions of TCP trunks when four management TCPs are used for each trunk.

3.2. Sending user packets via tunnel queue

User packets arriving at the trunk sender will be redirected to the queue of the tunnel interface as depicted in Figure 2, rather than the socket send buffer of the management TCP.

Each time after a management packet is transmitted or retransmitted by the management TCP, the trunk sender will be allowed to dequeue user packets, totalling at most VMSS bytes, from the tunnel queue. These packets are forwarded out from the trunk sender as ÒrawÓ packets. That is, they are not encapsulated with the TCP/IP header of the management TCP. When VMSS bytes have been dequeued from the tunnel queue, the management TCP is eligible for sending out the next management packet under the normal TCP control.

A TCP trunk will not automatically retransmit user packets in case they are lost. (This is in contrast to the management TCP of the trunk which will retransmit lost management packets.) Because different applications have different reliability requirements (e.g., FTP requires reliable data transfer and video-conferencing can live with unreliable transfer), retransmitting user packets, if it is required, should be handled by the application or some reliable protocol on the end hosts.

It is important to note that user and head packets will traverse on the same trunk path. For example, if the trunk path is on top of a layer-2 circuit or an MPLS path, then these packets will have layer-2 or shim header with the same circuit ID or path label, respectively. Because user and management packets use the same path, available bandwidth detected by probing management packets applies to user packets.

3.3. TCP trunking with guaranteed minimum bandwidth

Suppose that via bandwidth provision and connection admission, it is already guaranteed that the network can provide a guaranteed minimum bandwidth (GMB) of *X* bytes per millisecond for a TCP trunk. We describe how the trunk sender can send user packets at the GMB rate, while being able to send additional user packets under the TCP congestion control when extra bandwidth is available.

The trunk sender will use a GMB controller equipped with a timer. The GMB controller will attempt to send some number of user packets from the tunnel queue each time the timer expires. (In our implementation, the timer is set to be 1 millisecond.) When the timer expires, if there are packets in the tunnel queue, the GMB controller will send some of them under the control of a leaky bucket algorithm. The objective here is that, for any time interval of *Y* milliseconds, if there is a sufficient number of bytes to be sent from the tunnel queue, the total number of bytes actually sent by the GMB controller will approach the target of *X*Y*.

For each expiration of the GMB timer, after the GMB controller has finished sending all the user packets it is supposed to send, if there are still packets left in the tunnel queue, they will be sent out under the control of the management TCP as described in Section 3.1.

In this manner the sender will achieve its GMB under the control of the GMB controller, and also dynamically share the remaining network bandwidth under the control of the management TCP.

4. Router buffer considerations

To work with TCP trunks, a routerÕs buffer can be as simple as a single FIFO queue. The buffer will be shared by both user and management packets. To prevent loss of user packets, the router buffer will need to ensure that (1) when the FIFO queue buildup occurs, it will drop some incoming management packets early enough so that the corresponding TCP trunk sender can reduce, in time, the rate of sending user packets; and (2) the buffer will have sufficient space for user packets to accommodate control delay and possible arrival of new TCP trunks.

More precisely, the router will drop a management packet when the number of management packets in the buffer exceeds a certain threshold HP_Th. Following the arguments of [10, 11], we set:

$$HP_Th = \alpha * N \quad (1)$$

where N is the expected number of active TCP flows that will use the buffer at the same time, and α is the number of packets that the congestion window of a TCP connection must have in order to avoid frequent timeouts. A reasonable choice for α would be 8. This is because if a TCP connection has 8 or more packets in its congestion window, chances that the fast retransmit and recovery mechanism [8] can recover from a single packet loss are pretty good. Because use of RED [7] can lower the value of α somewhat, for all of our experiments reported in this paper, a simple RED-like scheme is used in routers.

Given α and N, we want to compute the required buffer size, in bytes, to ensure no loss of user packets during congestion. Let HP_Sz be the size of a management packet in bytes. Recall that VMSS is the virtual maximum segment size in bytes. (Typically, HP_Sz = 52 and VMSS = 1500.) Three types of packets may occupy the buffer of a router. We consider their size requirements:

1. *Management packets*. The required buffer size for these packets is:

$$HP_BS = HP_Th * HP_Sz$$

2. *User packets*. The required buffer size for these packets is:

$$UP_BS_TCP = HP_BS * (VMSS/HP_Sz) + N * VMSS$$

The first term reflects the fact that a user packet is VMSS/HP_Sz times larger than a management packet. The second

term takes into account the worst case that each of the N management TCPs has sent out VMSS-byte user packets but has not sent out the corresponding management packet.

3. *User packets sent under the control of GMB controllers*. Let the required buffer for these packets be UP_BS_GMB. Suppose that the fraction of the output linkÕs bandwidth allocated for the GMB traffic is β, with $\beta < 1$. Then one can expect that

$$\beta = UP_BS_GMB/(HP_BS+UP_BS_TCP + UP_BS_GMB)$$

Solving the above equation for UP_BS_GMB gives:

$$UP_BS_GMB = (HP_BS + UP_BS_TCP)*\beta/(1 - \beta)$$

Thus the total required buffer size, Required_BS, to accommodate these three types of packets is:

$$\begin{aligned} & Required_BS \\ &= HP_BS + UP_BS_TCP + UP_BS_GMB \\ &= (HP_BS + UP_BS_TCP)*1/(1- \beta) \\ &= (HP_BS + HP_BS*(VMSS/HP_Sz) +N*VMSS) \\ &\quad *1/(1-\beta) \end{aligned} \qquad (2)$$

where by Equation (1),

$$HP_BS = HP_Th*HP_Sz = \alpha*N*HP_Sz$$

The actual buffer requirement should be a few percent larger than Required_BS of Equation (2), to account for the fact that there could be a few percent more user packets than management packets in the buffer. (This corresponds to the fact that a few percent drops of management packets are normally expected due to loss of management packets when the management TCP probes for available network bandwidth.)

Thus, given the actual values or bounds for α, β, N, HP_Sz and VMSS, we can use Equation (2) to estimate the buffer requirement that will ensure no loss of user packets during congestion. Experiments have demonstrated this lossless property (see Figure 6 of Section 6).

5. Trunk sender buffer considerations

The sender of a TCP trunk will need to buffer user packets whenever they arrive at a rate higher than the available bandwidth of the trunk. When the buffer is full, arriving user packets will need to be dropped. This is similar to a fixed-bandwidth leased line whose sender will need to provide buffering. However, the TCP trunkÕs situation is more complex than the leased lineÕs situation, because the available bandwidth of the TCP trunk is subject to the control of its management TCP and thus may vary dynamically.

For the rest of this section, we assume that all user flows are TCP flows, and consider the interaction of the two levels (i.e., trunk and user levels) of TCP congestion control. We describe a solution for achieving fair and efficient use of the trunk by its user flows.

Consider the situation when a management packet of the management TCP of the trunk is dropped on the trunk path. After detecting this packet loss, the trunk sender will reduce its rate of sending user packets. In the meantime, any user flow on the trunk may not necessarily experience any packet loss yet. It will thus continue transmitting at the same or increased rate, in spite of the fact that the underlying trunk has already reduced its rate. The queue of the user flow at the trunk sender will therefore build up, until some time after a packet from the user flow is dropped at the queue due to queue overflow. The dropping of the user packet will then trigger the sender of the user flow to reduce its transmission rate.

Ideally, when the trunk reduces its bandwidth by some factor, we would want all the active user flows over the trunk to also reduce their bandwidths by the same factor. We use the following three methods to achieve this objective:

M1. Provide a buffer of size RTTup*TrunkBW to be shared by all user flows, where RTTup is an upper estimate of RTTs of user flows, and TrunkBW is the target peak bandwidth for the TCP trunk. This buffer is provided to accommodate the feed-back control delay for slowing down user flows.

M2. Use a RED-like packet dropping scheme [7] to keep the buffer occupancy at the TCP trunk sender below a certain threshold. When the threshold is reached, each arriving packet will be dropped with a probability proportional to the current buffer occupancy at the TCP trunk sender. When the buffer is full, all arriving packets will be dropped.

M3. After having dropped a packet from a user flow, the trunk sender will try not to drop another packet from the same user flow, until the user flow has recovered its reduced sending rate resulting from fast retransmit and recovery. Recall that the user flow will reduce its transmitting rate by one half. This rate reduction matches that of the underlying trunk when its management TCP loses a single management packet, and reduces its rate by one half.

We use a simple per-flow packet accounting method to implement M3 above. The trunk transmitter will estimate the total number U of packets that can be sent by a user TCP flow source between the time it reduces its sending rate by one half and the time its sending rate is about to ramp up to its previous sending rate at the time when its packet was dropped. We use this number U to set a threshold for the minimum number of packets from the user TCP flow that should be forwarded before another packet from the same flow will be dropped.

6. TCP trunk experiments and performance measurements

We have conducted TCP trunking experiments on several testbed networks, including some lab testbeds at Harvard and an academic research testbed network in Taiwan that involves a 80km ATM connection between two cities (Taipei and Hsinchu).

The hosts and routers in the testbeds are FreeBSD 2.2.7 systems running on 300 MHz PCs each with 96MB of RAM and Intel EtherExpress 10/100 cards set at 10 Mbps. A delay box implemented in the kernel is used to simulate a linkÕs propagation delay. Using the delay box, we can set the RTT of a connection to be any value with a 1-ms granularity.

These experiments have validated TCP trunksÕ properties in providing elastic and guaranteed bandwidths, delivering lossless transport over a trunk, isolating UDP flows, protecting Web traffic, etc. Due to space limitation, this section describes a subset of these experiments.

6.1. Experiments suite TT1: basic capabilities of TCP trunks

This experiment suite demonstrates the basic capabilities of TCP trunks in bandwidth management.

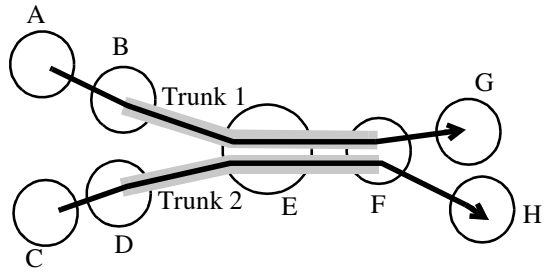

Figure 3. An experimental testbed network with 4 hosts (A, C, G and H) and 4 routers (B, D, E and F). The sender and receiver of Trunk 1 are B and F, respectively. The sender and receiver of Trunk 2 are D and F, respectively. A user flow from A to G, and another from C to H, use Trunk 1 and 2, respectively. All links are 10 Mbps. Trunks 1 and 2 share the same 10 Mbps link from E to F, which is the bottleneck link.

Below are the configurations common to experiments TT1 (a), (b) and (c):

- ¥ Each trunk uses 4 management TCPs, to provide smooth bandwidth.
- ¥ Each trunk has a FIFO buffer (tunnel queue) of 100 packets.
- ¥ The buffer in the bottleneck router E is of size Required_BS given by Equation (2) of Section 4.
- ¥ The user flows are greedy UDP flows using 1,500- byte packets.
- ¥ The propagation delay of the link between E and F is 10 ms, and that of any other link is negligible.
- ¥ Each experimental run lasts 400 seconds or longer.

Experiment TT1 (a):

Configurations:

- ¥ Trunk 1: GMB = 400 KB/sec, VMSS = 3000 bytes
- ¥ Trunk 2: GMB = 200 KB/sec, VMSS = 1500 bytes

This experiment is to demonstrate that, trunks can make full utilization of available bandwidth and share it in proportion to their GMBs. This is achieved by choosing Trunk 1Õs VMSS to be twice as large as Trunk 2Õs VMSS. Thus the achieved bandwidths should be:

- ¥ Trunk1: 400 + 2/3 * (1200 - 400 - 200) = 800 KB/sec
- ¥ Trunk2: 200 + 1/3 * (1200 - 400 - 200) = 400 KB/sec

For each of the above two equations, the first term is the trunkÕs GMB, and the second term is the extra bandwidth that this trunk should achieve when competing for available bandwidth with the other trunk. The available bandwidth is the remaining bandwidth on the bottleneck link (the link from E to F) after deducting Trunk 1 and Trunk 2Õs GMBs (400 and 200 KB/sec) from the bottleneck linkÕs bandwidth (10 Mbps = 1200 KB/sec). Since Trunk 1Õs VMSS is twice as large as Trunk 2Õs, Trunk 1 should achieve two times Trunk 2Õs bandwidth in sharing the available bandwidth. That is, Trunk 1 and 2 should achieve 2/3 and 1/3 of the available bandwidth, respectively.

The experimental results, as depicted in Figure 4, show that each trunk achieves what the analysis above predicts. That is, Trunk 1 and Trunk 2 achieve 800 and 400 KB/sec, respectively.

Experiment TT1 (b):

Configurations:

- ¥ Trunk 1: GMB = 200 KB/sec, VMSS = 3000 bytes
- ¥ Trunk 2: GMB = 400 KB/sec, VMSS = 1500 bytes

This experiment is to demonstrate that trunks can make full utilization of available bandwidth and share it in proportions independent of the trunksÕ GMBs. In this configuration, Trunk 1 has a larger VMSS value than Trunk 2, although the former has a smaller GMB than the latter.

Based on the same reasoning as that used in TT1 (a), the bandwidth allocation according to the analysis should be:

- ¥ Trunk1: 200 + 2/3 * (1200 - 400 - 200) = 600 KB/sec
- ¥ Trunk2: 400 + 1/3 * (1200 - 400 - 200) = 600 KB/sec

Again, the experimental results, as depicted in Figure 5, show that each trunk achieves about 600 KB/sec. This is what the above analysis predicts.

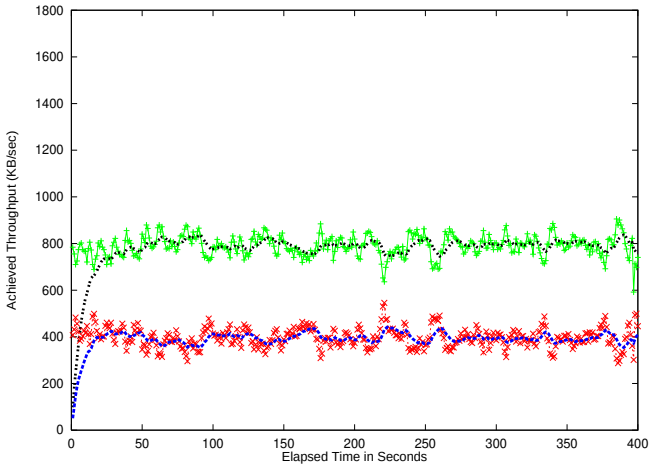

Figure 4. Results of Experiment TT1 (a). Each small point represents a trunkÕs achieved bandwidth averaged in that 1-second period around the point. The thick line represents the exponential running average of a trunkÕs achieved bandwidth over time. The achieved bandwidth of each trunk is exactly what the analysis predicts.

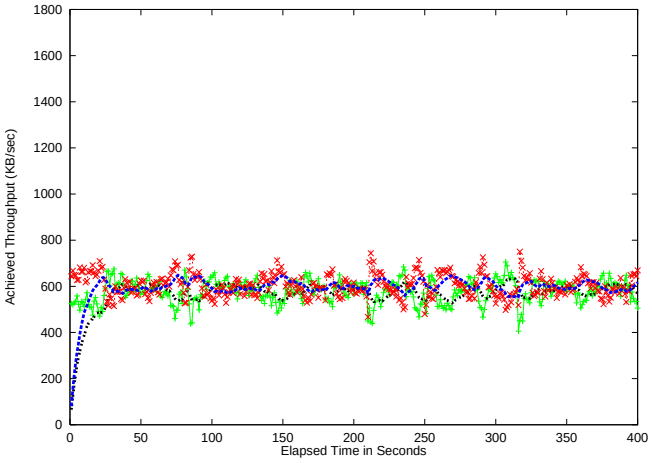

Figure 5. Results of Experiment TT1 (b). The achieved bandwidth of each trunk is exactly what the analysis predicts.

Experiment TT1 (c):

Configurations:

¥ Trunk 1: VMSS = 1500 bytes, GMB = 400 KB/sec

¥ Trunk 2: VMSS = 1500 bytes, GMB = 200 KB/sec

This experiment focuses on the buffer occupancy in the bottleneck router E. We compare it with the Required_BS value given by Equation (2) of Section 4. We are interested in verifying that there is indeed no loss of user packets in router E.

Using the notations of Section 4, the values of (α, β, N, VMSS) used for this configuration is (8, 0.5, 8, 1500). The value of α is set to be 8 so that each management TCPÕs fast retransmission and recovery can work reasonably well. The value of β is 0.5 because the sum of Trunk 1 and Trunk 2Õs GMB (400 + 200 = 600 KB/sec) is 50% of the bottleneck linkÕs bandwidth (1200 KB/sec). N is 8 because Trunk 1 and Trunk 2 together have 8 management TCP connections using the bottleneck router EÕs buffer. When plugging these values into Equation (2) of Section 4, the calculated Required_BS is 222,348 bytes.

In the 600-second run, the logged maximum buffer occupancy is 210,306 bytes. Since the buffer of Required_BS or 222,348 bytes provisioned in the experiment is greater than 210,306 bytes, there is no loss of user packets. The fact that Required_BS of 222,348 bytes is only about 5% off from the maximum buffer occupancy of 210,306 bytes suggests the high accuracy of the calculation of Required_BS by Equation (2). Figure 6 depicts sampled buffer occupancy in the bottleneck router E during this experiment.

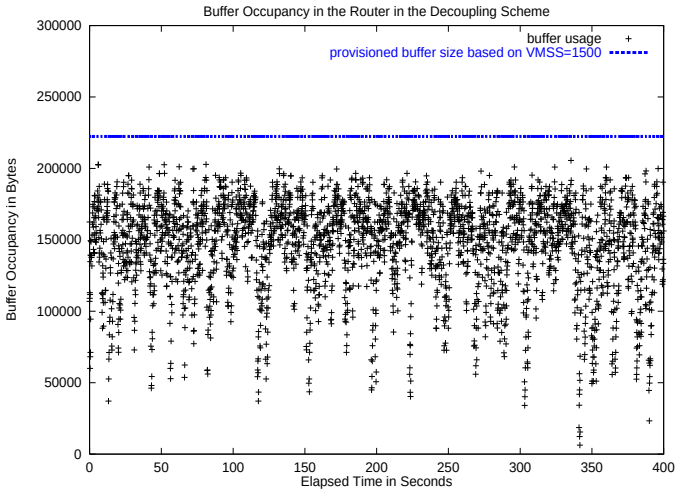

Figure 6. Results of Experiment TT1 (c). Sampled buffer occupancy in bytes in the bottleneck router E is shown. The top thick line is the Required_BS value given by Equation (2), i.e., 222,248 bytes. Note that sampled buffer occupancy is always below the line. In fact, the logged maximum occupancy is 210,306 bytes. Thus, in our experiment there is no loss of user packets.

In summary, the results of experiments TT1 (a), (b) and (c) show that a TCP trunk can:

¥ Provide GMB.

¥ Use multiple management TCPs to smooth bandwidth adaptation. (If each TCP trunk used only one management TCP rather than four, much larger bandwidth variations would have been observed.)

¥ Use different values of VMSS for different trunks to bias their bandwidth sharing. Equal sharing and proportional sharing based on trunksÕ GMBs are just two special cases of many that can be achieved.

¥ Provide lossless delivery of user packets. Figure 6 shows that the maximum buffer occupancy in the bottleneck router E is below the Required_BS value given by Equation (2) of Section 4.

6.2. Experiments suite TT2: protecting interactive Web users

This suite of experimental results, depicted in Figure 7, shows that TCP trunking can provide protection for interactive Web users when competing against long-lived greedy TCP connections. That is, short Web transfers can receive approximately their fair share of the available bandwidth and avoid unnecessary timeouts. In these experiments, each run lasts 10 minutes or longer.

Consider the configuration depicted in Figure 7 (b). On the middle router where traffic merges, there are many short-lived Web transfers coming from an input port (a site) to compete for an output port's bandwidth (1200 KB/sec) with other long-lived greedy ftp transfers that come from two other input ports (sites).

Figure 7 (a) shows that when there are only short-lived, 8KB Web transfers in the network, the offered load uses 453 KB/sec bandwidth. (The offered load is limited to 453 KB/sec, because TCP windows for these Web transfers never ramp up significantly, due to the small 8KB size of the transfers.) The request-response delays for these short-lived Web transfers are small and predictable. The mean delay, maximum delay, and the standard deviation of the delays are 353 ms, 1,270 ms, and 82 ms, respectively.

Figure 7 (b) shows that after long-lived greedy ftp transfers (Òput fileÓ sessions) are introduced into the network, the short-lived Web transfers can only achieve 122 KB/sec bandwidth in aggregate, which is much smaller than their fare share (1200/3 KB/sec). The mean delay, maximum delay, and the standard deviation of the delays increase greatly and become 1,170 ms, 11,170 ms, and 1,161 ms, respectively. This means that the short-lived Web transfers are very fragile (the reasons are discussed in [9]) and encounter more time-outs than before. As a result, they cannot receive their fair share of the bandwidth of the bottleneck link when competing with long-lived greedy ftp transfers.

Figure 7 (c) shows that when a TCP trunk is used for each site to carry the site's aggregate traffic, the bandwidth used by the short-lived Web transfers increases to 238 KB/sec. The mean delay, maximum delay, and the standard deviation of the delays also improve greatly and become 613 ms, 2,779 ms, and 274 ms, respectively.

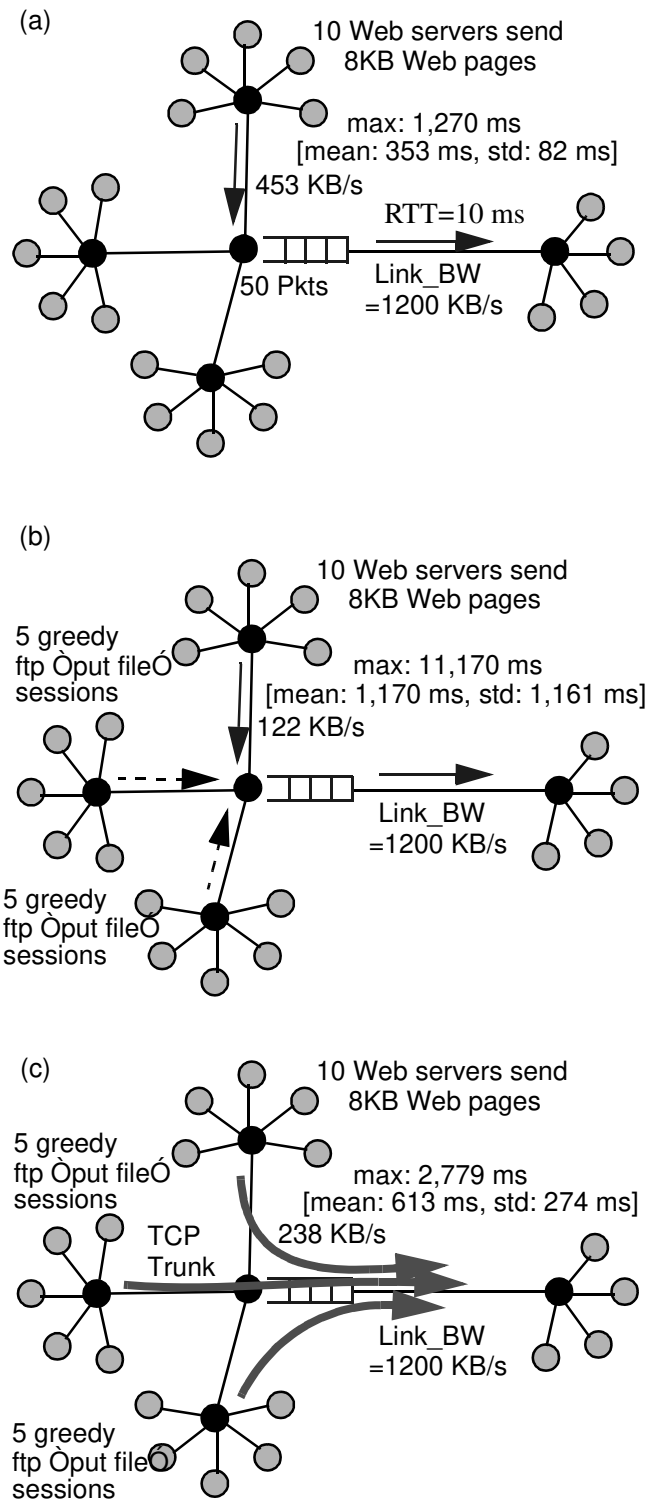

Figure 7. TCP Trunking Experiments Suite TT2. Web site throughput: (a) under no competing ftp traffic and (b) under competing ftp traffic. (c) Web side performance for load (b) when three TCP trunks, one for each site, are used.

6.3. Experiments suite TT3: protecting TCP flows against UDP flows over a ring

This experiments suite shows that TCP trunks can help protect TCP flows against UDP flows. We use a ring testbed network of Figure 8, on which TCP connections will experience multiple bottlenecks. As depicted in the figure, the testbed has five routers on the ring, five edge routers where the senders and receivers of TCP trunks are implemented, and five hosts where senders and receivers of user TCP or UDP flows reside.

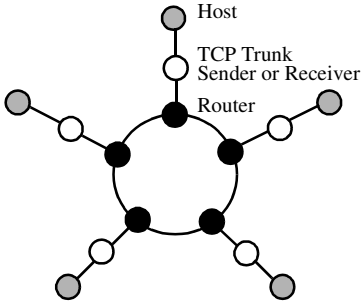

Figure 8. A ring testbed network for TCP trunking experiments TT3. The testbed consists of five hosts, five edge routers which are used as TCP trunk senders or receivers, and five routers on the ring.

All the experiment runs last 300 seconds. Configuration parameters and traffic loads have been so chosen that possible packet drops due to congestion will only occur in routers on the ring. In particular, we have configured each of these routers to have a buffer of 50 packets for management packets, and each trunk sender a buffer of 100 packets. All the links on the testbed have negligibly small propagation delays. The maximum window size for user TCP flows is 64KB.

In Case (a) of Figure 9, there are only small TCP transfers with no competing traffic. In case (b), there is a competing UDP flow from node 3 to node 4. This is an on-off UDP flow with each on or off period lasting 10ms. The source of the UDP flow will try to send as many 1024-byte UDP packets as possible during each on period. In case (c) there are two trunks: one trunk carries small file transfers from node 2 to node 1, and the other carries UDP traffic from node 3 to node 4. In case (d), there are two additional greedy long-lived TCP transfers from node 4 to node 5, and from node 5 to node 2.

Table 1 shows average throughput and delay statistics for the small file transfers from node 2 to node 1. From the experimental results, we see that these small transfers suffer when they compete with UDP traffic. Their throughput is reduced from about 380 KByte/s to about 53 KByte/s. Their mean, standard deviation, and maximum delay are increased. With TCP trunks, the situation is much

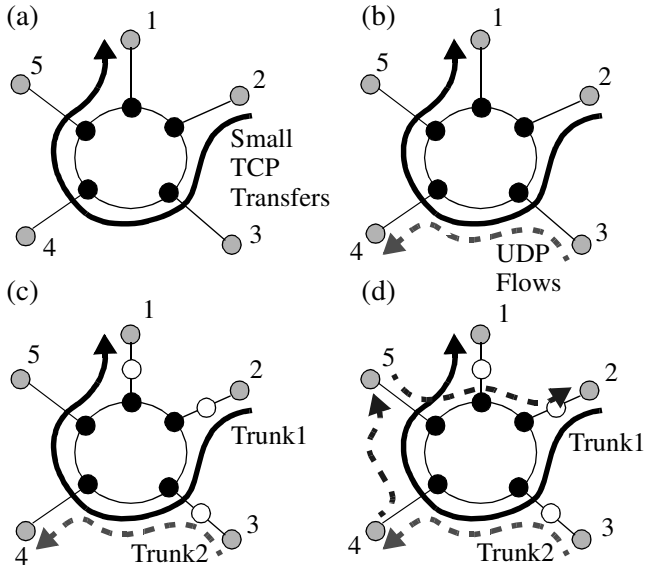

Figure 9. TCP Trunking Experiments Suite TT3: small TCP transfers compete with UDP flows. Performance results are summarized in Table 1.

improved. The throughput for small transfers increases to about 270 or 252 KByte/s for case (c) or (d), respectively. The delay statistics are also improved.

Case	Average Throughput (KByte/s)	Delay Statistics (ms) for 8 K transfers		
		Mean	SD	Max
(a)	380.05	451.5	147.9	1336
(b)	53.21	2541.1	4021.7	13053
(c)	270.45	507.9	136.5	1921
(d)	252.65	663.9	166.9	1892

Table 1: Performance results of TCP Trunking Experiments Suite TT3 (b) of Figure 9. Average throughputs and delays for small TCP transfers from node 2 to node 1 are much improved when TCP trunks are used.

7. Summary and conclusions

TCP trunking is a method of providing congestion control for aggregate traffic streams, typically implemented on layer-2 virtual circuits or MPLS label switched paths. For per flow congestion control, TCP has been the most widely used method. For aggregate streams, congestion control has been studied in the context of virtual circuits,

including explicit congestion notification methods [6, 13, 14] and ATM flow control schemes such as ABR [19].

The novelty of TCP trunking is its specific proposal of using TCPÕs congestion control for aggregate flows. This approach takes advantage of a large body of experience and knowledge built up over the years about TCP, regarding its efficient and fair use of the network, and its buffer management requirements in routers. None of the other congestion control methods for virtual circuits have comparable knowledge bases.

This paper describes how normal TCP congestion control algorithms can be effectively applied to the trunking environment. This involves the concepts of management TCP and decoupling control from data for TCP congestion control, as well as buffer management algorithms for the trunk sender and for routers on the trunk path, and methods for guaranteeing minimum bandwidth. We are able to preserve congestion control effects of TCP algorithms, while avoiding some of their drawbacks such as loss of user packets when TCP probes network for available bandwidth, and delays due to retransmission of lost user packets.

TCP trunking suggests a new way of providing QoS attributes to layer-2 circuits and MPLS label switched paths. The approach can realize various bandwidth management objectives such as guaranteed minimum bandwidths and elastic sharing of available bandwidth; transport objectives such as lossless delivery of user packets; and aggregation and isolation objectives such as protecting Web connections and isolating UDP traffic. Experimental results of this paper have validated that TCP trunks can indeed achieve these objectives.

For further information on the design and implementation of TCP decoupling, experimental results about TCP trunks, as well as migration and deployment issues, see [16].

Acknowledgment

This research was partially supported by Nortel, Sprint, Air Force Office of Scientific Research Multidisciplinary University Research Initiative Grant F49620-97-1-0382, and National Science Foundation Grant CDA-94-01024.

References

[1] D. O. Awduche, J. Malcolm, J. Agogbua, M. OÕDell and J. McManus, ÒRequirements for Traffic Engineering Over MPLS,Ó IETF draft-ietf-mpls-traffic-eng-00.txt, October 1998.

[2] R. Callon et al., ÒA Framework for Multiprotocol Label Switching,Ó draft-ietf-mpls-framework-02.txt, Nov. 1997.

[3] A. Chapman, and H. T. Kung, ÒEnchancing Transport Networks with Internet Protocols,Ó IEEE Communications Magazine, Vol. 36, No.5, May 1998, pp. 100-104.

[4] A. Chapman, and H. T. Kung, ÒTraffic Management for Aggregate IP Streams,Ó Proc. the 3rd Canadian Conference on Broadband Research, Nov. 1999.

[5] D. D. Clark and W. Fang, ÒExplicit Allocation of Best-Effort Packet Delivery Service,Ó IEEE Transactions on Networking, Vol. 6, No. 4, August 1998.

[6] S. Floyd, ÒTCP and Explicit Congestion Notification,Ó ACM Computer Communication Review, V. 24 N. 5, October 1994, p. 10-23.

[7] S. Floyd and V. Jacobson, ÒRandom Early Detection Gateways for Congestion Avoidance,Ó Transactions on Networking, August 1993.

[8] V. Jacobson, ÒCongestion Avoidance and Control,Ó Proc. SIGCOMM'88 Symp., Aug. 1988, pp. 314Ð32.

[9] D. Lin, and H. T. Kung, ÒTCP Fast Recovery Strategies: Analysis and Improvements,Ó Proceedings of IEEE INFOCOM'98, pp. 263-271.

[10] D. Lin, and R. Morris, ÒDynamics of Random Early Detection,Ó SIGCOMMÕ97.

[11] R. Morris, ÒTCP Behavior with Many Flows,Ó IEEE ICNP, 1997.

[12] K, Nichols, V. Jacobson, L. Zhang, ÒA Two-bit Differentiated Services Architecture for the Internet,Ó IETF draft, 1998.

[13] K. K. Ramakrishnan, and S. Floyd, ÒA Proposal to add Explicit Congestion Notification (ECN) to IP,Ó RFC 2481, January 1999.

[14] K. K. Ramakrishnan and R. Jain, ÒA Binary Feedback Scheme for Congestion Avoidance in Computer Networks with Connectionless Network Layer,Ó ACM Transactions on Computer Systems 8(2): 158-181, May 1990.

[15] E. C. Rosen, A. Viswanathan, and R. Callon, ÒMultiprotocol Label Switching Architecture,Ó IETF draft-ietf-mpls-arch-05.txt, April 1999.

[16] S. Y. Wang, ÒDecoupling Control from Data for TCP Congestion Control,Ó Ph.D. Thesis, Harvard University, September 1999. (http://www.eecs.harvard.edu/networking/decoupling.html)

[17] S. Y. Wang and H.T. Kung, ÒA Simple Methodology for Constructing an Extensible and High-Fidelity TCP/IP Network Simulator,Ó INFOCOMÕ99.

[18] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, ÒRSVP: A New Resource Reservation Protocol,Ó IEEE Network 7(5), September 1993, pp. 8-18.

[19] ATM Forum Traffic Management Specifications 4.0.

[20] The TCP-Friendly Website, http://www.psc.edu/networking/tcp_friendly.html, 1999.