TCP Trunking for Bandwidth Management of Aggregate Traffic

H.T. Kung

kung@eecs.harvard.edu
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138, USA

S.Y. Wang

shieyuan@csie.nctu.edu.tw
Department of Computer Science and Info. Engr.
National Chiao Tung University
Hsinchu, Taiwan

Abstract

TCP trunking is a novel way of applying TCP congestion control to bandwidth management of aggregate traffic. This is accomplished by setting up a separate TCP connection to probe network congestion, and then using the TCP connection's congestion control to regulate the bandwidth usage of the aggregate traffic.

TCP trunking offers several advantages in managing bandwidth of aggregate traffic. It can dynamically allocate bandwidth to competing traffic aggregates in a fair manner, while assuring no loss of user packets due to network congestion. In addition, it can transmit user packets at guaranteed rates for traffic aggregates. Implementation details and experimental results are described in the paper.

1. Introduction

Traffic aggregation has been an essential technique in managing backbone networks of the Internet. It can reduce the number of states that backbone nodes need to maintain for routing [3] and for providing quality of service (QoS) [4, 5]. It can also reduce the number of flows that backbones need to handle, in order to lower packet loss rates [6].

As aggregate flows become widely used, their congestion control will be important. Traditionally TCP has been the dominant protocol that provides congestion control for individual flows between end hosts. In this paper we show that TCP is also suited in providing congestion control for aggregate flows between network nodes.

Our approach, which we call TCP trunking, uses a separate TCP connection to provide congestion control for aggregate traffic. The TCP trunking approach offers several advantages. For example, it can provide fair and dynamic bandwidth allocation as well as guaranteed transmission rates for traffic aggregates, while assuring no loss of user packets due to network congestion.

Author names are listed in alphabetical order.

An earlier version of this paper [1] was presented at the ICNP'99 conference. More details about TCP trunking can be found in [2].

This paper is organized as follows: Section 2 gives an overview of TCP trunks. Section 3 presents the design goals for TCP trunking. Section 4 describes our implementation of TCP trunks. Buffer management algorithms for routers on the path of a TCP trunk and at its sender are discussed in Sections 5 and 6. In Section 7, we describe TCP trunking experiments on laboratory testbed networks and report measured performance results. Section 8 discusses related work. Section 9 concludes the paper.

2. Overview of TCP Trunks

A TCP trunk is a network path which carries traffic of multiple user flows and has its total bandwidth usage regulated by the TCP congestion control algorithm. For a four-node IP network of Figure 1 (a), Figure 1 (b) shows two TCP trunks: tcp-trunk-1 from A to C and tcp-trunk-2 from D to B.

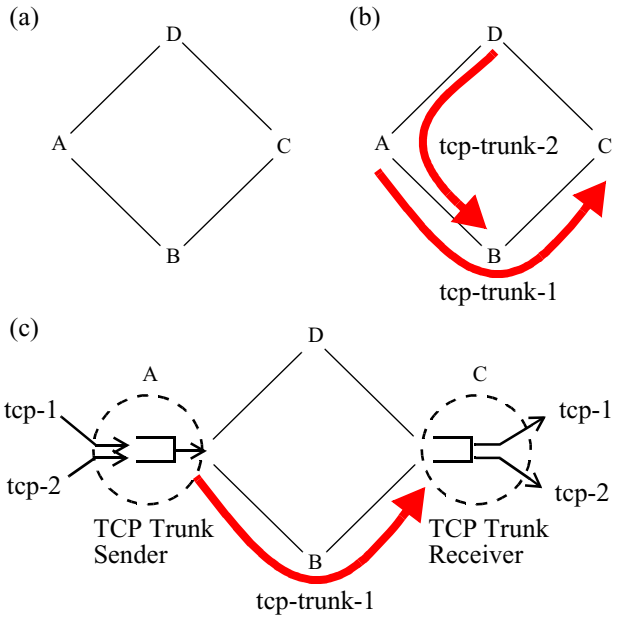

Figure 1. (a) An IP network; (b) two TCP trunks over the network; and (c) two user flows (tcp-1 and tcp-2) over tcp-trunk-1.

Figure 1 (c) depicts that tcp-trunk-1 carries two TCP user

flows: tcp-1 and tcp-2. We call packets of these user flows *user packets*.

To implement TCP congestion control of a TCP trunk, a *management TCP connection* (also called management TCP for simplicity) is set up. The sender and receiver of this management TCP connection are called the trunk sender and receiver, respectively. For example, in Figure 1 (c), tcp-trunk-1's sender and receiver are on A and C, respectively.

Packets of the management TCP are *management packets*. Since the management TCP is only for congestion control purposes, a management packet contains only a TCP/IP header and carries no TCP payload. As in a normal TCP connection, based on received acknowledgment packets or their absence, the trunk sender uses a congestion window to determine the sending rate of management packets.

In turn, the sending rate of management packets determines that of user packets. That is, only each time after having sent a management packet, the trunk sender will send some additional number (VMSS of Section 4.1) of bytes of user packets. In this way the management TCP regulates the sending rate of user packets of the trunk.

The routing path taken by 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 path of a TCP trunk is normally a layer-2 circuit. In the future, it can also be an MPLS path [7, 8]. The use of layer-2 circuits or MPLS paths ensures that user and management packets of a TCP trunk will take the same path. This allows the use of management packets to probe congestion level of the path for user packets.

3. Design Goals for TCP Trunks

In this section, we describe design goals for TCP trunks. For each goal, we list those sections and figures in the paper which discuss and demonstrate how it is achieved.

G1. *Guaranteed and elastic bandwidth.* See Section 4 for implementation, and Figures 4 and 5 for experimental results.

- Guaranteed minimum bandwidth (GMB): Suppose that via a separate admission control mechanism (a topic beyond the scope of this paper), a TCP trunk is given a guaranteed minimum bandwidth on every hop of its path. Then the trunk sender can send its user packets at least at this rate at all time.
- Elastic bandwidth: Beyond GMB, a TCP trunk can grab additional network bandwidth when it is available. This property can also keep the overall network utilization high.

G2. Flexible and fine-grained bandwidth allocation. See Section 4 for implementation, and Figures 4 and 5 for experimental results.

- By choosing proper VMSS values, competing TCP trunks can share the available bandwidth flexibly, in any desired proportion and in any granularity. For example, if trunk 1 and trunk 2's VMSSs are 1,500 and 500 respectively, then the achieved bandwidth of trunk 1 will be about 1500/500 = 300% times that of trunk 2. If trunk 1 and trunk 2's VMSSs are 1500 and 1499 respectively, then the achieved bandwidth of trunk 1 is about 1500/1499 = 1.000667% times that of trunk 2.

G3. *High-speed, low latency, and in-sequence forwarding.* See Section 4 for implementation.

- The TCP sender and receiver will forward arriving user packets immediately to the next hop as is and in-sequence.

G4. *Lossless delivery.* See Section 5 for required buffer management and provisioning in routers, and Figure 6 for experimental results.

- Suppose that routers on the trunk path can differentiate management and user packets by packet marking, and during congestion, they can drop management packets before user packets. (This capability is similar to what routers supporting diff-serv [4, 5] can do.) Then TCP trunking guarantees that user packets will not be dropped inside routers due to congestion.

G5. *Traffic aggregation and isolation.* See Figure 7 for hierarchical TCP trunking, and Figure 10 and Figure 12 for experimental results.

- A TCP trunk uses a single TCP connection to provide congestion control for multiple TCP user flows. This reduces the total number of competing TCP flows on a backbone network. This can reduce packet drop rates in the backbone [6].
- When carrying UDP user flows, a TCP trunk can isolate them from other competing flows. Note that UDP flows for multimedia streaming applications are usually not "TCP-friendly" [9], that is, they do not employee adequate congestion control. By containing them in a TCP trunk (perhaps with some GMB), these congestion unresponsive UDP flows can no longer starve other competing TCP flows.
- When carrying "fragile" short-lived TCP user flows such as transfers of small Web pages, a TCP trunk can protect them from competing large flows. These fragile flows can thus experience reduced delays and packet loss rates.
- When carrying aggregate traffic from multiple user sites, TCP trunks can ensure fair use of a backbone independent of the number of TCP users flows a site may have.

To validate the TCP trunking approach, we have implemented it on FreeBSD 2.2.8 machines. We have constructed

laboratory testbeds with nodes based on these FreeBSD machines. We have done TCP trunking experiments on these testbeds. In addition, we have performed intensive simulations on the Harvard TCP/IP network simulator [10] for large network configurations. The simulation results are consistent with the experimental results reported in Section 7.

In the rest of this paper we describe our TCP trunking implementation, discuss its design considerations, and report experimental results obtained on our testbeds.

4. TCP Trunking Implementation

This section describes our implementation of TCP trunks. An implementation overview is depicted in Figure 2.

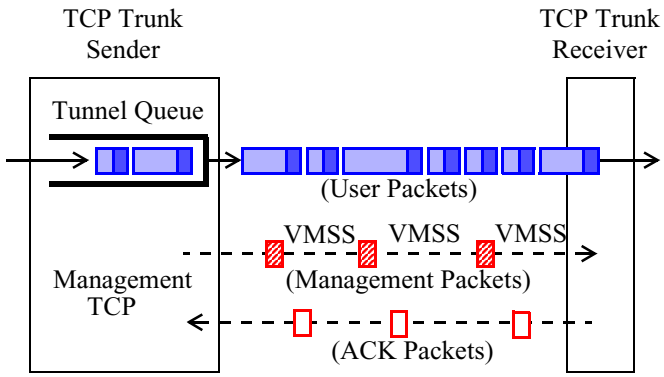

Figure 2. Overview of TCP trunking implementation. At the trunk sender, arriving packets are redirected to the tunnel queue. The tunnel queue plays the same role as a TCP socket send buffer for the management TCP, except that once a user packet has been forwarded by the management TCP, its occupied buffer 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.) Each time after VMSS bytes of user packets have been forwarded out, the management TCP can send out a management packet that contains only a TCP/IP header but no TCP payload. At the trunk receiver, arriving user packets are immediately forwarded out based on their headers. All operations are done efficiently in the kernel; no user level operations or overhead are involved.

4.1. Management TCP

To provide TCP congestion control for user packets of a TCP trunk, a management TCP connection is set up. The management TCP is a normal TCP connection from the trunk sender to the trunk receiver except that it does not transmit any real data. The sender of the management TCP sends out *management packets*, which contain only TCP/IP headers and no TCP payloads. This is sufficient for the purpose of the management TCP, which is to provide congestion control for user packets. The management TCP can be thought as a normal TCP connection that transports a "virtual byte stream," which does not physically exist.

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*. However, the receiver does not need to append any data to the TCP receive socket buffer since it receives no real data.

Each time when the trunk sender sends out a management packet, it will also send *VMSS* (*virtual maximum segment size*) bytes of user packets. VMSS is a parameter to be set for the trunk, and typically it is 1,500 (Ethernet's MTU). This means that the rate at which user packets are sent will be VMSS/MP_Sz times the rate at which management packets are transmitted, where *MP_Sz* is the size of management packets in bytes. Since the sending rate of management packets is regulated by the management TCP's TCP congestion control, so is that of user packets.

To smooth bandwidth change, the trunk may 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 7.1 demonstrates smooth bandwidth transitions of TCP trunks when four management TCPs are used for each trunk.

4.2. Sending User Packets via Tunnel Queue

User packets arriving at the router on which the trunk sender operates will be redirected to the queue of a tunnel interface as depicted in Figure 2. From the kernel's point of view, a tunnel interface is no different from a normal interface such as Ethernet [11]. This means that the redirection can be accomplished simply by changing a routing entry in the router.

Each time after a management packet is transmitted or retransmitted by the management TCP, the trunk sender receives VMSS credits and can dequeue VMSS bytes of user packets from the tunnel queue. These user packets are forwarded out by the trunk sender as independent packets in the sense that they are not encapsulated with the TCP/IP header of the management TCP. When VMSS bytes have been dequeued from the tunnel queue and forwarded out, the management TCP is allowed to send out the next management packet provided that its TCP congestion control allows. Note that since a packet needs to be transmitted in its entirety, sometimes not exactly VMSS bytes of user packets are dequeued and sent out. In this case, credits are borrowed and later returned to solve the problem.

A TCP trunk does not automatically retransmit lost user packets. The buffer space occupied by a user packet in the tunnel queue is reclaimed immediately after the packet is dequeued and forwarded. This allows applications to use

different retransmission policies based on their own reliability requirements. (E.g., FTP requires reliable data transfer whereas video-conferencing can live with unreliable transfer.) For some real-time applications such as video conferencing, retransmitted packets may arrive at the receiver too late to be useful, and thus will only waste network bandwidth. Therefore when retransmitting user packets is desired, it will be handled by applications at the end hosts. In contrast, for the management TCP which is used to probe network congestion, a lost management packet must be retransmitted to keep the probe going.

It is important that user and management packets traverse on the same trunk path so that the congestion level detected by probing management packets is applicable to user packets. 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 identifier or path label, respectively, to ensure that user and management packets use the same path.

4.3. TCP Trunking with Guaranteed Minimum Bandwidth

Suppose that via admission control and bandwidth reservation 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 TCP congestion control when extra bandwidth is available.

The trunk sender uses a GMB controller equipped with a timer. The GMB controller attempts 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 that 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 4.1.

In this manner, the sender will send user packets at GMB under the control of the GMB controller and, at the same time, dynamically share the available network bandwidth under the control of the management TCP.

5. Buffer Management and Provisioning in Routers

To work with TCP trunks, a router's buffer can be as simple as a single FIFO queue for all TCP trunks' user and management packets. The router uses the following buffer management scheme to prevent loss of user packets due to buffer overflow. When the FIFO queue occupancy starts to build up, the router will drop some incoming management packets. Dropping these management packets will trigger their corresponding TCP trunk senders to reduce their sending rates of user packets and thus lower the congestion level. The router does the packet dropping sufficiently early to ensure that before the buffer is full, the congestion level will have been lowered so that buffer overflow will not happen.

More precisely, the router will drop a management packet when the number of management packets in the buffer exceeds a certain threshold MP_Th. Following the arguments of [6], we set:

$$MP\_Th = \alpha * N \quad (1)$$

where N is the expected number of active management TCPs 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 time-outs. 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 can recover from a single packet loss are pretty good [12]. Because experimental results show that use of RED [13] 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, under the condition that the router always keeps the number of management packets below MP_Th, we can compute the maximum buffer occupancy, Require_BS, in Equation (2) below. By configuring the router buffer to be larger than Require_BS, we can ensure that no user packets will be dropped due to buffer overflow. That is, only management packets will be dropped during congestion, not user packets.

Let MP_Sz be the size of a management packet in bytes. Three types of packets may occupy the buffer of a router. We consider their maximum buffer occupancy respectively as follows.

(1) Management packets

The maximum buffer occupancy of these packets is:

$$MP\_BS = MP\_Th * MP\_Sz$$

(2) User packets sent under management TCP control

The maximum buffer occupancy of these packets is:

$$UP\_BS\_TCP = MP\_BS*(VMSS/MP\_Sz) + N*VMSS$$

The first term reflects the fact that a user packet is VMSS/MP_Sz times larger than a management packet. The second term takes into account the situation that each of the N management TCPs has sent out VMSS-byte user packets but not the corresponding management packet.

(3) User packets sent under GMB control

Let the maximum buffer occupancy of these packets be UP_BS_GMB. Suppose that during the admission time the fraction of the output link's bandwidth allocated for the GMB traffic is β, with $\beta < 1$. Then one can expect that when the buffer occupancy builds up, the fraction of the buffer space occupied by GMB packets is about β. That is,

$$\beta = UP\_BS\_GMB/(MP\_BS+UP\_BS\_TCP + UP\_BS\_GMB)$$

Solving the above equation for UP_BS_GMB gives:

$$UP\_BS\_GMB = (MP\_BS + UP\_BS\_TCP)*\beta/(1 - \beta)$$

Thus the maximum buffer occupancy, Required_BS, of these three types of packets altogether is:

$$\begin{aligned} & Required\_BS \\ &= MP\_BS + UP\_BS\_TCP + UP\_BS\_GMB \\ &= (MP\_BS + UP\_BS\_TCP)*1/(1- \beta) \\ &= (MP\_BS + MP\_BS*(VMSS/MP\_Sz) +N*VMSS) \\ &\quad *1/(1-\beta) \end{aligned} \qquad (2)$$

where by Equation (1),

$$MP\_BS = MP\_Th*MP\_Sz = \alpha*N*MP\_Sz$$

The actual maximum buffer occupancy will be a few percent larger than Required_BS of Equation (2). The reason is that, to provide the "lossless" property for user packets, some management packets are dropped while their corresponding user packets are not.

Given the actual values or bounds for α, β, N, MP_Sz and VMSS, we can use Equation (2) to calculate the maximum buffer occupancy. Then the router can be configured with a buffer size larger than the calculated value to provide the "lossless" property for user packets. Experiments have demonstrated this lossless property and the accuracy of Equation (2) (see Figure 6 of Section 7).

For routers whose buffers are sized in packets rather than bytes, one can do an analysis similar to the one above to estimate Require_BS in packets. That is, by making use of the fact that all user packets must have some minimum number of bytes, one can derive an upper bound on the number of user packets that a block of VMSS bytes can contain. Using the upper bound, one can then calculate the number of user and management packets that may need to be buffered in a router to prevent loss of user packets.

6. Buffer Management on Trunk Sender

The sender of a TCP trunk will need to buffer user packets when 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 also needs to provide buffering and, when necessary, drop packets. However, unlike the leased line's situation, the available bandwidth of a TCP trunk may vary dynamically subject to the control of its management TCP.

In this section, we consider the case in which all user flows of a TCP trunk are TCP flows. We show how the interaction of the two levels (i.e., trunk and user levels) of TCP congestion control can be dealt with, to allow a TCP user flow to dynamically adapt to the trunk's available bandwidth without being timed out.

Various buffer management and packet scheduling schemes can be used at the trunk sender such as single FIFO or per-flow queueing, and RED or round-robin scheduling. We considered the use of per-flow queueing with round-robin scheduling as it is well-known that this can provide traffic isolation and bandwidth allocation. However, per-flow queueing with round-robin scheduling alone can easily cause a trunk's TCP user flows to time-out. At the time when the trunk's bandwidth is suddenly reduced by a half by its management TCP's congestion control, if these per-flow queues are almost full, then a TCP user flow may time-out due to multiple packet drops in a row. To mitigate this problem, we use a RED-like packet dropping scheme to keep the occupancy of these per-flow queues under a low threshold most of the time. This ensures that when the trunk's bandwidth suddenly reduces, there is enough buffer space to temporarily hold arriving packets without dropping them all.

In our experiments, rather than using per-flow queueing with round-robin scheduling, we used a single FIFO with per-flow packet accounting to implement our RED-like packet dropping method. Our experimental results show that our method can fairly allocate bandwidth among TCP user flows while keeping them running smoothly without time-outs.

When the trunk reduces its bandwidth by some factor, we need all the active user flows over the trunk to reduce their bandwidths by the same factor. Therefore, when a per-flow queue's buffer occupancy suddenly increases, which is

a signal that the underlying TCP trunk's bandwidth shrinks, the TCP user flow should be notified to reduce its sending rate. Note that for a TCP user flow, a single packet drop suffices to make its sender reduce its sending rate by a half. In our RED-like packet dropping method, the trunk sender will try not to drop another packet from the same user flow until the user flow has successfully recovered from its fast retransmit and recovery.

More precisely, when the trunk's bandwidth is reduced by a half, the trunk sender estimates the congestion window size W of each active TCP user flow by dividing the current congestion window size of the management TCP by the number of active TCP user flows. (This is based on the assumption that every active TCP user flow uses about the same share of the trunk's bandwidth.) From W, we derive the total number U of packets that can be sent by a TCP user flow source during a fast retransmit and recovery cycle. That is, U = W/2 + (W/2+1) + (W/2+2) + ... + W. This is the number of packets sent between the time the source reduces its sending rate by a half and the time its sending rate is about to ramp up to its previous sending rate when its packet was dropped. We use U/2 as a threshold for the minimum number of packets from the same TCP user flow that should be forwarded before another packet from the same flow will be selected to be dropped. Choosing U/2 is based on the observation that if a TCP flow can still send out U/2 packets after reducing its congestion window size from W to W/2, the chance that the TCP sender has successfully recovered from TCP fast retransmit and recovery without timing-out is very high. Therefore, the TCP flow can now be subject to another rate reduction if necessary.

7. TCP Trunk Experiments and Performance Measurements

We have conducted TCP trunking experiments on several testbed networks, including some laboratory testbeds at Harvard and National Chiao Tung University in Taiwan. The hosts and routers in the testbeds are FreeBSD 2.2.8 systems running on 300 or 550 MHz PCs with 128MB of RAM and Intel EtherExpress 10/100 cards set at 10 or 100 Mbps. A delay box implemented in the kernel is used to simulate a link's 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 trunks' properties in providing elastic and guaranteed bandwidth, hierarchical bandwidth allocation, providing lossless transport for user packets, isolating UDP flows, protecting Web traffic, etc. This section describes a representative set of these experiments.

7.1. Experiments Suite TT1: Basic Properties of TCP Trunks

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

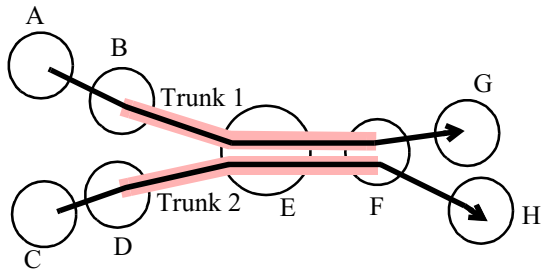

Figure 3. An experimental testbed network with 4 hosts (A, C, G and H) and 4 routers (B, D, E and F). The sender and receiver of Trunk 1 are B and F, respectively. The sender and receiver of Trunk 2 are D and F, respectively. A user flow from A to G, and another from C to H, use Trunk 1 and 2, respectively.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

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

- Each trunk uses 4 management TCPs to smooth bandwidth change of the trunk. (See a discussion of multiple management TCPs for a single trunk at the end of Section 4.)
- 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 5.
- 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 1's VMSS to be twice as large as Trunk 2's VMSS. Under this setting, the achieved bandwidths of Trunk1 and Trunk2 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 trunk's GMB, and the second term is the extra bandwidth that this trunk should obtain 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 2's GMBs (400 and 200 KB/sec) from the bottleneck link's bandwidth (10 Mbps = 1200 KB/sec). Since Trunk 1's VMSS is twice as large as Trunk 2's, Trunk 1 should achieve two times Trunk 2's 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 indeed achieves what the above analysis predicts. That is, Trunk 1 and Trunk 2 achieve 800 and 400 KB/sec, respectively.

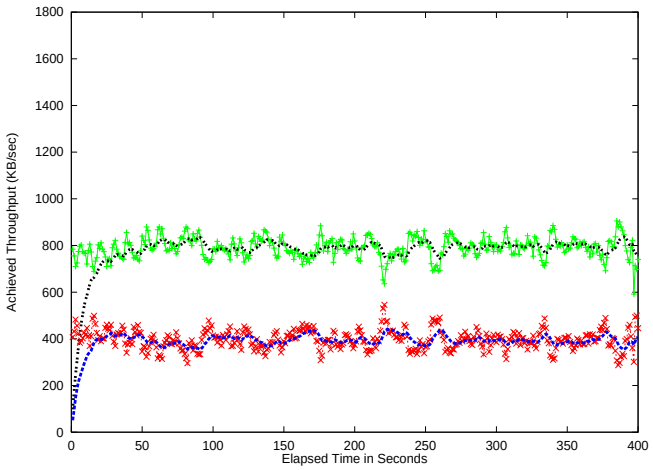

Figure 4. Results of Experiment TT1 (a). Each small point represents a trunk's achieved bandwidth averaged in a 1-second period around the point. The thick line represents the exponential running average of a trunk's achieved bandwidth over time. The achieved bandwidth of each trunk is exactly what the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

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 trunks' 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.

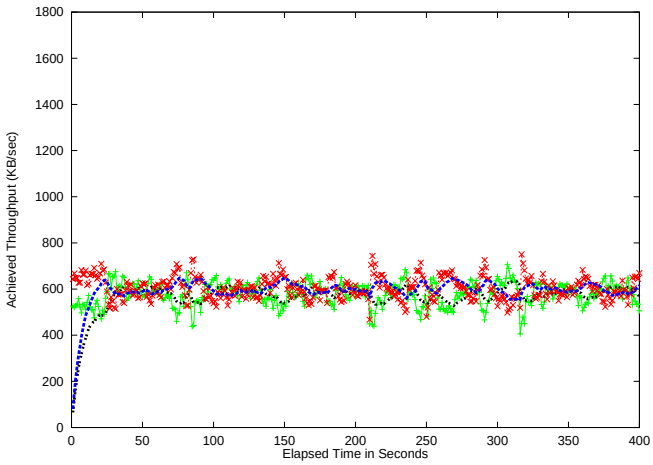

Figure 5. Results of Experiment TT1 (b). The achieved bandwidth of each trunk is exactly what the analysis predicts.

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 5. We are interested in verifying that there is indeed no loss of user packets in router E.

Using the notations of Section 5, the values of (α, β, N, VMSS) used for this configuration are (8, 0.5, 8, 1500). The value of α is set to be 8 so that each management TCP's TCP fast retransmit and recovery can work reasonably well. The value of β is 0.5 because the sum of Trunk 1 and Trunk 2's GMB (400 + 200 = 600 KB/sec) is 50% of the bottleneck link's bandwidth (1200 KB/sec). N is 8 because Trunk 1 and Trunk 2 together have 8 management TCP connections. When plugging these values into Equation (2) of Section 5, we find Required_BS to be 222,348 bytes.

In a 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

that Equation (2) calculate Required_BS accurately. Figure 6 depicts sampled buffer occupancy in the bottleneck router E during this experiment.

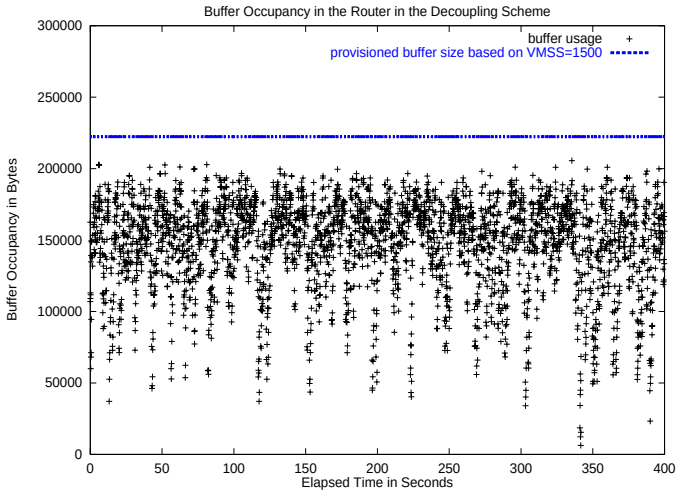

Figure 6. Results of Experiment TT1 (c). Sampled buffer occupancy in bytes in the bottleneck router E is shown. The top thick line is the Required_BS value given by Equation (2), i.e., 222,248 bytes. Note that sampled buffer occupancy is always below the line. In fact, the logged maximum occupancy is 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 for the VMSSs of different trunks to bias their bandwidth sharing in a fine-grained way. Equal sharing and proportional sharing based on trunks' GMBs are two special cases of many that can be achieved.
- Provide lossless delivery of user packets. Figure 6 demonstrate that the maximum buffer occupancy in the bottleneck router E is below the Required_BS value given by Equation (2) of Section 5.

7.2. Experiments Suite TT2: Bandwidth Management via Hierarchical TCP Trunking

This experiment suite shows bandwidth management via hierarchical TCP trunking involving nested TCP trunks. As depicted in Figure 7, the experimental network testbed has 16 nodes connected together by 10 and 100 Mbps

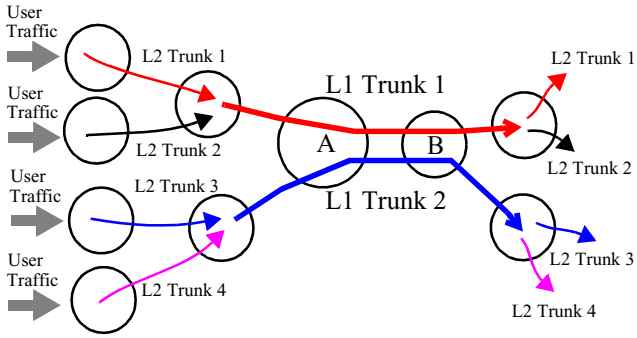

Figure 7. The experimental testbed network to demonstrate TCP trunk's hierarchical bandwidth allocation capability. In total, 16 nodes are used. Two of them are used as the shared traffic source and sinker node respectively, for the four level-2 trunks. (For presentation clarity, not all used nodes are shown in this figure.) All links are 10 Mbps except for those that connect to the shared traffic source and sinker nodes, which are 100 Mbps. There are two level-1 trunks contending for the bandwidth of the bottleneck link from node A to node B. In each level-1 trunk, there are two level-2 trunks contending for the achieved bandwidth of the level 1 trunk.

Ethernet links and hubs. For presentation clarity, not all nodes are shown in the figure.

This experiment demonstrates a two-level TCP trunking system. On level one, there are two TCP trunks (L1-T1 and L1-T2) contending for the bottleneck link's bandwidth. On level two, within each of level-1 TCP trunks, there are two competing level-2 TCP trunks (L2-T1 and L2-T2 inside L1-T1, and L2-T3 and L2-T4 inside L1-T2).

Some parameters important to this suite are as follows. Each TCP trunk has 4 management TCPs as before. The value of VMSS used by all of these management TCPs are 1,500. The GMB is set to zero for all TCP trunks. The user traffic in each level-2 TCP trunk is UDP greedy traffic.

During the experiment, we purposely inject various amounts of load into the four level-2 TCP trunks at different times to test the dynamic and hierarchical bandwidth allocation capability of TCP trunking. We show the achieved bandwidth of the two level-1 TCP trunks in Figure 8 and the achieved bandwidth of the four level-2 TCP trunks in Figure 9, respectively.

At the beginning of the experimental run, on level 1, only L1-T1 is active and inside L1-T1, on level 2, only L2-T1 is active. From Figure 8, we see that L1-T1 achieves 100% of the bottleneck link's bandwidth and, from Figure 9, L2-T1 achieves the full bandwidth of L1-T1.

At the 28th second, the second level-2 trunk L2-T2 inside L1-T1 becomes active. From Figure 8, we see that because on level 1, L1-T1 is still the sole active trunk, it remains to occupy 100% of the bottleneck link's bandwidth. From Figure 9, we see that the two competing level-2 trunks

(L2-T1 and L2-T2) inside L1-T1 share the bandwidth of L1-T1 fairly.

At the 58th second, the level-2 trunk L2-T3 becomes active. This makes the level-1 L1-T2 that contains L2-T3 become active as well. From Figure 8, we see that the two competing active level-1 trunks (L1-T1 and L1-T2) share the bandwidth fairly. From Figure 9, we see that the two level-2 trunks (L2-T1 and L2-T2) inside L1-T1 share the bandwidth of L1-T1 fairly, and the sole level-2 trunk (L2-T3) inside L1-T2 uses the whole bandwidth of L1-T2.

At the 90th second, the level-2 trunk L2-T4 becomes active. It starts to compete with L2-T3 for the bandwidth of L1-T2. From Figure 9, we see that now L2-T3 and L2-T4 fairly share the bandwidth of L1-T2. From Figure 8, we see that L1-T1 and L1-T2 still fairly share the bottleneck link's bandwidth on level 1. The introduction of a new active level-2 trunk in L1-T2 does not affect the bandwidth allocation on level 1 between L1-T1 and L1-T2.

In summary, the above experimental results show that TCP trunking can dynamically allocate bandwidth for nested TCP trunks.

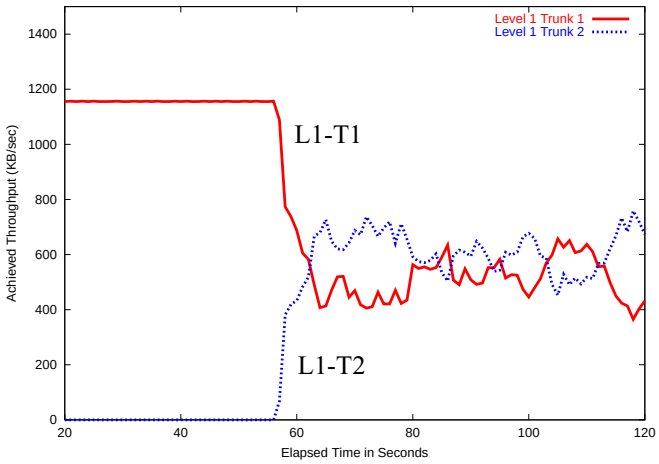

Figure 8. Achieved bandwidth of the two level-1 trunks.

7.3. Experiments Suite TT3: Protecting Interactive Web Users

This suite of experimental results, depicted in Figure 10, 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 time-outs. In these experiments, each run lasts 10 minutes or longer.

Consider the configuration depicted in Figure 10 (b). On the middle router where traffic merges, there are many

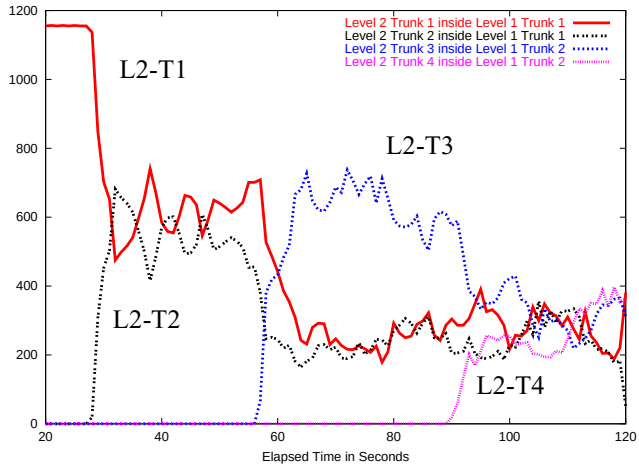

Figure 9. Achieved bandwidth of the four level-2 TCP trunks.

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 10 (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 10 (b) shows that after long-lived greedy ftp transfers ("put file" 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 [12]) 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 10 (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.

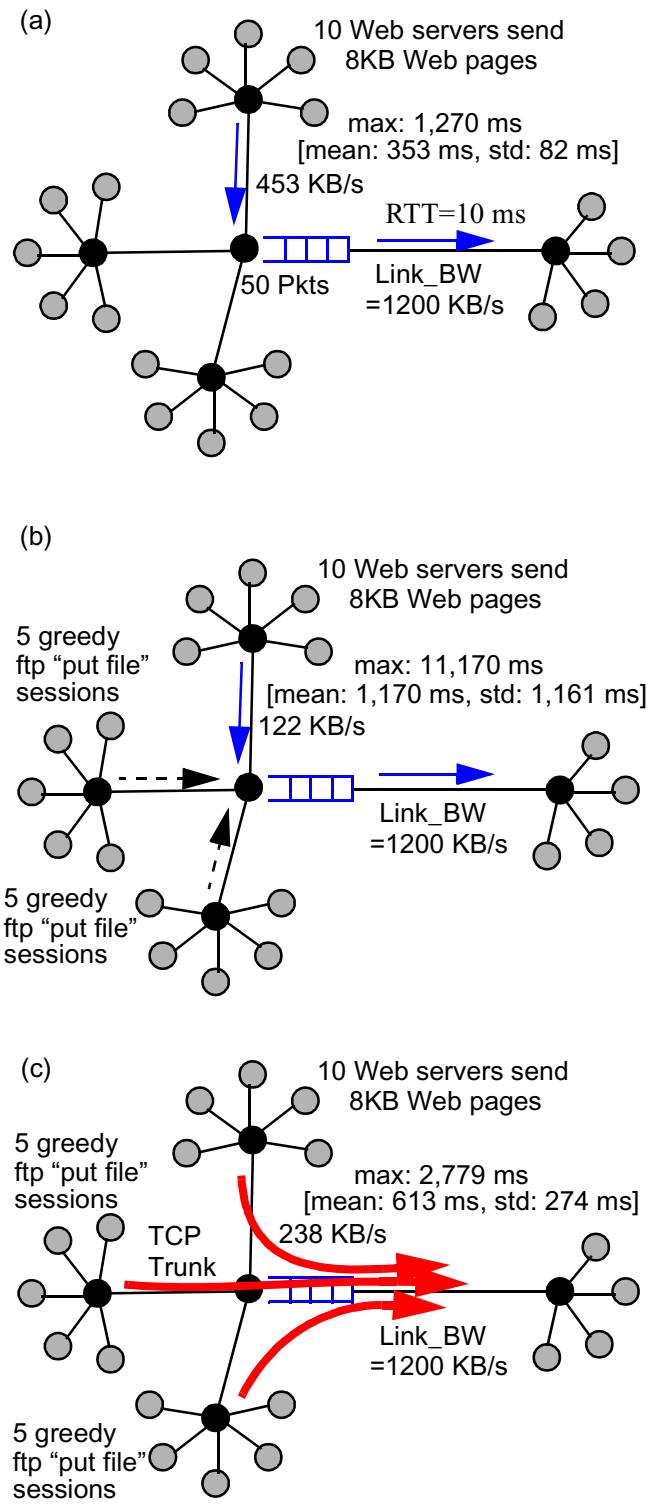

Figure 10. TCP Trunking Experiments Suite TT3. Web site throughput: (a) under no competing ftp traffic and (b) under competing ftp traffic. (c) Web side performance for load (b) when three TCP trunks, one for each site, are used.

7.4. Experiments Suite TT4: 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 11, 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.

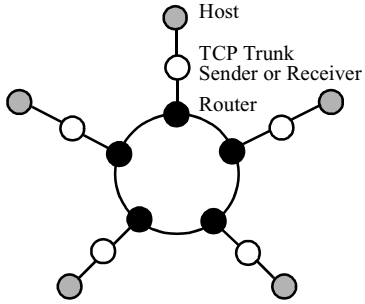

Figure 11. A ring testbed network for TCP trunking experiments TT4. The testbed consists of five hosts, five edge routers which are used as TCP trunk senders or receivers, and five routers on the ring.

All the experiment runs last 300 seconds. We configured each of these routers to have a buffer of 50 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 12, 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 improved. The throughput for small transfers increases to about 270 and 252 KByte/s for case (c) or (d), respectively. The delay statistics are also improved.

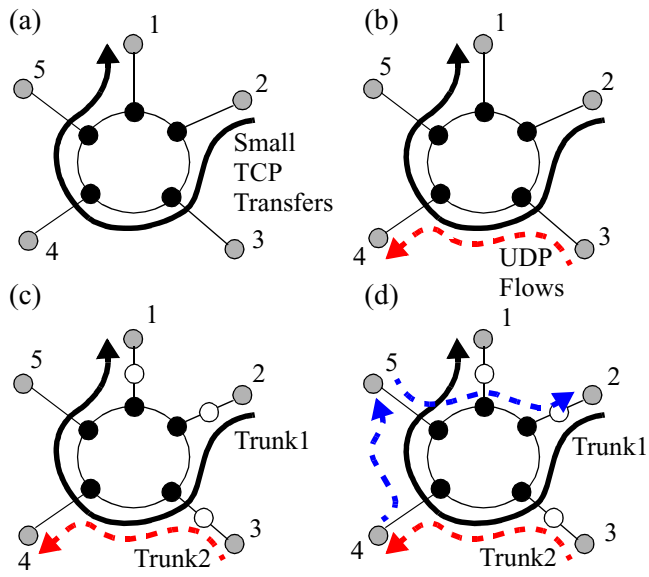

Figure 12. TCP Trunking Experiments Suite TT4: small TCP transfers compete with UDP flows. In (d), two other external traffic flows are introduced to create the multi-bottleneck configuration. Performance results are summarized in Table 1.

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 TT4 (b) of Figure 12. Average throughputs and delays for small TCP transfers from node 2 to node 1 are much improved when TCP trunks are used.

8. Related Work

The approach of using management packets for TCP trunks is similar to that of using resource management cells [14] for ATM ABR virtual circuits. These management packets or cells are independent of user traffic in the sense that they are injected into and removed from the network without modifying the user packets and they do not have to be aware of the user data protocols.

Being the dominant congestion control protocol used in the current Internet, TCP has received much attention from many researchers (e.g., [15, 16, 17]). The TCP trunking approach can incorporate advanced TCP congestion control results when they become available.

Explicit congestion notification methods [18, 19, 20] such as ECN mark user packets to indicate congestion in routers. These marked packets will then signal their source to reduce their sending rates. Although ECN methods potentially can provide the "lossless" property for user packets, there are some problems with them. Marking packets in routers alone does not automatically achieve the "lossless" property. That is, traffic sources must also reduce their sending rates when receiving marked packets. It is not clear how a traffic source's congestion control should respond to these marked packets to make the traffic TCP-friendly. Marked packets may be dropped or remarked when they are on their way to the destination or back to the source. These problems are difficult and require much further investigations. In contrast, the TCP trunking approach uses the well-established TCP congestion control to regulate the sending rate of a traffic flow, and is 100% TCP-friendly.

9. Conclusions

TCP trunking is a novel way of applying TCP congestion control to bandwidth management of aggregate traffic. It appears to be one of the few techniques that could provide dynamic congestion control for traffic aggregates. As the Internet usage continues to scale up, traffic aggregation becomes increasingly important, so does their bandwidth management tools such as TCP trunking.

Traditionally, TCP has been the dominant protocol that provides dynamic congestion control for individual flows between end hosts. The TCP trunking work of this paper has shown that TCP is also suited in providing dynamic congestion control for aggregate flows between network nodes.

References

[1] H.T. Kung and S.Y. Wang "TCP Trunking: Design, Implementation, and Performance", IEEE ICNP'99, Toronto, Canada, 1999.

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

[3] Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy, RFC1519, 1993

[4] K, Nichols, V. Jacobson, L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet," RFC 2638, July 1999.

[5] S. Blake et al., "An Architecture for Differentiated Services," RFC 2475, December 1998.

[6] R. Morris, "TCP Behavior with Many Flows," IEEE ICNP'97, Atlanta USA, 1997.

[7] D. O. Awduche, J. Malcolm, J. Agogbua, M. O'Dell and J. McManus, "Requirements for Traffic Engineering Over MPLS," RFC 2702, September 1999.

[8] R. Callon et al., "A Framework for Multiprotocol Label Switching," Internet draft, September 1999.

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

[10] S. Y. Wang and H.T. Kung, "A Simple Methodology for Constructing an Extensible and High-Fidelity TCP/IP Network Simulator," INFOCOM'99, New York, USA, 1999.

[11] FreeBSD web site at www.freebsd.org.

[12] D. Lin, and H. T. Kung, "TCP Fast Recovery Strategies: Analysis and Improvements," INFOCOM'98, San Francisco, USA, 1998.

[13] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," Transactions on Networking, Vol. 1, No. 4, August 1993.

[14] ATM Forum Traffic Management Specifications 4.0.

[15] Francois Baccelli and Dohy Hong, "TCP is Max-Plus Linear," ACM SIGCOMM'2000, Stockholm, Sweden, 2000.

[16] Eitan Altman, Kostia Avrachenkov, Chadi Barakat, "A Stochastic Model of TCP/IP with Stationary Random Losses," ACM SIGCOMM'2000, Stockholm, Sweden, 2000.

[17] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," ACM SIGCOMM'98, Vancouver, Canada, 1998.

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

[19] K. K. Ramakrishnan, and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP," RFC 2481, January 1999.

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