# TCP Tunnels: Avoiding Congestion Collapse

B. P. Lee, R. K. Balan,
L. Jacob, W. K. G. Seah, A. L. Ananda
*Centre for Internet Research (CIR)*
*School of Computing*
*National University of Singapore*
*3 Science Drive 2, Singapore 117543*
*{ leebp, rajeshkr, jacobl, wseah, ananda }@comp.nus.edu.sg*

## Abstract

This paper examines the attributes of TCP tunnels which are TCP circuits that carry IP packets and benefit from the congestion control mechanism of TCP/IP. The deployment of TCP tunnels reduces the many flows situation on the Internet to that of a few flows. TCP tunnels eliminate unnecessary packet loss in the core routers of the congested backbones which waste precious bandwidth leading to congestion collapse due to unresponsive UDP flows. We also highlight that the use of TCP tunnels can, in principle, help prevent certain forms of congestion collapse described by Floyd & Fall [FF98].

Using a testbed of ten Intel PCs running the Linux operating system and traffic generators simulating user applications, we explore: the benefits which TCP tunnels confer upon its payload of user IP traffic; the impact on the congestion within network backbones; and the protection that tunnels offer with respect to the various competing classes of traffic in terms of bandwidth allocation and reduced retransmissions.

The deployment of TCP tunnels on the Internet and the issues involved are also discussed and we conclude that with the recent RFC2309 recommendation of using Random Early Drop (RED) as the default packet-drop policy in Internet routers, coupled with the implementation of a pure tunnel environment on backbone networks makes the deployment of TCP tunnels a feasible endeavour worthy of further investigation.

***Keywords:*** *TCP tunnels, aggregation, quality of service (QoS), congestion collapse, queue management, flow back-pressure, RED routers*

## 1. Introduction

The proliferation of multimedia applications involving voice and video is straining the capacity of the Internet. The evolution and deployment of new network technologies involving the provision of bandwidth has continually fed the insatiable end user's appetite for even more bandwidth. Whenever a leap in network technology for provisioning bandwidth occurs, the end user demands for bandwidth threatens to consume the available capacity and stress the limits of technology. In addition, the increasing use of TCP-unfriendly and unresponsive flows compounds the problem of tackling congestion on the Internet.

A great deal of in-depth research has been carried out on the transport protocols of the Internet, primarily on the Transmission Control Protocol (TCP), scheduling algorithms and on the efficacy of packet-dropping policies such as Tail-Drop and Random Early Drop (RED) [FJ93].

A TCP tunnel is a TCP circuit which carries IP frames over the Internet. By layering user flows over this construct, we hope to benefit from the congestion control mechanism of TCP/IP, especially in segregating unresponsive flows from TCP-friendly flows. Kung & Wang [KW99] did related work on the properties of TCP trunks which are similar to TCP tunnels. In their paper, they demonstrate how TCP trunking protects web users from competing ftp traffic. They also examine how TCP trunking may be used to allow a site to control its offered load into a backbone network so that the site can assure some QoS for its packets over the backbone.

This paper studies the behaviour and properties of TCP tunnels in conjunction with RED routers and examines the benefits and disadvantages they confer on the IP traffic. We try to identify the network conditions and scenarios where the deployment of TCP tunnels would be beneficial to overall network performance. We also discuss how TCP tunnels can assist in avoiding certain forms of congestion collapse on the Internet as considered by Floyd & Fall [FF98].

We believe that the study of TCP tunnels would be most useful, considering the fact that commerce on the Internet has fueled significant interest and efforts in the deployment of Virtual Private Networks (VPNs) over the Internet, particularly in assisting resource sharing, work collaboration and meeting privacy concerns. Typically, VPNs are layered over the Internet and these VPNs are deployed using tunnels. These tunnels appear as layer-2 constructs to user applications.

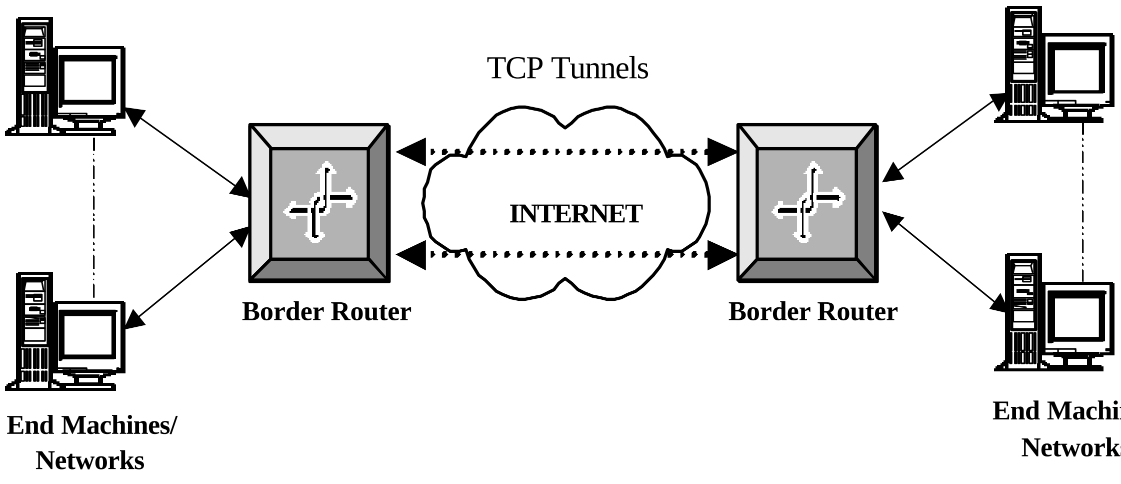

**Fig 1:** TCP Tunnels as transport mechanisms for IP flows over the Internet

We envision that one or more TCP tunnels can be installed between border routers which connect the LANs to the WANs or the Internet cloud (see Fig. 1). Later, we show that this has the effect of confining packet drops to the LAN side and helps to keep congestion low on core routers (also known as WAN routers). TCP tunnels can also be deployed by Internet Service Providers on point-to-point links to take advantage of attributes of TCP tunnels and offer better service. For instance, if traffic from interactive applications such as telnet or http arrive at a last hop bottleneck link and are liable to be dropped, TCP tunnels can be deployed over this link to offer protection from losses due to congestion.

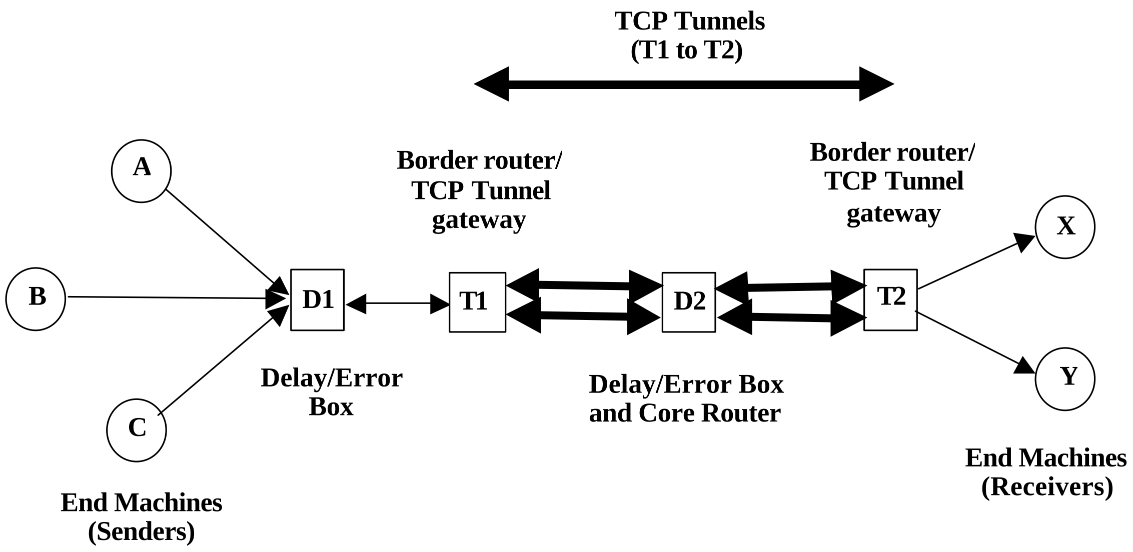
TCP Tunnels (T1 to T2)
A
B
C
D1
T1
D2
T2
X
Y
Border router/ TCP Tunnel gateway
Border router/ TCP Tunnel gateway
Delay/Error Box
Delay/Error Box and Core Router
End Machines (Senders)
End Machines (Receivers)

**Fig 2:** Testbed

The experiments that are described in this paper make reference to the testbed depicted in Fig 2. RedHat 6.0 with Linux Kernel version 2.2.12 are installed on our PCs, all of which are Intel Celeron 300A with 128MB of memory each. The network cards we use are the Intel Ether Express Pro 10/100 (100Mbps). The delay/error box is able to buffer and delay packets to simulate various latencies, and selectively drop and corrupt packets. We use *ttcp* [MS85] and *iperf* [GW] to generate bulk TCP traffic and *Surge* [BC98] for generating http traffic. *tcpdump* is used to collect the traffic and the traffic analysis is carried out using *tcptrace*. RED and Tail-Drop are implemented using the Linux Traffic Control Extensions [Al99].

Our TCP tunnel implementation is written in C. On the TCP tunnel transmitter side, it receives IP packets from the incoming interface using *libpcap* routines and transmits these packets over one or more TCP connections. We configured a Tail-Drop queue of 1000 packets leading into each TCP tunnel. The TCP tunnel receiver reads the IP frames from the various TCP connections and writes them out to the outgoing interface using raw sockets.

The rest of the paper is as organised as follows: Section 2 will cover the protection that tunnels can offer to TCP-friendly flows in various scenarios. The effects of TCP tunnels and RED on router queue lengths are discussed in Section 3. Section 4 elaborates on the congestion control property of TCP tunnels and explains how this attribute gives rise to back-pressure which pushes congestion to edge routers from the core routers. The types of congestion collapse that may arise on the Internet and the congestion control property of tunnels that can help to prevent some forms of congestion collapse are discussed in Section 5. Section 6 covers the possible application of tunnels in the provisioning of Proportional Differentiated Services over the Internet and Section 7 touches on the deployment issues for TCP tunnels.

## 2 Protection of TCP-friendly flows

UDP traffic are not TCP-friendly [FF98] and hence, do not respond to packet drops which signal congestion. This aggressive behaviour degrades and even shuts out TCP flows, both bulk transfers (ftp) and interactive applications (http and telnet), and thus preventing them from obtaining their fair share of bandwidth over a congested link. In this section, we examine how tunnels can be deployed to isolate different types of traffic from one another and protect interactive user applications from aggressive flows.

## 2.1 Protection of TCP bulk flows from unresponsive UDP flows

In this section, we show how TCP flows carrying bulk traffic are protected from aggressive UDP flows over a congested link. We assume that a number of tunnels are deployed over the link, each carrying a specific kind of traffic, and that they compete with each other for bandwidth. All traffic flows, UDP or TCP, traversing the congested link has to go through at least one of the tunnels. As a result, UDP flows are prevented from clogging the congested link as the tunnels carrying the UDP flows will suffer packet drops and will be forced to throttle their sending rates. Here, we see that the UDP flows are endowed with a crude form of end-to-end congestion control by virtue of being wrapped in TCP tunnels.

We demonstrate this by sending a 10 Mbps UDP flow (CBR data) from A to X and 100 TCP flows (representing 100 concurrent file transfers of 512KB each) from C to Y (see Fig. 2). The end-to-end Round Trip Time (RTT) is 10ms and all the links are 10 Mbps except the link D2-T2 which is 1 Mbps. The UDP flow is *very aggressive and unresponsive*. The MTU is 1500 bytes for both sources of traffic. The bottleneck link of 1 Mbps is fed by a RED queue (with parameters $min_{th}$=20KB, $max_{th}$=520KB, $max_{p}$=0.02). The run is considered complete when all file transfers are complete. We use the above scenario to run two experiments; one without TCP tunnels and one with TCP tunnels (tunnels deployed between T1-T2). The results of this experiment are shown in Table 1.

Without the tunnels, we observe that the UDP flow consumes most of the bandwidth, leaving each TCP flow with a mean throughput of 1.18 Kbps. Things are different when we use TCP tunnels to isolate both types of traffic. Here, the mean throughput of the TCP flows increases to 2.85 Kbps. The UDP flow is carried over one tunnel and the TCP traffic is carried over a separate tunnel. Both tunnels are fed by their own Tail-Drop queues, each of which has a capacity of 1000 packets. Should the TCP tunnels be served by a single shared queue, the greedy non-responsive UDP flow can just hog this queue and shut out other flows.

When we run the UDP flow against a few TCP flows (less than five), it is observed that the TCP flows are totally shut out and could not complete their transfers. The situation improves when we increase the number of competing TCP flows to a significant figure, say, 100 as given above. During the experiments, it is noticed that the aggressive nature of the TCP tunnels is determined by its payload. A tunnel carrying UDP traffic is more aggressive when pitted against another tunnel which carries a few TCP flows. Presumably, when either tunnel suffers a packet drop, it reduces its congestion window and propagates this effect to its payload. The TCP flows take time to ramp up their transmission rates, unlike the UDP flow whose transmission rate is unaffected.

| **Per TCP flow statistics** | **Without TCP Tunnel** | **With TCP Tunnel** | **% Improvement** *(+ means in TCP Tunnel's favour, - means unfavourable)* |
|---|---|---|---|
| RTT average (ms) | 4133.49 | 15821.83 | (-) 282.77 |
| RTT minimum (ms) | 1292.33 | 1200.14 | (+) 7.13 |
| RTT maximum (ms) | 4563.39 | 17103.36 | (-) 274.79 |
| Number of RTT samples | 83.16 | 168.50 | (+) 102.62 |
| Retransmitted bytes | 385783.90 | 57773.36 | (+) 567.75 |
| Retransmitted packets | 266.78 | 39.93 | (+) 568.12 |
| Retransmitted data segments over total segments | 28.10 | 6.10 | (+) 360.51 |
| Data segments sent | 624.45 | 397.47 | (+) 57.11 |
| Acknowledgement segments sent | 324.27 | 247.32 | (+) 31.11 |
| Data segments over total segments | 65.82 | 61.66 | (+) 6.74 |
| Total Segments | 948.72 | 644.79 | (+) 47.14 |
| Average throughput (Kbits/s) | 1.18 | 2.85 | (+) 141.53 |

**Table 1:** Per TCP flow statistics obtained from tcptrace analysis program

From Table 1, we see that TCP tunnels reduce the overall amount of traffic sent. The amount of retransmissions per connection are reduced by over 500% as the tunnel provides reliability in the face of congestion. Packet loss from the tunnel due to heavy congestion has minimum impact on TCP flows and precious link bandwidth is not wasted on retransmissions.

However, the tunnels degrade the TCP RTT estimations badly (up to 280%!). Note that when the TCP tunnel is forced to throttle its sending rate due to congestion at the bottleneck link, packets get queued up at the tunnel ingress. The sources which are oblivious to the congestion continue to send at an increased rate until the queues at the tunnel ingress overflow. Only lost packets due to this overflow are retransmitted by the sender. It would be interesting to examine the effect that the sizes of the queues which feed the TCP tunnels have on the RTTs of TCP flows.

## 2.2 Protection of Interactive Traffic over Congested Links

Interactive end user applications such as http often suffer long delays and even timeouts over WAN links as they usually traverse over a bottleneck link and compete with other types of traffic. Usually, the bottleneck link is a single hop of low latency. The http connections suffer packet drops at this link which leads to inevitable timeouts.

An Internet Service Provider (ISP) may choose to deploy TCP tunnels over this short latency bottleneck link to isolate the interactive traffic from other aggressive traffic. The TCP tunnels will also handle retransmissions over this congested link and minimise retransmissions by the end user applications and possible timeouts.

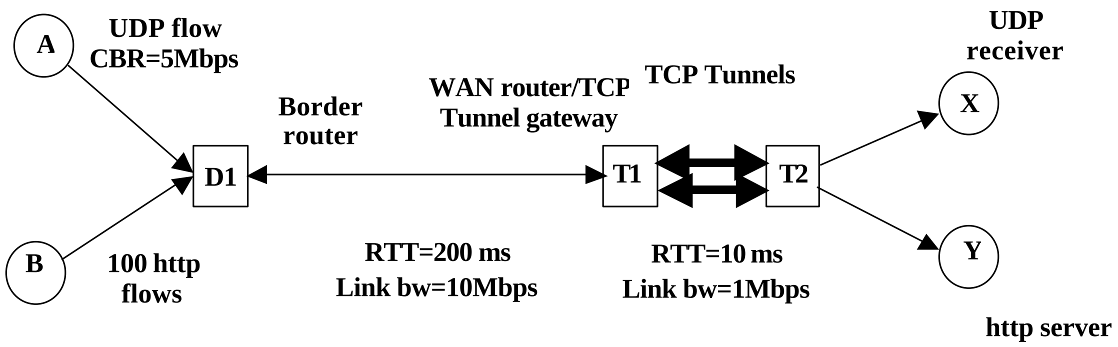

**Fig 3**: Tunnel offering reliability for interactive flows over congested bottleneck link.

To demonstrate this scenario, we set up the configuration shown in Fig. 3 using the testbed. T1 and T2 are the tunnel end-points and the TCP tunnels are layered over this 1 Mbps bottleneck. Note that this bottleneck link is limited to 1 Mbps in the forward direction (towards X and Y) and the reverse path is of bandwidth 10 Mbps. The experimental run lasts 300 seconds. The 100 http connections from B to Y are started first and the UDP flow of 5 Mbps, from A to X, is started 10 seconds later. Note that each one of the traffic classes is carried by different tunnels.

| Per http flow statistics | Without TCP Tunnels | With TCP tunnels |
|---|---|---|
| Packets in Forward direction (http requests) | 13.64 | 25.48 |
| Packets in Reverse direction (http data) | 13.11 | 30.00 |
| Total Packets transferred | 27.75 | 55.48 |
| Forward/Reverse Aggregated Throughput (Bps) | 1260 / 64247 | 1582 / 126325 |
| Successful / Total http connections made | 45 / 208 (21.6%) | 290 / 624 (46.5%) |

**Table 2:** http flow statistics

From Table 2, we see that the http connections protected by the TCP tunnels managed to achieve higher average packet and aggregated throughput. More importantly, for the same experimental run and duration, a greater number of successful and complete http connections were made with the help of the tunnels. Without the protection of the tunnels, numerous http connections suffered timeouts and were unable to complete their transfers within the same interval (i.e. 300 seconds).

## 2.3 Protection from Fragmentation

The performance of user connections on the Internet is limited by the *Path MTU* [RFC1191]. This refers to the largest size that an IP datagram can take which permits it to traverse the network without undergoing fragmentation.

In [KM87], it was argued that fragmentation is inefficient, the loss of any fragment degrades performance due to retransmission and that efficient reassembly of fragments is hard. [KM87] concludes that the disadvantages of fragmentation far outweigh its benefits and fragmentation should be avoided as far as possible.

A TCP tunnel behaves like a level 2 (datalink) circuit and hides the different MTUs of the underlying networks and links it spans over the Internet. The user flows (in the form of IP packets which is the tunnel's payload) are oblivious to these different MTUs and hence are free to use its own value, presumably the largest (limited by the sender's interface's MTU setting) to maximise throughput. If the use of larger MTUs is permitted, each packet carries less header overhead and this allows TCP connections to ramp up their congestion windows faster.

To get an idea of the relationship between throughput and MTU sizes, we vary the Path MTUs as shown in Table 3 (1st column) and measure the resulting throughput of a TCP. Our first scenario is without TCP tunnels while the second is with TCP tunnels. In both scenarios, a round-trip delay of 211ms was set between A to X (Fig. 2) which were the end machines used to create the TCP connection used in the two scenarios. The different MTUs were set on the path between T1 to T2 (Fig. 2) which has a round-trip delay of 10ms. For the scenario with TCP tunnels, the tunnels were deployed between T1 and T2 to encompass the path with the different MTU.  Ten runs were conducted per MTU setting. Note that the traffic source's MTU setting is fixed at 1500 bytes for all runs.

| **Path MTU (T1-T2) (bytes)** | **Mean Throughput of 10 runs (Mbps) No TCP Tunnel** | **Mean Throughput of 10 runs (Mbps)** |
|---|---|---|
| 1500 | 7.40 | 7.40 (+0%) |
| 1006 | 7.10 | 7.40 (+4%) |
| 576 | 6.50 | 7.08 (+9%) |
| 296 | 5.00 | 6.70 (+34%) |

**Table 3:** Mean connection throughput for various MTUs

From Table 3, it can be seen that the tunnels improved the throughput of the end-to-end connection for lower MTU values across T1 to T2. This was because the tunnels confined the fragmentation caused by the lower MTU path to the tunnel only and did not allow the fragmentation to be propagated to the rest of the network.

Similarly, UDP flows would experience higher throughput by sending datagrams with large MTUs over links with smaller MTUs via tunnels. Finally, the use of Path MTU Discovery does not obviate the relevance of TCP tunnels. With Path MTU Discovery, the sender still has to send IP frames no larger than the smallest MTU along the path to avoid fragmentation. TCP tunnels allow senders to send packets at the largest possible MTU (limited by the MTU settings on their interfaces) despite the smaller MTUs along the path. The presence of these smaller MTUs are totally transparent to the sender and the tunnels protect the sender from fragmentation caused by these smaller MTUs.

## 3. Effects of Tunnels on Router Queue Lengths

Tail-Drop has been adopted widely on routers as the default packet-dropping policy. However, this policy causes many TCP flows to throttle their sending rates at the same time, when the queue overflows. These flows would then ramp up their transmission rates until the queue overflows again. This synchronisation in flow behaviour causes burst effects and may reduce average flow throughput and link utilisation. In addition, the average queue length of Tail-Drop routers tend to be high and this introduces longer packet delays.

RED is designed to penalise TCP-friendly flows which consume more than their fair share of bandwidth by randomly dropping packets from the flows when the average queue size exceeds the minimum threshold. RED eliminates the synchronisation and burst effects seen in Tail-Drop. RED, however, has no effect on UDP or unresponsive flows which do not respond to packet drops. The flows that suffer from RED-induced packet drops would be the good guys, the TCP-friendly flows.

To underscore our point on the queue lengths of routers adopting RED and Tail-Drop packet-dropping policies, we run experiments with an increasing number of TCP flows from a 10 Mbps source which traverse a bottleneck link of 1 Mbps. The runs are repeated for RED0.02, RED0.1 (RED with maximum drop probability, $max_p$, of 0.02 and 0.1, respectively) and Tail-Drop. The minimum and maximum thresholds for the RED queues, $min_{th}$ and $max_{th}$, are 20 KB and 520 KB, respectively. The sizes of both RED and Tail-Drop queues are 520 KB.

Fig. 6 and 7 summarise the results. Note that the various queues' hard limits of 520KB translates to a maximum queue occupancy of 354 packets of MTU 1500 bytes each. The confidence interval for Tail-Drop in Fig. 6 extends past this limit as the queues contain some small amount of spurious traffic due to Network Time Protocol (NTP) synchronisation between the machines on our testbed.

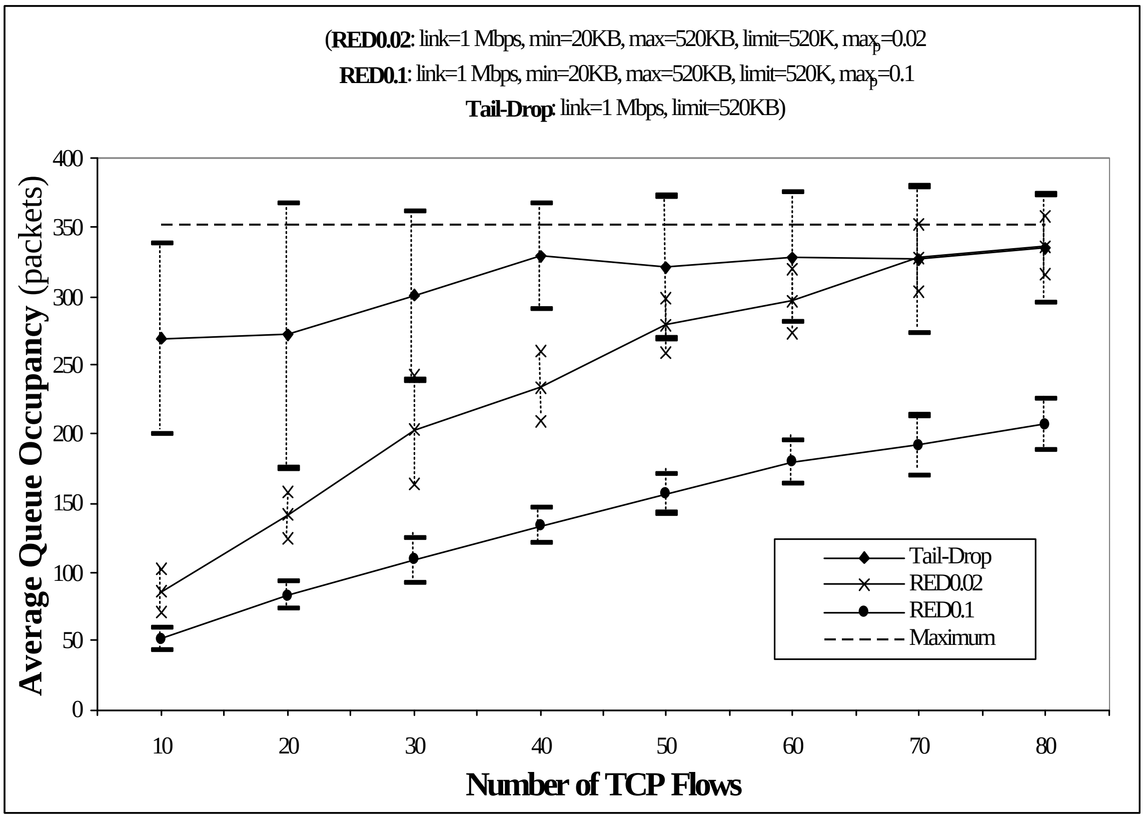

**Fig 6:** Average Queue Occupancy for RED & Tail-Drop (No TCP Tunnels)

From Fig. 6, we can see that the mean queue length for Tail-Drop is high and the instantaneous queue lengths are highly variable. The RED policies manage to keep the average queue lengths low compared to Tail-Drop. The average queue length under RED scales with the number of TCP flows. RED0.1 demonstrates that increasing the maximum packet drop probability has the effect of reducing the average queue length. Fig. 7 shows that the instantaneous queue length is close to overflowing the queue for 80 connections.

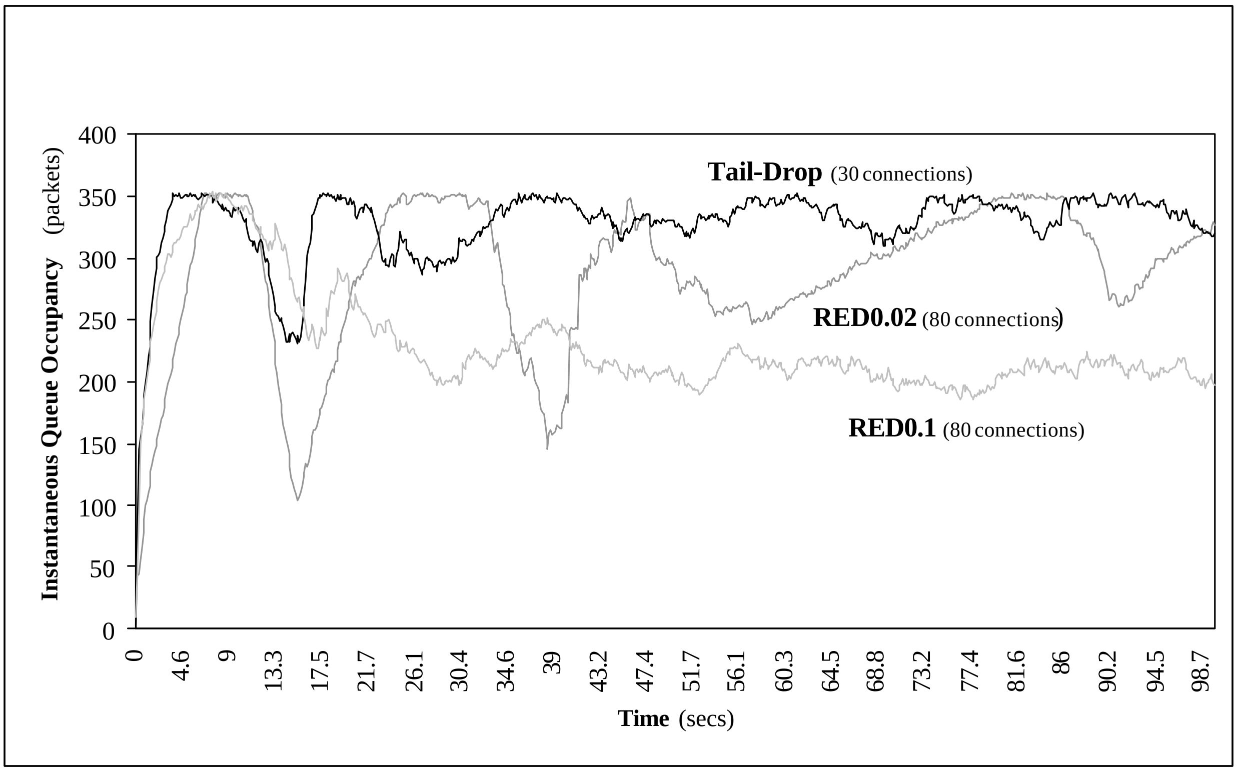

**Fig 7:** Instantaneous queue occupancy for RED & Tail-Drop (No TCP Tunnels)

Although RED is able to keep the average queue length low for a small number of flows, its behaviour seems to approximate that of Tail-Drop when the number of flows that it has to deal with becomes large.

The point we are emphasising is that the deployment of TCP tunnels reduces the number of flows within the core routers. The core routers only need to manage the few TCP tunnel flows. Therefore, as RED is being increasingly adopted by routers as the default packet-dropping policy (recommended by RFC2309), we feel that TCP tunnels in conjunction with RED make an important contribution towards controlling congestion on the Internet.

## 4. Congestion Control and Back-Pressure Effects of TCP Tunnels

[FF97] discusses the problems of unresponsive flows and the danger of congestion collapse of the Internet while [PSG] discusses the use of back pressure in moving congestion away from the core. The rates of UDP flows are not limited by congestion control and such aggressive flows only starve TCP-friendly flows of their rightful share of bandwidth. In addition, UDP flows typically exacerbate congestion problems and waste precious bandwidth when their packets are dropped during the onset of congestion.

TCP tunnels provide a possible solution out of this dilemma. Tunnels could be used to wrap UDP flows at edge routers. The tunnels carry the UDP flows over the Internet and are then subjected to the ever-changing conditions of the network. When congestion occurs, the packet drops force the tunnels to throttle their sending rates. The advantage here is that the UDP flows are now TCP-friendly (so to speak) and sensitive to packet drops from the perspective of the core routers. This is of particular relevance when RED is deployed on routers as the default packet dropping policy. In short, packet losses in the congested core routers are now pushed to edge routers.

The main objection to this idea is that the tunnel would then introduce unnecessary delay and jitter variances to delay-sensitive UDP flows, on the order of several RTTs as the tunnel attempts retransmission (the tunnel being a TCP circuit offering reliable delivery of data). This will be discussed further in Section 8.

A huge advantage of carrying user flows over TCP tunnels and deploying tunnels in WAN environments is that congestion will be pushed to the edge routers from the core routers (similar to the back-pressure effect in ATM networks). The tunnels would seek out their fair share of bandwidth as network conditions change and packets are dropped from the tunnels. In response to the packet drops, the tunnels would throttle their sending rates and drop packets of user flows at the ingress tunnel gateways. The packet drops experienced by the tunnels within the backbone of the WAN network (at the core routers) would be low, since these losses are from the TCP tunnels themselves which would throttle their transmission rates in response. Contrast this with the situation where huge numbers of user flows enter the core without the deployment of tunnels. The link efficiency and utilization of the core is expected to be higher when tunnels are deployed. Notice that the deployment of tunnels are akin to the Virtual Paths of ATM networks.

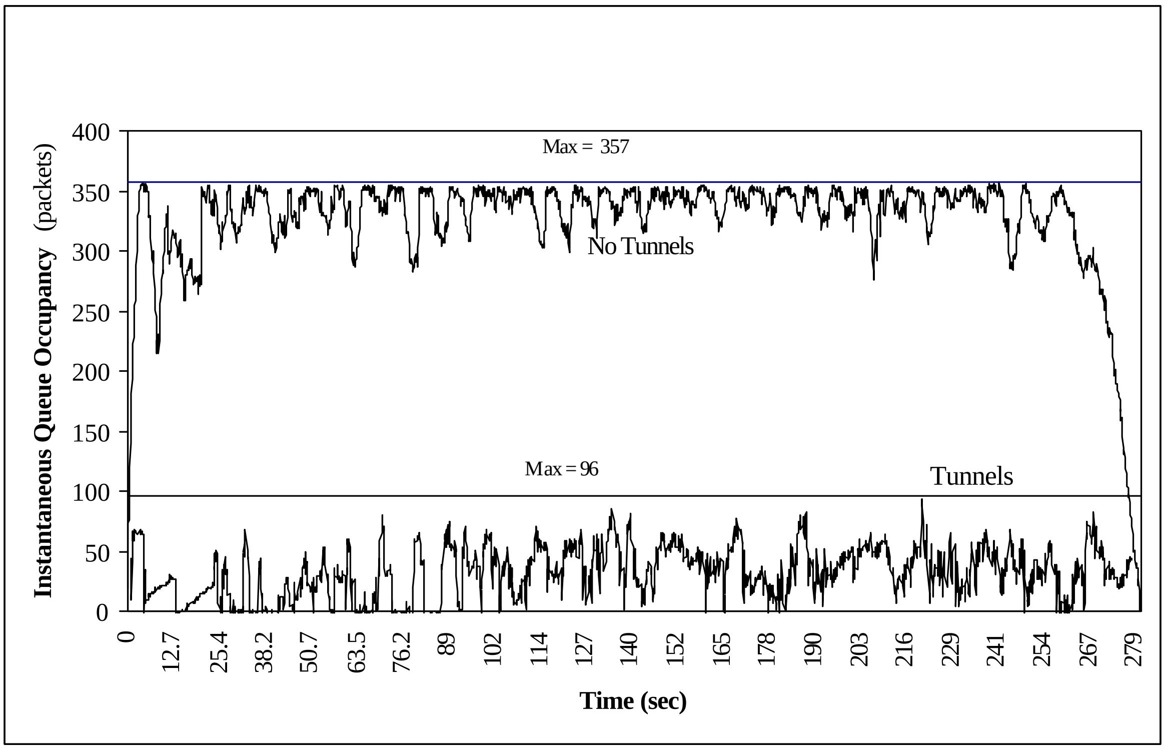

**Fig 8:** Instantaneous Queue occupancies for core router with and without TCP tunnels

To demonstrate the back-pressure effect, we run 50 bulk TCP flows on each of the machines A, B, and C to destinations X and Y (see Fig. 2). We also monitor the queue lengths at the core router (D2), and at the TCP tunnel router (T1). Each one of the three tunnels is fed by a separate Tail-Drop queue (maximum size of 1000 packets), and RED is adopted as the packet-drop mechanism on the core router, with the following settings: $min_{th}$=20KB, $max_{th}$=520KB, limit=520KB, $max_p$=0.02.

From Fig. 8, we can see that the queue length of the core router is high. It reaches the limit of the RED queue. When TCP tunnels are deployed, the core router experiences a much lower queue length (peak queue length = 96). Fig. 9 displays the instantaneous queue lengths of the three Tail-Drop queues of the tunnel router along with that of the RED queue of the core router. Notice that the queue lengths at the tunnel router are much higher than the queue length at the core router, thus indicating that the congestion has been moved from the core to the edge.

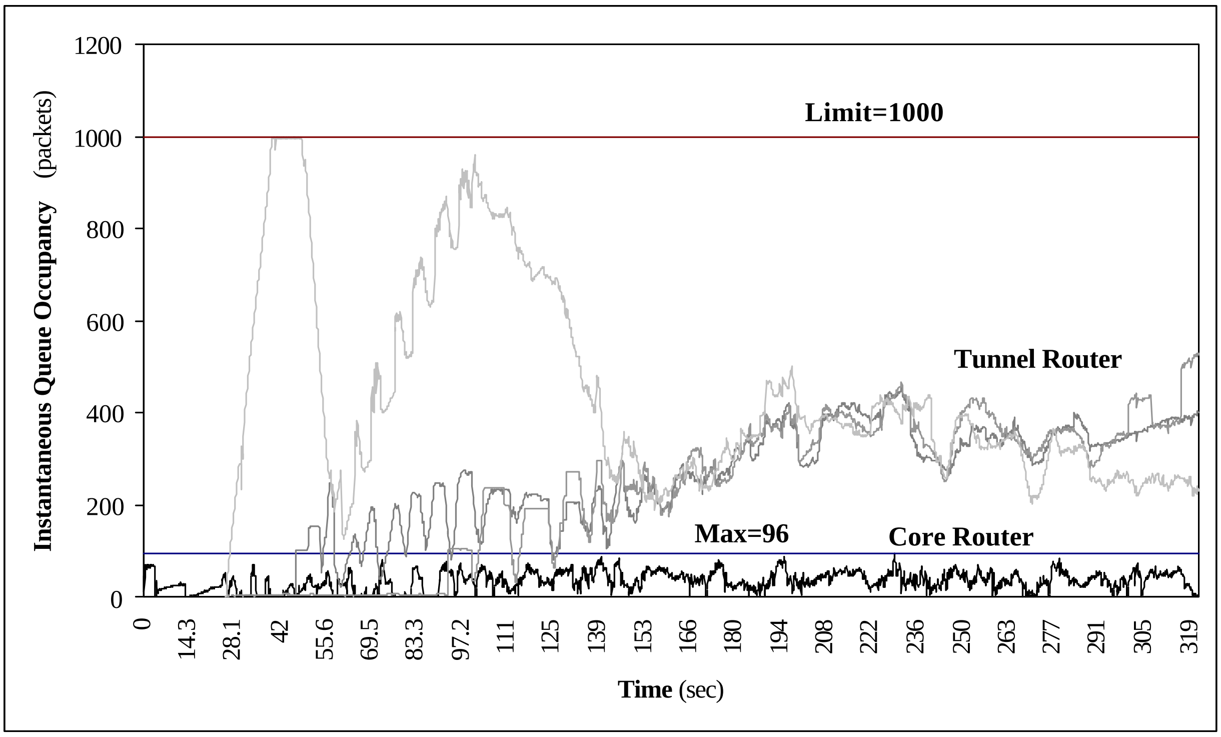

**Fig 9:** Instantaneous Queue occupancies for core and tunnel routers using TCP tunnels

| **Number TCP flows in the core** | **Without Tunnels** | **With Tunnels** |
|---|---|---|
| 60 | 3.12% | 0.15% |
| 120 | 7.27% | 0.14% |
| 180 | 11.80% | 0.15% |
| 240 | 19.41% | 0.16% |

**Table 4:** Percentage of dropped packets in the core router

Using the above experimental setup, we run experiments to determine the percentage of dropped packets within the core router as the TCP load is steadily increased. From Table 4, it can be seen that tunnels significantly reduce and even stabilise the percentage of dropped packets in the core router even when the number of TCP flows increases.

## 5. Avoiding Congestion Collapse

[FF98] discusses at length the various forms of congestion collapse.

***Classical congestion collapse*** [RFC896] occurs when the TCP connections unnecessarily retransmit packets which are still in transit or have been received, during heavy network load. The network reaches saturation and continues to operate under degraded conditions where the throughput is much lower than normal. This form of congestion collapse has been avoided by the adoption of congestion control principles put forward by Van Jacobson [RFC2581]. TCP tunnels can help to avoid this form of congestion collapse as the tunnels reduce the number of unnecessary retransmissions as established in Section 2.1.

***Congestion collapse from undelivered packets*** results when precious bandwidth is consumed and wasted by packets which are dropped before reaching their final destination. Open-loop applications such as voice, video and music applications probably will be the number one contributors to this type of congestion collapse as such applications do not utilise congestion control. The tunnel guarantees delivery of the permitted amount of open-loop user traffic to their destination and no user packets are lost within the tunnel. The use of TCP tunnels pushes the congestion from the core routers to the edge routers which causes packet drops to occur at the entry points, rather than further down the network.

***Fragmentation-based congestion collapse*** arises when packets are dropped when their fragments cannot be reassembled into complete packets as some fragments are lost, possible due to congestion experienced en route to the receivers. The continued congestion by packets which will probably be dropped by the receivers wastes bandwidth and results in reduced throughput. Early Packet Discard and Path MTU Discovery were highlighted [FF98] as two mechanisms useful in combating this form of congestion collapse. We claim TCP tunnelling could be one more of such mechanisms as it confines the fragmentation to the tunnel alone and permits the use of the largest possible MTU for a TCP connection.

***Congestion collapse from increased control traffic*** manifests when increasing load is accompanied by increased control traffic such as route updates, multicast join requests etc. TCP tunnels can prevent control traffic and their related user traffic from congesting the busy backbones by remapping the tunnels carrying those types of traffic to relatively uncongested paths in the network.

## 6. Providing Quality of Service Guarantees

It was proposed by [KW99] that TCP Trunks could be used as a means of assuring Quality of Service (QoS) by exploiting the elastic property of the TCP circuit (tunnel). TCP's congestion and flow control would dynamically probe for available bandwidth when the load conditions of the network change. [KW99] also suggested that the tunnel and its payload (user flows) could be allocated a guaranteed minimum bandwidth and be given more bandwidth when spare link capacity was available. Few other details were offered on how this was possible and it seems that deploying TCP tunnels is tricky. However, the deployment of TCP tunnels and its benefits would be more apparent if we attempt to qualify its applications to certain scenarios.

Using TCP tunnels to guarantee absolute QoS is not feasible without some form of route pinning. The reservation of resources necessary to support user services have to be instantiated at every hop of the network, even for tunnels, either manually or through some form of signalling (e.g. RSVP). This alone is a huge obstacle in deploying QoS-enabled tunnels on the Internet. However, the aggregation of user flows into tunnels offering different classes of service naturally complements any network QoS system (over WAN) as it contributes to scalability and eases the problem of flow identification and classification.

A more feasible approach in adapting TCP tunnels for QoS over the Internet would be to offer *Proportional Differentiated Services* (proportional QoS) vis-a-vis *Absolute Differentiated Services* (absolute QoS) [DR][DSR]. Consider capacity differentiation as an example. A CBQ system could offer traffic class A 30% of the available bandwidth and traffic class B the remaining portion (70%) over the Internet. However, there is no minimum bandwidth guarantee that we can extract from the Internet without the cooperation of hops along the route and the bandwidth available changes from time to time. But, tunnels carrying both classes of traffic are always probing and utilising their fair share of the total bandwidth and we think it is not inconceivable that a CBQ system, applying some form of *weighted fair queueing*, would be able to deliver the flow of both traffic classes over the Internet in the promised proportions.

## 7. Deployment of TCP Tunnels

The tunnels are most suitable for deployment at border routers as the greatest mismatch of link bandwidths occur at the edges of the Internet cloud. This mismatch is still true today as LANs now offer Gigabit speeds while WANs struggle to keep up, managing at most Megabit capacities. As traffic transit at such boundaries, TCP tunnels can be suitably positioned there to manage various classes of traffic and yet drop excess traffic closest to the sources, rather than allowing the unwanted traffic to be discarded in the core backbones and consuming precious bandwidth in the process.

The Internet has managed to meet the needs of its communities and users simply because its concept and deployment have been rigorously kept simple and scalable. *Aggregation* using tunnels reduces the problem of *many flows* to that of a *few flows*. This allows Internet Service Providers (ISPs) to identify and provide for classes of services in a scalable manner. The adoption of tunnels by ISPs would be invisible to user applications and these ISPs could provide incentives to encourage use of tunnels by favouring tunnels in their routers through the allocation of traffic priorities and judicious application of preferred scheduling and queueing policies.

Tunnels lend themselves easily to traffic management and provisioning through MPLS and RSVP. Tunnel flows can be uniquely identified by their IP addresses and ports, i.e. <source IP, destination IP, source TCP port, destination TCP port>. However, it is critical to ensure that the deployment of tunnels do not alter the behaviour of their payload, especially those of TCP flows.

## 8. Summary & Future Research

We have demonstrated that TCP tunnels offer TCP-friendly flows protection against aggressive and unresponsive flows. In addition, we find that TCP tunnels hide the different MTUs (typically smaller) of the underlying network links from the user flows and allow these flows to send traffic at the largest possible MTUs (dictated by their end network interfaces).

We also find that aggregation of user flows into TCP tunnels resulted in lower average queue lengths and less packet drops in the core routers. The lower number of packet drops indicate that less bandwidth is being wasted in the core backbone while the lower average queue lengths indicate that the core routers can be provisioned with less memory resources for handling tunnel flows. In other words, tunnels allow the routers to handle even more flows than otherwise possible, with the same amount of memory resources.

By carrying greedy flows over TCP tunnels, the core routers are able to control the amount of traffic from unresponsive flows such as UDP (from multimedia applications) in the event of congestion within the network backbones. Therefore, precious limited bandwidth is not hogged by unresponsive flows which will be wasted when their packets are dropped by the core routers when congestion occurs.

Aggregating flows into a TCP tunnel may also *introduce synchronisation and burst effects*. If the network is in a steady state, the bandwidth and round-trip times offered by the tunnel is also steady. However, changes in network conditions may cause wild swings in tunnel behaviour and transmit such effects to the user flows which it transports. This will probably impact delay-sensitive traffic badly, especially if the tunnel attempts data recovery over a long latency link. One solution to this could be to design a TCP-like protocol offering *the flow and congestion control characteristics of TCP/IP* and yet, *without its (TCP's) reliable delivery of data*. Such a protocol would not be unlike RTP complemented by flow and congestion control principles of TCP. More importantly, this would be of greater relevance as a transport protocol for end-to-end delay sensitive applications.

## References

[Al99] W. Almesberger, *"Linux Network Traffic Control – Implementation Overview"*, April 1999, ftp://lrcftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps

[BC98] P. Barford and M. Crovella, *"Generating Representative Web Workloads for Network and Server Performance Evaluation"*, Proc. ACM SIGMETRICS '98, June 1998

[DJ91] Peter Danzig, Sugih Jamin, "tcplib: A Library of TCP/IP Traffic characteristics", University of Southern California, October 1991.

[DR] C. Dovrolis, P. Ramanathan, "A Case for Relative Differentiated Services and the Proportional Differentiated Model"

[DSR] C. Dovrolis, D. Stiliadis, P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling"

[FF97] S. Floyd, K. Fall, "Router Mechanisms to Support End-to-End Congestion Control", 1997

[FF98] S. Floyd, K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet", 1998

[FJ93] S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", August 1993 IEEE/ACM Transactions on Networking

[GW] M. Gates, A. Warshavsky, "Iperf version 1.1", http://www.east.isi.edu/~tgibbons/iperf/index.html

[KM87] C. A. Kent, J. C. Mogul, "Fragmentation Considered Harmful", Dec 1987, Digital Western Research Laboratory

[KW99] H. T. Kung and S. Y. Wang, "TCP Trunking: Design, Implementation and Performance", Proc. 7th International Conference on Network Protocols (ICNP'99), October 1999

[MS85] M. Muuss and T. Slattery, *"ttcp: Test TCP"*, US Army Ballistics Research Lab (BRL), Nov 1985. Enhanced version by Silicon Graphics Incorporated. October 1991, ftp://ftp.sgi.com/sgi/src/ttcp

[PSG] Carlos M. Pazos, Juan C. Sanchez Agrelo, Mario Gerla, "Using Back-Pressure to Improve TCP Performance with Many Flows"

[RFC896] J. Nagle, "Congestion Control in IP/TCP Internetworks", RFC896

[RFC1191] J. Mogul, S. Deering, "Path MTU Discovery", RFC1191

[RFC2309] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, "Promoting the Use of End-to-End Congestion Control in the Internet", RFC2309

[RFC2581] M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC2581