Implicit Admission Control

Richard Mortier, *Student Member, IEEE*, Ian Pratt, Christopher Clark, and Simon Crosby, *Member, IEEE*

***Abstract*—Internet protocols currently use packet-level mechanisms to detect and react to congestion. Although these controls are essential to ensure fair sharing of the available resource between multiple flows, in some cases they are insufficient to ensure overall network stability. We believe that it is also necessary to take account of higher level concepts, such as connections, flows, and sessions when controlling network congestion. This becomes of increasing importance as more real-time traffic is carried on the Internet, since this traffic is less elastic in nature than traditional web traffic. We argue that, in order to achieve better utility of the network as a whole, higher level congestion controls are required. By way of example, we present a simple connection admission control (CAC) scheme which can significantly improve overall performance.**

This paper discusses our motivation for the use of admission control in the Internet, focusing specifically on control for TCP flows. The technique is not TCP specific, and can be applied to any type of flow in a modern IP infrastructure. Simulation results are used to show that it can drastically improve the performance of TCP over bottleneck links. We go on to describe an implementation of our algorithm for a router running the Linux 2.2.9 operating system. We show that by giving routers at bottlenecks the ability to intelligently deny admission to TCP connections, the goodput of existing connections can be significantly increased. Furthermore, the fairness of the resource allocation achieved by TCP is improved.

***Index Terms*—Admission control, Internet, quality of service, transmission control protocol (TCP).**

I. Introduction

CONGESTION is widespread in today's Internet. It causes packet loss and excess delay, leading to retransmission of data for reliable services, and degradation in quality for real-time services. Due to the dynamic and bursty nature of IP traffic, many schemes have been proposed for alleviating network congestion. They have generally relied on end-system based detection of, and reaction to, congestion solely at the packet level. At the other extreme, traditional connection-orientated networks, such as the public switched telephone network (PSTN), in which each connection consumes unit resource, and latterly asynchronous transfer mode (ATM) networks, in which each connection's resource requirement is specified using a generalized traffic specification, implement *connection admission control (CAC)* in order to ensure that per-connection performance requirements are met. CAC addresses congestion at the connection level by requiring that each connection request admission to the network, allowing the network to decide to *accept* or *reject* it. In this way the network can guarantee performance per connection.

Current Internet traffic consists largely of transmission control protocol (TCP) flows, which are of a connection-orientated nature, and elastic in their resource requirements. That is, TCP flows can operate under a variety of network conditions, and in particular, can perform useful work at a variety of bandwidths. However, streaming multimedia flows are becoming more prevelant, a prime example being the real time protocol (RTP) [1] which typically uses the user datagram protocol (UDP) for transport. Although these protocols are often not explicitly connection-orientated in nature at the transport layer, they may be considered so at a higher, session layer. They are usually much more inelastic in their resource requirements than protocols such as TCP—they have a narrower bandwidth operating region, requiring a minimal level of service to perform useful work. Both these types of flow may be considered single network transactions requiring a minimal resource in order for adequate user utility to be achieved. For example, most interactive uses of the Internet, such as downloading a web page, require a latency bound corresponding to a minimum useful bandwidth. If this bandwidth is not delivered, the latency bound will not be met, and the user is likely to either give up, or attempt to restart the download. It is only feasible to provide minimal guarantees of flow-level performance if the network performs admission control of some type, not solely at the level of individual packets. In this paper we deal with the application of admission control specifically to TCP flows, the case where one might expect least benefit. We believe that the technique of implicit admission control is applicable to other types of traffic, and are attempting to address this and the more general session-level admission control problem in ongoing work.

A. The Transmission Control Protocol

TCP is designed to provide flow-control and reliable transmission on top of the connectionless, unreliable Internet protocol (IP). Congestion occurs due to contention for limited network resources, typically buffer space or transmit bandwidth. If it is not detected and prevented, then *congestion collapse* may occur; this is where the network, or some subset of the network, is loaded to such a level that *goodput*—the throughput of data, disregarding retransmissions—falls to negligible levels. Following the rapid increase in the use of TCP and enormous changes in the topology and size of the Internet, a succession of *congestion control* mechanisms have been proposed and implemented in TCP.

These began with Jacobson's "slow start" and "congestion avoidance" schemes [2], and include the TCP varieties known as Tahoe, Reno, SACK, and Vegas [3], [4]. Most are based on the idea that when the TCP sender notices congestion, it will

Manuscript received October 15, 1999; revised April 15, 2000. The work of R. Mortier was supported by an EPSRC CASE award in collaboration with BT.

R. Mortier and I. Pratt are with the Systems Research Group, Cambridge University Computer Laboratory, New Museums Site, Cambridge CB2 3QG U.K. (e-mail: richard.mortier@cl.cam.ac.uk; ian.pratt@cl.cam.ac.uk).

C. Clark and S. Crosby are with CPlane Inc., El Camino Real, Los Altos, CA 94022 USA (e-mail: christopher@cplane.com; simon@cplane.com).

Publisher Item Identifier S 0733-8716(00)09232-5.

multiplicatively decrease its transmission rate (*back off*). Congestion is traditionally detected through loss of a packet, but alternatives where packets are merely marked according to a variety of policies are under consideration [5], [6]. Subsequently, the connection will linearly increase its transmission rate, until another loss event is detected. This gives TCP its characteristic sawtooth transmission pattern, as it probes for bandwidth, experiences loss, backs off, and repeats the cycle.

Although these methods have largely been successful in the past, it is still the case that in the current Internet a TCP flow may observe near-zero goodput when a large number of TCP flows share a bottleneck link. The consequent competition for resources results in catastrophic collapse of the per-flow performance, even though the link is operating at full utilization. Each TCP flow probes for available bandwidth to see if it may increase the amount of data it has “in-flight” in the network. Current implementations of TCP have a minimum probe bandwidth of one segment per round-trip time (RTT), or one segment per retransmission timeout (RTO) if the probe packet is discarded. If too many TCP connections are admitted, the total probe bandwidth can itself exceed the capacity of the bottleneck link, resulting in a substantial increase in retransmitted data and therefore wasted bandwidth, and woefully inadequate performance per flow. Given the inability of current TCP implementations to back off further, the congestion control problem at this point has become network-centric, rather than host-centric, and so it requires appropriate network controls.

This state of congestion collapse has frequently been observed on the U.K.–U.S. SuperJANET transatlantic link. This link is a major bottleneck for traffic flowing from the U.S. to British universities, and has historically been gravely under-resourced relative to peak demand. Given the introduction of usage-based charging on this link[1] —and such measures show every indication of becoming more, not less, widespread—ensuring reasonable goodput in such cases has become important in order to limit the total cost of bandwidth used. Frustration experienced while trying to use this link during peak times provided the main motivation for undertaking this work.

Even if this somewhat extreme scenario does not occur, there is often a minimum TCP bandwidth required to achieve a minimal session-level user utility. For example, web users who have to wait too long for all of the objects within a web page to complete downloading may give up and hit “stop,” or worse, “restart” the download. This wastes already scarce network resources, reducing the number of “successfully completed” TCP connections, which in turn decreases the number of successfully downloaded pages—their connection level and session level goodput, respectively. When a user causes a flow to be aborted due to poor performance, bandwidth has effectively been wasted at the very time it was most scarce, since the data already transferred is of little or no use, and restarting the flow will usually require that this data be retransmitted.

Furthermore, it is known that TCP does not share the available bandwidth fairly under high load consisting of many flows [7].

Thus, as Massoulié and Roberts [8] and Kumar *et al.* [9] argue in more detail, it makes sense to allow Internet service providers (ISPs) to control the admission of traffic at a variety of levels and not just at the packet level. This should help to temper congestion, and ensure that bottlenecks never become so heavily overloaded that real-time services and interactive applications over TCP can make no useful progress.

[1]Currently £0.02 ($0.032) per megabyte for U.K.-bound traffic.

B. Contribution

In this paper we consider only connection-level admission, as opposed to session or flow-level admission, and focus our attention on the most prevalent current protocol—TCP. In future work we intend to address other protocols, such as RTP and RealAudio, and to extend our scheme to take session-level semantics into account. For example, persistent HTTP connections, as in HTTP/1.1, reduce the semantic gap between session- and flow-level admission control, and the effects of this should be further investigated.

We first introduce admission control and its application to the Internet in Section II, along with a brief discussion of how we measure success. Section III discusses the details of our approach and the implementation of admission control for TCP. Our initial investigation was carried out using simulations over a simple dumbbell topology with a single bottleneck link; these are described and discussed in Section IV. We then validated our method using a test-bed implementation, described in Section V, and carried out further simulations of particular interesting cases. We summarize and conclude with a brief discussion of future work in Section VI.

II. Admission Control

Any admission control function requires knowledge of both the state of the network and the potential impact on existing flows of the admission of another flow. Using this information it is possible to decide whether or not a new flow should be admitted to a link of limited resource. Admission control must be performed by the *network*, since the network cannot rely on cooperative behavior of the sources in competition for the resource, even when the protocol does so, as in TCP. In traditional networks using CAC, the source explicitly signals the network to request access. TCP has no such explicit network-signaling procedure, and it is therefore necessary to perform admission control by having the network examine traffic and identify new flows as they commence.

Access to the network is only part of the problem. The network must also ensure that resources are available to carry the accepted traffic. CAC in conventional telephony systems is simplified by the fact that connections require unit resource and are established end-to-end. This makes it easy for the network to know if it may accept a connection, since it is of a known, constant bandwidth, with a route determined at connection setup time. Any switch on the route may reject a connection during the connection setup phase. Typical ATM signaling methods [10] use a similar end-to-end system, but require that the connection should declare certain parameters, such as the peak and sustained rates, in order that resources may be reserved at connection setup.

An alternative solution to requiring the connection to explicitly declare its traffic parameters is to use *measurement based* *admission control (MBAC)*, in which the network measures its current load [11], [12]. It then uses these measurements to make a decision about whether it should accept a new connection. This approach has the advantage that it relaxes the requirement that the application know *a priori* the statistical details of the traffic it will send. In many cases these parameters cannot be known in advance because the content of the connection may be dynamically generated (e.g., by a compression algorithm) and moreover the packet flow may be modified en route to the bottleneck due to buffering at intermediate routers. Obviating the need for applications to parameterize themselves is clearly desirable in a fast-moving environment like the Internet, where new applications are developed and deployed frequently. In addition, since the Internet is a public access network which currently has poor support for network charging or policing, it is unlikely that the network would be able to trust traffic parameters declared by users.

A. Admission Control in the Internet

The problem with applying admission control to the Internet is that the Internet is based on a connectionless packet-forwarding protocol, IP. In general, it is not possible to know in advance the route that any one packet will follow between source and destination. Indeed, the packets constituting a flow may well take different routes during the lifetime of a flow, and there is consequently no way to reserve bandwidth for a flow in advance. Although RSVP [13] has been proposed as a candidate soft-state signaling protocol for enabling this, it has yet to be widely accepted. Even if route-pinning is used, the pinning may only apply to a subset of the total number of packets comprising a flow.

In spite of the Internet's underlying connectionless forwarding mechanism, most communication between Internet hosts is actually connection-orientated, using higher level protocols such as TCP. By delving inside IP datagrams to decode the higher level protocol information, routers can identify individual connections. This can be done using modern software or hardware-based packet filtering mechanisms, which allow the identification of particular types of packets, such as TCP `SYN` packets, that are attempting to establish new connections.[2]

An admission control algorithm applying such implicit identification of new flows can selectively cause the deferral or rejection of connection setup attempts by reacting to the connection setup process. The decision as to whether to intervene in the creation of a new connection can be driven by an MBAC process which monitors the level of congestion at the router, enabling severe overload conditions to be avoided. Since this scheme operates without cooperation from end-systems, and without modification to the network as a whole, we term it *implicit admission control*.

Note that an implicit admission control scheme does not require per-flow state to operate successfully. Admission control decisions are based on the estimated resources remaining on the link at the point at which the connection request is intercepted. We assume that data packets belonging to a given flow will generally follow the same route as its connection setup packet. In particular, one obvious place for deployment of our system is at the users' ingress to the network, where there is no alternative route. This is not an absolute requirement, however, since our load estimate is calculated in real time and based solely on measurements of the traffic. This means that such routers will still perform satisfactorily even when they only see one direction of a flow, or when a flow is rerouted through or around a link in midstream.

B. Measuring Success

We measure performance by the goodput of a connection, as previously defined. We attempt to achieve high per-connection goodput by limiting the number of connections active on the bottleneck link at any time. With no admission control, overall goodput may remain high since each successfully `ACK`ed packet contributes to the system's goodput, but a given flow's goodput may become unacceptably low. This is a consequence of the high number of retransmissions it must make to get each packet through, and the corresponding substantial increase in wasted bandwidth and the flow's duration.

Even so, we still wish to admit a "useful" number of connections as it is clearly not satisfactory to be overly conservative in the number of connections admitted. We also wish to ensure that the flows themselves can complete in a reasonable time. For many application-layer protocols running over TCP, it is often the case that received data are not useful until all the data have been received and the flow has completed.

In addition, a good admission control algorithm should not be unfair to any particular type of connection. For example, connections with high RTTs should not be penalized in comparison to those with lower RTTs.[3] Furthermore, a practical admission control algorithm must be efficient in terms of its measurement, computation, and state requirements, and should be capable of dealing with a potentially large number of connections.

In summary, adding admission control to a link should not interfere with the link when it is not overloaded; should not overly limit the utilization or number of flows allowed into the link; and should not be biased against a particular type of flow. It should increase the utility users receive from the network by ensuring that the goodput of the system is accurately reflected by each flow.

III. Implementation of Admission Control in the Internet

Our approach is to perform admission control at layers above IP that have a well-defined notion of a connection. Although we address only TCP admission control in this paper, we believe this approach is feasible for other connection-orientated protocols running over IP. Rather than require that TCP be changed, we modify a few specific routers at well-known bottleneck links, thus avoiding the problems in requiring widespread deployment of new technology.

[2]We note that the use of IPsec would prevent this style of connection detection; should IPsec become widespread, alternative methods of connection detection and rejection would be required.

[3]TCP itself does penalize such connections, but we would hope that an admission control algorithm would not increase this unfairness.

By capturing TCP `SYN` or `SYN`/`ACK` packets, it is possible to efficiently intercept connection setup requests. New flows can then be accepted, by allowing the packet to proceed as normal, or rejected by suitable means. We discuss methods of connection rejection in Section V-B. We do not deal with the resource allocation problem, leaving that to the protocol, TCP, to perform. This allows our scheme to work without modification to the TCP protocol, so once a connection is accepted, the bandwidth achieved is dependent on the TCP implementation. This is similar to the philosophy of the UNITE protocol for IP on ATM [14], "separating connectivity from QoS control."

In summary, our algorithm is very simple.

- The load estimator places the admission controller in one of two states: `accepting` or `rejecting`.
- On detection of a new connection attempt:
    - if the controller is in the `accepting` state, the `SYN` or `SYN`/`ACK` packet is allowed to proceed without interference down the bottleneck link;
    - if the controller is in the `rejecting` state, then the connection attempt represented by the `SYN` or `SYN`/`ACK` is rejected or deferred.

A. The Admission Control Decision

Whether the admission controller is in the accepting or rejecting state can be based on a variety of information. Perhaps the most obvious basis for the admission decision would be to limit the number of connections passing through the router. However, due to the asymmetric nature of much Internet traffic [15], the bottleneck router carrying the bulk of the traffic may not see all the control packets associated with a flow ("asymmetry of routing"), and may only be overloaded in one direction ("asymmetry of load"). Consequently, any scheme based on counting connections would be obliged to continually infer the active connections traversing it, for example, by applying filters either to a sample or to all packet headers and keeping state on the ⟨*source address, source port, destination address, destination port*⟩ tuples associated with packets. This requires that per-connection state be kept, which might only be considered scalable at the network edge. In addition, due to asymmetry of routing, a method to expire flows considered inactive is required. The existence of routers capable of implementing weighted fair queueing for large (approximately 64 000) numbers of flows suggests that it might be possible, however.

Rather than explicitly counting the number of flows traversing a node, one might estimate the number of flows. For instance, there are models [16], [17] of TCP steady-state behavior, giving throughput estimates in terms of RTT and packet drop-probability. Such models might enable one to simply relate the drop statistics at a router to a number of flows, giving an estimate of the number of flows traversing the router. Another approach is to use the evolution of the queue length over time and the fact that TCP has well-known behavior to estimate the number of flows. This is obviously complicated by the variation in RTTs in the Internet, and by the difference in behavior between TCP in slow-start and TCP in congestion-avoidance. In the current implementation we have followed a different approach, and use a particular on-line effective bandwidth estimator to inform the admission control decision, with a threshold chosen by the operator. Kumar *et al.* [9] take a similar approach, but use a measure of the expected per-session bandwidth share ("normalized offered load") and relate this to queue occupancy. Again, the operator then chooses a threshold that will give users the desired minimum level of service.

B. Algorithm Implementation

The current implementation uses on-line measurements based on aggregate load to move the controller between the `accepting` and `rejecting` states. It retains no per-connection state, and is therefore oblivious to the termination of flows. In this initial implementation of the proposed scheme, we chose to use an effective bandwidth [18] estimator from the `Mtk` toolkit, developed by Glasgow University Computer Science Department, as part of the Measure project [19]–[21], in conjunction with Cambridge University Computer Laboratory and the Dublin Institute for Advanced Studies.

This estimator uses simple measurements of the arrivals process to a queue to estimate the entropy of the traffic. Combining this estimate with two constraints, the maximum queue size in the router and a target overflow probability of the buffer, the estimator computes the effective bandwidth requirement of the aggregate traffic mix. The effective bandwidth can then simply be subtracted from the total capacity of the bottleneck link to yield the remaining resource in the system. Alternatively, given the total transmission capacity, the arrivals process and the buffer size, `Mtk` can be used to determine when the admission of a new flow is likely to cause the target overflow probability to be violated. The Appendix provides a brief introduction to the mathematics underlying the estimation procedure used by `Mtk`.

Those connections that are accepted onto the link compete with each other in the usual way, relying on the TCP flow and congestion control algorithms to achieve fair allocation of the available bandwidth. Each TCP connection is limited in its transmission rate by the lowest capacity link on its path. Consequently, all connections that are not limited by receiver window will experience packet discard. As a result it makes no sense to attempt to limit the probability of packet discard to zero. Nonetheless, in a properly dimensioned network, the packet discard probability should not become excessively high. The threshold target value is up to the operator to set: lower means connections will see a higher quality, more exclusive network; higher means connections will see a lower quality, more accessible network. For this estimator we would ideally use a target overflow probability which would achieve, for each of n active connections, a loss rate equal to the minimum loss rate required by TCP in order to correctly establish the available rate at the bottleneck link. The results below contain a wide range of target loss rates, to show the system response under various loads. The exact parameterization of the system will depend on the estimator in use, and on the service that the network operator wishes to provide.

We would stress that our approach does not rely on this particular estimator, and indeed we would not claim that this estimator is especially suited for the task at hand; ongoing work

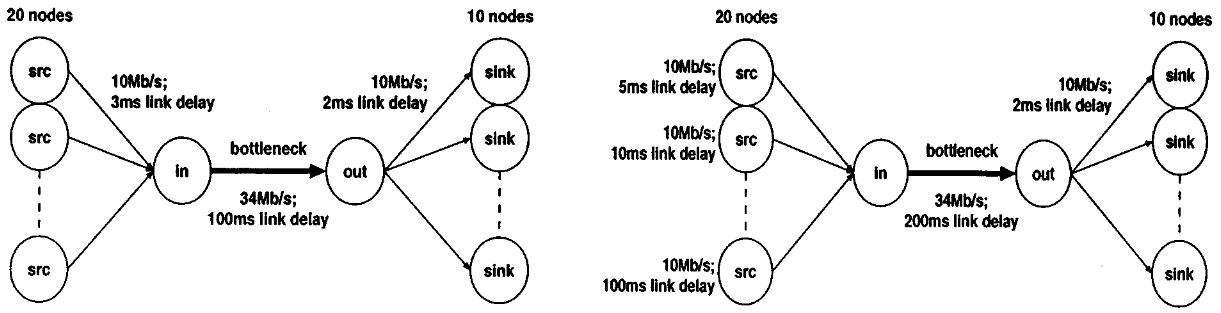


Fig. 1. Simulator topologies used. The left-hand figure shows the identical link delay topology, and the right-hand figure shows the differing link delay topology. Admission control is applied at the "in" node.

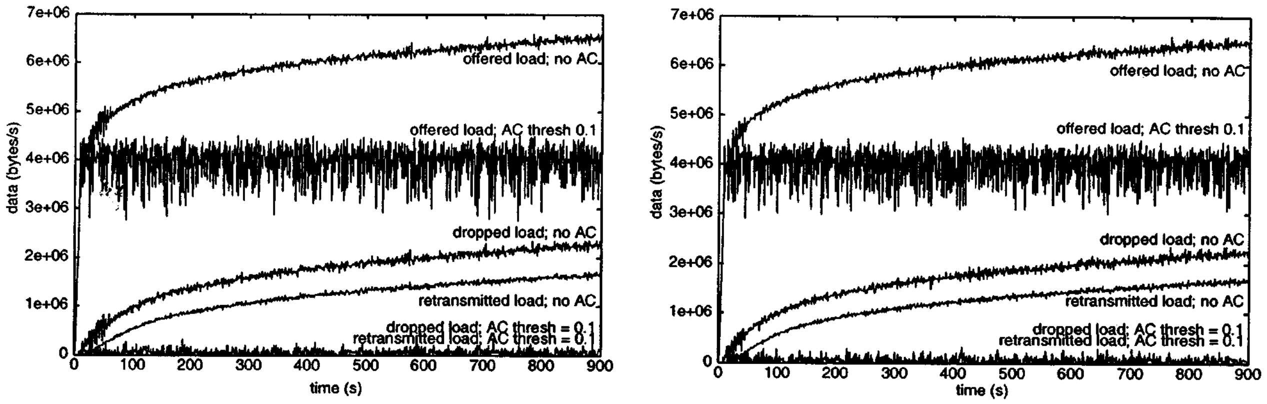


Fig. 2. Graphs of offered load, drops, and retransmissions, without admission control and with an admission threshold of 0.1. The topologies are shown in Fig. 1, with the left-hand graph for the left-hand topology and the right-hand graph for the right-hand topology, both using the simple traffic model.

is attempting to identify more appropriate algorithms for this traffic. We believe that the general approach of implicit admission control is robust to the choice of estimator. We hope that this is supported by the success of our results using the `Mtk` estimator in situations quite outside its design remit.

IV. Simulations

We used `NS` (*Network Simulator v2*) developed by the VINT project at Lawrence Berkeley Laboratories [22] to obtain performance results for our proposed system. `NS` is a discrete event simulator designed for the simulation of Internet protocols. It contains code to simulate a large number of TCP implementations, in addition to standard network elements, such as simplex- and duplex-links, and routers with different queueing disciplines, principally *DropTail* and *RED*. By linking `NS` with `Mtk`, we were able to test the efficacy of our proposed admission control scheme.

When a connection starts in a simulation, the node at the ingress to the bottleneck link considers its current drop-probability estimate in relation to the threshold set by the operator. This allows it to `accept` or `reject` the connection, based on whether the estimate is lower or higher than the threshold set by the operator. Section V-B describes approaches used to deny admission to connection attempts in our test-bed network. Since `Mtk` only utilizes measurements of aggregate arrivals in this implementation, the overhead of the estimation process in the router is not as high as might first appear. The `Mtk` estimator performs computation periodically, and not based on traffic arrivals, so we believe that a denial-of-service attack by overloading the node with `SYN` packets should not be possible.

A. Network Model

We used two topologies, shown in Fig. 1: one is a simple dumbbell topology with constant delay links; the other is similar, but with links of varying delay to simulate flows with differing RTTs. We hope to address more complex topologies in followup work, including multiple bottlenecks and situations where cross-traffic, both responsive and unresponsive, interferes with flows through the bottlenecks. We also used two basic traffic models: one a simple model with fixed packet size and 1 MB flow length purposely chosen to overload the link, and with interflow arrival intervals drawn from an exponential distribution with mean 1 s, and the other a more complex model with varying flow lengths, constructed from data obtained by analyzing web-cache logs from a variety of sources. We added a `drop-tail-mtk` node type to `NS`, an extension of its `drop-tail` node. This implemented the admission control algorithm, and collected the per-flow trace information necessary to produce the tables and graphs in the following section.

B. Simple Traffic Model

The first set of results are shown for the simple traffic model in Fig. 2 with the left-hand graph showing the results for the simple, identical link topology, and the right-hand the results for the differing link topology. For the first case, we see that without admission control, the offered load is approximately

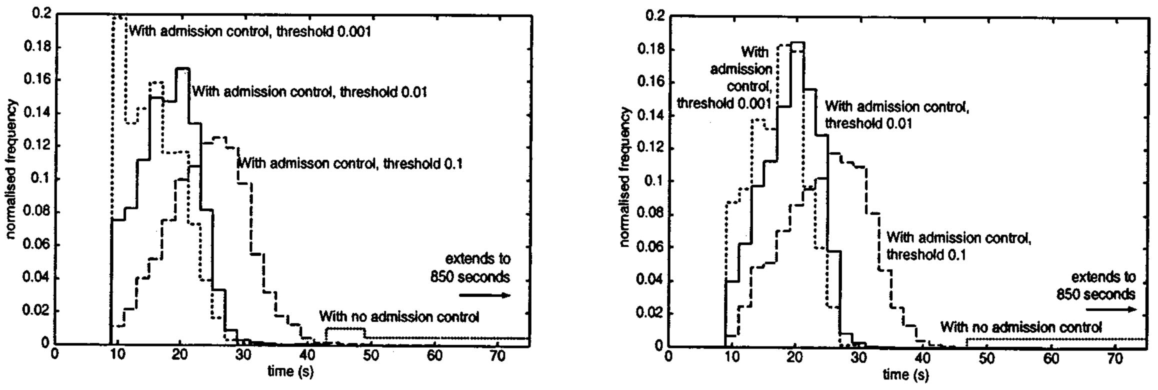


Fig. 3. These graphs show histograms of flow durations, where the frequency count has been made over buckets of 2 s, and normalized to the total number of flows to complete. The x axis is the duration, and the y axis the normalized frequency. The topologies used are shown in Fig. 1, with the left-hand graph for the left-hand topology and the right-hand graph for the right-hand topology. Both graphs are for the simple traffic model. Note that the x axis has been truncated for clarity; due to the "no admission control" case, it actually extends to 894 s.

Flow durations: Identical link delays, simple traffic model

Threshold	Flows Completed	Packets Received	Packets Retransmitted	Mean (s)	Std Dev (s)
None	186	186000	687504	509.06	228.03
1.0	2493	2493000	20052	135.08	84.951
0.5	2831	2831000	9457	130.62	94.948
0.1	3323	3323000	1372	54.583	25.543
0.05	3349	3349000	562	42.563	16.146
0.01	3413	3413000	255	34.073	16.950

Flow durations: Differing link delays, simple traffic model

Threshold	Flows Completed	Packets Received	Packets Retransmitted	Mean (s)	Std Dev (s)
None	162	162000	699275	482.33	226.67
1.0	2476	2476000	18444	130.29	85.856
0.5	2874	2874000	9082	123.73	92.613
0.1	3365	3365000	1307	52.168	29.405
0.05	3394	3394000	617	42.747	31.098
0.01	3443	3443000	231	33.638	15.046

Fig. 4. Tables showing the number of flows completed, packets transferred by completed flows, the total number of packets retransmitted, and the duration means and standard deviations for the completed flows. The simulations were run for 900 s. using the topologies shown in Fig. 1 and the simple traffic model. The upper table is for the left-hand topology and the lower table for the right-hand topology.

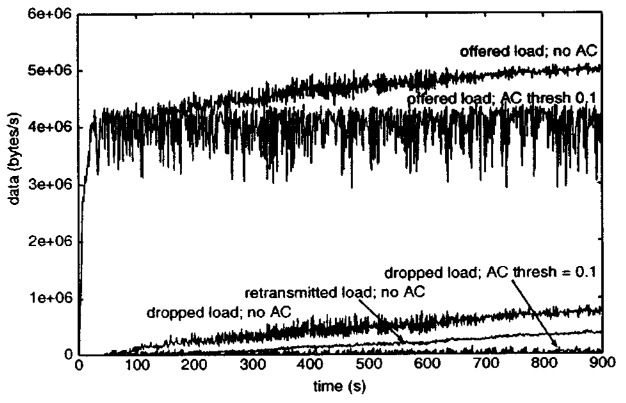
30% higher than the link capacity, leading to approximately 30% of the traffic on the link being retransmissions, due to the large volume of packets being discarded. Conversely, when admission control is turned on, the offered load is kept slightly below the link's capacity, ensuring that drops and consequent retransmissions are tightly constrained. For the second case, we see results that do not differ significantly from those in the first. We also simulated this topology using a variety of packet sizes, and found that this had similarly negligible effect.

Based on these results, we also show histograms of the time to successful completion for flows in Fig. 3, and tables of their mean and standard deviation in Fig. 4. These clearly demonstrate that employing admission control can greatly increase the number of flows that successfully complete in a given time interval by allowing flows to complete substantially faster. Without admission control, most flows do not complete, and those that do have a mean of 509 s and a standard deviation of approximately half the mean. Conversely, completion times when admission control is applied as leniently as the current estimator allows have a mean of 135 s, and a correspondingly lower standard deviation, with nearly 20 times more flows completing.

Since TCP is "greedy," that is, admitted flows will attempt to use the available bandwidth in the bottleneck, the link remains at near-full utilization even with admission control in place. This is shown by the results in Fig. 2. In conjunction with those results, the results shown in Figs. 3 and 4 demonstrate that many applications will achieve higher utility when admission control is applied. Users may be prepared to wait for 1 min for a large download to complete; they are less likely to be prepared to wait for 15 min. The results also show that it is possible, even with the `Mtk` estimator which is not optimized for this type of traffic, for the network operator to tune the network based on users applications' requirements, in order that they receive higher utility.

C. Complex Traffic Model

We now consider the results for the web-log-based complex traffic model. As can be seen in Fig. 5, admission control has a similar effect as with the simple traffic model: the offered load is kept at or slightly below the link capacity when admission control is applied, but continues to rise when no admission control is in place. The drops and retransmissions also exhibit similar behavior to the results for the simple traffic model. However, the flow duration histogram in Fig. 5 and the table of the mean and standard deviations in Fig. 6 show that fewer flows complete successfully with admission control in place.



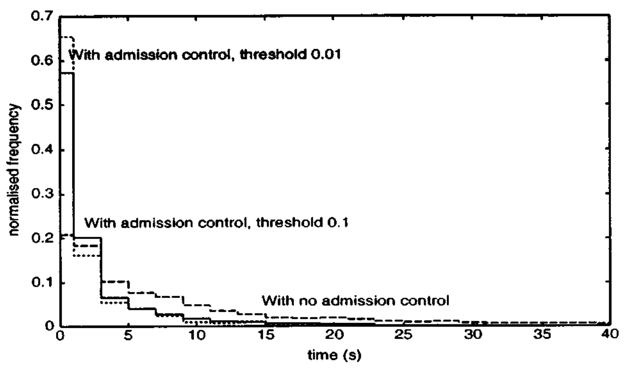


Fig. 5. The left-hand graph shows offered load, drops, and retransmissions, without admission control and with an admission threshold on the target loss probability of 0.1. The right-hand histogram is of flow durations, where the frequency count has been made over buckets of 2 s, and normalized to the total number of flows to complete. The x axis is the duration, and the y axis the normalized frequency. Both used the left-hand topology in Fig. 1, with the complex traffic model.

Flow durations: Identical link delays, complex traffic model

Threshold	Flows Completed	Packets Received	Packets Retransmitted	Mean (s)	Std Dev (s)
None	17180	1842083	63269	15.350	35.273
1.0	14027	1869053	10325	10.712	31.099
0.5	13389	1855134	5738	9.5144	28.359
0.1	12070	1860911	1146	7.2418	21.878
0.05	11778	1825970	859	6.7397	20.311
0.01	10915	1771488	313	5.8935	17.947

Fig. 6. Table showing the number of flows completed, packets transferred by completed flows, the total number of packets retransmitted, and the duration mean and standard deviations for the completed flows. The simulations were run for 900 s, using the complex web-cache log based traffic model. The topology used was the left-hand topology in Fig. 1.

Threshold	Completed Flows	"Good" Flows	[%]	"Bad" Flows	[%]
None	15219	4595	[30%]	10624	[70%]
1.0	12065	7255	[60%]	4810	[40%]
0.5	11427	7968	[70%]	3459	[30%]
0.1	10192	8591	[84%]	1601	[16%]
0.05	9900	8634	[87%]	1266	[13%]
0.01	9157	8618	[94%]	539	[6%]

Fig. 7. Table showing the number of completed flows with the number that met a target of 10 packets per second over their lifetime ("good" flows), and the number that failed to meet this target ("bad" flows). Only flows that started after the first 100 s had passed are counted, in order to remove initial transient behavior.

The table in Fig. 6 clearly shows that flows are completing faster and with more tightly controlled durations when admission control is applied. However, fewer flows complete successfully which appears discouraging. Examination of the number of packets received reveals an explanation. When admission control is applied, approximately the same number of packets are successfully received, suggesting that link utilization remains the same. Without admission control, the proportion of retransmissions, for the longer flows in particular, rises as the existing longer flows lose out to the shorter flows in slow start. When admission control is applied, the longer flows are able to complete since the excess short flows are unable to enter the link and cause the long flows to experience excessive loss.

Examining the table in Fig. 7 provides further insight. Setting a target of 10 packets per second per flow as a measure of "useful" goodput, we see that application of admission control nearly doubles the number of "good" flows that complete. This suggests that a large number of the extra flows that manage to complete with no admission control are receiving very low transfer rates, and are hence of less use. This table also suggests a manner in which the operator could set the threshold. One might choose a target throughput and then adjust the admission threshold to achieve it—the particular value depending on the traffic mix and on the level of service the operator wishes to provide for its customers. Mtk appears to give a reasonable range of values for the operator to tune to, which is encouraging given that the link is only experiencing overload of approximately 20%. Better estimators might give one a more controllable parameter with greater dynamic range—a weakness of Mtk in these circumstances is that its maximum threshold is 1.0, which leaves quite a large gap in system behavior between no admission control and admission control at its most lenient.

V. Implementation

The purpose of the implementation was to allow us to validate the simulations, and to test interaction with applications and protocol implementations. The implementation used standard Pentium III PCs running the Linux 2.2.9 operating system for both the sources and sinks, and the admission control router. The network was 100 Mb/s switched Ethernet, and all machines used 3Com 3c509 NICs. The router software consisted of the Linux 2.2.9 kernel compiled with support for IP forwarding, and for the QoS/Fair queueing options. The router was configured using

Operating System	SYN RTO interval sequence (s) Data RTO interval sequence (s)	total (s) total (s)
FreeBSD 2.2.7	2.8, 6.0, 12.0, 24.0	44.8
	1.4, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0*7	511.4
HPUX 9.05	3.7, 10.1, 24.0	37.8
	0.5, 0.5, 4.0, 8.0, 16.0, 32.0, 64.0*7	509.0
Linux 2.2.9	3.0, 6.0, 12.0, 24.0, 48.0, 96.0, 120.0*5	789.0
	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8, 25.6, 51.2, 102.4, 120.0*6	924.6
NetBSD 1.3	6.0, 12.0, 24.0	42.0
	1.0*11	11.0
OSF/1 3.2D	0.7, 3.0, 6.0, 12.0, 24.0	45.7
	1.4, 3.0, 6.0, 12.0, 24.0, 48.0, 64.0*8	606.6
SunOS 5.5.1	1.7, 5.1, 11.8, 25.3, 52.3, 106.3, 162.6	365.1
	0.9, 0.8, 1.5, 3.0, 6.0, 12.0, 24.0, 48.0, 56.3*6	434.0
SunOS 5.6	3.5, 6.4, 12.8, 25.6, 51.2	99.5
	0.2, 0.5, 1.0, 3.8, 7.6, 15.3, 30.6, 61.2, 122.4	242.6
Windows 98	2.9, 6.0, 12.0	20.9
	0.3, 0.6, 1.2, 2.4, 4.8	9.3
Windows NT4.0 SP3	3.2, 6.6, 13.1	23.0
	0.6, 0.9, 1.8, 3.5, 7.0	13.8

Fig. 8. Measurements of packet retransmission intervals for some TCP implementations following SYN and data loss. $x,\ y$ means that packet p_y was retransmitted y seconds after packet p_x. $x * n$ means that n packets were retransmitted at intervals of x seconds.

the Linux traffic control engine [23], for which a module was written to enable queue measurements to be made and passed to the estimator.

A. Performance Tests

The baseline performance of the Linux forwarding code was measured by performing a "flood ping" between the two hosts that were connected by the router. Even using small packets, Linux is able to forward at the 100 Mb/s line rate without CPU usage exceeding 20%.[4] Installing the instrumented queueing module and the `Mtk` estimator made no difference to the router's CPU usage. Our experience suggests that significant amounts of CPU time are available, enabling us to consider more complex bandwidth estimation functions in the future.

In order to test the admission control algorithm, traffic was generated using `pttcp`, a locally developed utility, based on `ttcp` [24]. `pttcp` allowed us to set up large numbers of connections between the hosts, in either a transmit–receive mode, or in a client–server mode, to more accurately reflect web traffic (i.e., the client sets up a connection and requests n bytes from the server, which then transmits n bytes to the client).

The admission control process was implemented using the Linux IPChains [25] software, which allows user-space programs to be passed packets of interest for examination, and corresponding responses to be generated and transmitted. This was modified to allow more flexibility in specifying packets of interest. The IPChains software intercepted packets with header bits of interest and passed them to the handler in the user code for the admission decision to be made. Analysis of the results was performed using John Ostermann's `tcptrace` [26] along with further postprocessing.

B. Connection Admission and Rejection

The best way to implement the detection of connection admission requests and the rejection of connections is dependent on a number of factors, including the traffic characteristics in the network, the underlying link technology, the longevity of congestion at the link, and the protocols in use. Our implementation allows any combination of the TCP option bits to be treated as of interest; typically one would treat SYN or SYN/ACK packets as connection requests. To signal rejection, we tested both dropping the admission request, and sending an RST to both parties.

We believe that a practical implementation would need to allow all the above possibilities. For example, consider a web server being accessed by a client which experiences asymmetric routing to/from the server. The bottleneck link might never see the SYN, in which case the SYN/ACK must be used. Dropping the request naturally reduces the retry rate as TCP backs off, but means that the web user sees no response, and may well retry more quickly. Sending the RST allows the web user to notice that the server cannot be accessed, presenting them the option of cancelling the session. In cases where it is detected that the connection is not being used to transfer Web traffic, there may well be other more appropriate options.

In the case where the SYN/ACK is intercepted, the sending of the RST might itself be considered harmful as it will use resource at the times when it is most scarce. One might prefer to simply drop the SYN/ACK, or indeed the SYN, leaving TCPs normal operation to deal with the retries and possible eventual denial. This has the advantage that traffic is not injected into the link at times of high load, but does mean that the user may experience large timeout delays before being informed that the connection cannot be made at this time.

The relevant RFC [27] states that the backoff sequence when a SYN is dropped should be exponential as for normal traffic loss, but *must* last for at least 3 min instead of 100 s, the recommended value for data traffic. This would ensure that retransmitted SYN packets do not themselves overload the link. Fig. 8 is the result of measuring the RTO intervals for a number of TCP implementations, and shows this to be the case. The decision to retry may also be taken by the application or user, rather than the protocol.

Other suggested methods of denying admission to a connection include using ICMP `Source Quench` and ICMP

[4]The routing table used in this experiment contained only two entries. Some degradation in performance may occur with significantly larger tables.

`Reject: Unknown Protocol` messages [9]. The former has the advantage that it also allows the operator to control the throughput of active connections as it reduces the receiver's congestion window to one. The latter has the disadvantage that in addition to denying the requesting flow access, it can also cause existing flows between the same endpoints to break.

Perhaps the most important thing to note from this table is that all the implementations examined do use an exponential backoff sequence for `SYN` retries, so deferral of a connection should not result in excessive control traffic being generated. In addition, the `SYN` backoff sequence for all the implementations studied is quite reasonable, albeit not completely RFC compliant, with at least 4 retries occurring over at least 20 s. Consequently, TCP can be relied on to retry later if a CAC-capable router chooses to defer or reject a connection. Another notable point is that some TCP implementations are not very robust to midstream loss. For example, Windows 98 gives up in under 10 s after losing just 6 packets. Application of admission control as we propose should make such loss sequences less likely, and hence reduce the number of broken connections.

C. Impact on Applications

In addition to using the implementation to gain some real-life confidence in our method, we also used it to test the behavior of some popular Internet applications when denied admission by a `RST`. Since the Web is currently the most popular use of the Internet, we tried Netscape v4.5 on both Linux 2.2.9 and Microsoft Windows NT, and Internet Explorer v4 on Microsoft Windows NT. In all these situations, we found that when a TCP connection is rejected by our admission controller the application will silently accept that it could not retrieve an object on the page unless it is the base page itself, in which case a dialog box is popped up informing the user that the page cannot be retrieved. Further, it appears that Netscape has a timeout of approximately 30 s before it gives up on TCP retry attempts, whereas Internet Explorer attempts to connect 4 times for a given source port, and then repeats this for a further 4 different source ports, incrementing the source port by one each time.

VI. SUMMARY

We have argued for the use of admission control at routers in the Internet as a mechanism which we believe has the potential to improve the performance experienced by users of both inelastic real-time services, and elastic data services, such as TCP. We have focused specifically on the admission control of TCP flows, and have shown that, under certain circumstances, TCP performance can collapse due to overadmission of flows to congested bottleneck links. Under these circumstances, as the number of admitted flows grows in an unbounded fashion, the corresponding throughput experienced by each flow falls to an operating region in which TCP is neither stable, fair, nor useful from the application and user points of view.

We have shown by way of simulation that the implementation of a simple admission control scheme at routers in the Internet can dramatically improve the performance experienced by all users. Although it is true that our controller will reject some flows, in the unconstrained network those flows may only receive a share of the bottleneck link bandwidth likely to be too small for interactive use, and so their performance is almost unchanged. For those flows that are admitted, performance, whether measured by the per flow goodput or the completion time for transfer of data for an elastic flow, is dramatically improved. Our scheme also increases the fairness of the resource allocation between those TCP connections that have been admitted. Furthermore, it need not be deployed throughout the Internet for the benefit to be seen; deployment at a limited number of known bottlenecks should have a noticeably positive effect. A beneficial side effect of limiting the number of flows traversing a bottleneck link is that we also greatly reduce the bandwidth wasted by retransmissions upstream of the bottleneck, freeing this resource for use by other flows for which these upstream routers may be bottlenecks.

Our admission controller is simple and efficient, requiring little per-packet processing and no per-flow state. We believe that using a measurement based admission control scheme confers significant advantages: the scheme is robust to fluctuations in the offered load, requires no *a priori* per-flow traffic characterization—generally impossible to derive—and bases its estimations on measurements of the aggregate load. Finally, using the `Mtk` estimator, the network operator can tune the admission threshold, giving greater control over the service quality experienced by users' traffic in the network.

We have implemented our controller on PCs running Linux, and confirmed that the processing overhead of the scheme is minimal. Our experiments allowed us to examine the behavior of popular Internet applications in the face of admission control; we found that all applications tested were robust to the failure of their connection setup attempts. In addition, we tested a number of TCP implementations and found that their state-machines were robust in the face of `SYN`s and `SYN/ACK`s being reset, and that they also implemented reasonable backoff strategies for both dropped data and dropped control traffic.

Future Work

Our investigation of admission control in the Internet is ongoing. We are continuing work on instrumenting our implementation, measuring its performance under a variety of loads, and will further validate application behavior when admission is denied. As discussed, we are also researching additional algorithms and MBAC estimators, of which a variety have been published in the literature, and the need to address the specification of admission thresholds by network operators. Robustness in the face of the highly correlated TCP traffic process, and the highly variable topology of the Internet is important. We are also developing and testing both implicit and explicit admission controls for other Internet protocols. We believe that our results to date are promising and indicate the potential of this approach.

APPENDIX
LARGE DEVIATIONS' THEORY

The theory of large deviations [28] is based around the observation that as more data points are taken from a distribution, one expects that the ratio of "rare events"—those events that give the distribution its tail—to other events should decay exponentially. That is, *the tail of the distribution decays exponentially*. In a sense it is the flip side of the central limit theorem (CLT): the CLT governs random fluctuations near the mean, of the order of $\sigma/\sqrt{n}$, where n is the number of statistics, and σ^2 is the variance; deviations of the order of σ are much larger ("large deviations") and occur only rarely. It is these events that are governed by large deviations' theory. More formally, Cramér's theorem states the following.

Let $X_1, X_2, X_3, \cdots$ be a sequence of bounded, independent and identically distributed random variables, each with mean m and let

$$M_n = \frac{1}{n}(X_1 + \cdots + X_n)$$

denote the empirical mean; then the tails of the probability distribution of M_n decay exponentially, with increasing n at a rate given by a convex rate-function $I(x)$:

$$\begin{aligned} \mathbb{P}(M_n > x) &\asymp e^{-nI(x)} \qquad \text{for } x > m \\ \mathbb{P}(M_n < x) &\asymp e^{-nI(x)} \qquad \text{for } x < m. \end{aligned}$$

More generally, the conditions required for large deviations' theory to be successfully applied can be relaxed to $X_1, X_2, X_3, \cdots$ being bounded, weakly dependent, and stationary: the result that the tails of the distribution will decay exponentially still holds.

When one applies this result to queueing theory, one finds that for a single-server queue, the queue-length distribution has asymptotics of the form

$$\mathbb{P}(Q > q) \asymp e^{-\delta q}$$

where δ, the decay-rate, can be calculated from the rate-function $I(\cdot)$ of the arrival process via

$$\delta(r) = \min_x I(x)/x$$

where r is the service rate. More simply, if one can measure $I(\cdot)$, then δ may be calculated for any value of r, which is known. Given δ one can estimate packet-loss and packet-delay.

In general, one will not know the rate-function precisely. However, it turns out that δ may be estimated from a quantity known as the *scaled cumulant generating function (sCGF)* of the arriving work. An analogy is often made between this and the "entropy" of the arriving work. The sCGF is defined as

$$\lambda(\theta) = \lim_{n\to\infty} \frac{1}{n} \ln \mathbb{E}e^{n\theta M_n}$$

which can be shown to be the Legendre transform of $I(x)$, and so

$$\lambda(\theta) = \max_x(\theta x - I(x)).$$

Thus, if one measures the sCGF of the arriving work, one can calculate the decay rate of the function giving the probability of a packet being dropped, i.e., the *drop-probability*. Given this decay rate, one may calculate how big the buffer should be to ensure the drop-probability stays bounded above by some amount.

Estimating the sCGF

The crux of the matter is the estimation of the sCGF; if this is accurate, then the resultant calculations should be accurate (in the long term). Moreover, for this to be a feasible method for MBAC, the process must be cheap enough to perform on-line. There has been a great deal of investigation into the estimation of the sCGF recently, as it has application to the area of effective bandwidth [18], and so to charging for Internet traffic. As part of the measure project, a number of estimators for the sCGF have been developed. These estimators have been applied to ATM networks, and to the problem of resource allocation in operating systems, specifically the `Nemesis` operating system [29].

Given that the arrivals, X_i are weakly dependent, we can approximate the sCGF by a finite time *cumulant generating function (CGF)*. Letting A_t be a random variable representing the number of packets arriving at the queue in an interval of length t, the sCGF λ_A: $\mathbb{R} \to \mathbb{R}$ of the arrivals process is approximated as

$$\lambda_A^T(\theta) \simeq \frac{1}{T} \ln \mathbb{E}e^{\theta A_t}$$

for the block size, T, sufficiently large. Since $\delta(r)$ is related to λ, estimation of the expectation can then be performed by breaking data into K blocks of length T and averaging over them:

$$\lambda_A^T(\theta) \simeq \frac{1}{T} \ln \frac{1}{K} \sum_{k=1}^{K} e^{\hat{X}_k}$$

where the $\hat{X}_k$ are the block sums $\hat{X}_1 := X_1 + \cdots + X_T$, $\hat{X}_2 := X_{T+1} + \cdots + X_{2T}$, etc.

Then the asymptotic decay-rate of the queue-length distribution can be obtained, given the service rate, r, via

$$\widehat{\delta(r)} = \max_\theta \{\widehat{\lambda(\theta)} \le 0\}.$$

There are other ways of estimating the sCGF—for example, using a varying block size T rather than a fixed value as above.

Acknowledgment

The authors wish to acknowledge the authors of the various pieces of software used in this work, and J. Crowcroft, A. Donnelly, S. Hand, T. Harris, and P. Jardetzky for interesting discussions and suggestions. They would also like to thank the anonymous reviewers and the editors for their helpful comments.

`pttcp` and the IPChains code will be made available from http://www.cl.cam.ac.uk/Research/SRG/netos/netx/; details about Measure and `Mtk` are available from the Measure web pages [21].

References

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," RFC1889, Jan. 1996.

[2] V. Jacobson and M. J. Karels, "Congestion avoidance and control," in *Proc. SIGCOMM'88*, Nov. 1988, pp. 314–329.

[3] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Commun. Rev.*, vol. 26, no. 3, pp. 5–21, July 1996.

[4] L. Brakmo and L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1465–1480, Oct. 1995.

[5] K. K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC2481, Jan. 1999.

[6] P. Key, D. McAuley, P. Barham, and K. Laevens, "Congestion pricing for congestion avoidance," Microsoft Research, Tech. Rep. MSR-TR-99-15, http://www.research.microsoft.com/research/network/disgame.htm, Feb. 1999.

[7] R. Morris, "TCP behavior with many flows," in *Proc. IEEE Int. Conf. Network Protocols*, Atlanta, GA, Oct. 1997.

[8] L. Massoulié and J. W. Roberts, "Arguments in favor of admission control for TCP flows," in *Proc. ITC-16: Teletraffic Eng. Competitive World*.

[9] A. Kumar, M. Hegde, S. V. R. Anand, B. N. Bindu, D. Thirumurthy, and A. A. Kherani, "Nonintrusive TCP connection admission control for bandwidth management of an Internet access link," *IEEE Commun. Mag.*, May 2000.

[10] ATM Forum Technical Committee, *User-Network Interface Signaling Specification*, July 1996.

[11] S. Jamin, S. J. Shenker, and P. B. Danzig, "Comparison of measurement-based admission control algorithms for controlled-load service," in *Proc. INFOCOM'97*, Apr. 1997.

[12] ——, "Measurement-based admission control algorithms for controlled-load service: A structural examination," Univ. Michigan, Rep. CSE-TR-333-97, Apr. 1997.

[13] J. Wroclawski, "The use of RSVP with IETF integrated services," RFC2210, Sept. 1997.

[14] K. K. Ramakrishnan, G. Hjalmtysson, and J. E. Van der Merwe, "The role of signaling in quality of service enabled networks," *IEEE Trans. Commun.*, vol. 37, pp. 124–132, June 1999.

[15] V. Paxson, "Measurements and analysis of end-to-end Internet dynamics," Ph.D. dissertation, Computer Science Division, Univ. California Berkeley, Apr. 1997.

[16] M. Mathis, J. Semske, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM Computer Commun. Rev.*, vol. 27, no. 3, July 1997.

[17] T. Ott, J. Kemperman, and M. Mathis. (1996, Aug.) The stationary behavior of ideal TCP congestion avoidance. [Online]ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps

[18] F. P. Kelly, "Notes on effective bandwidths," in *Stochastic Networks: Theory and Applications*, F. P. Kelly, S. Zachary, and I. Ziedins, Eds: Oxford University Press, 1996, pp. 141–168.

[19] S. Crosby, I. Leslie, J. Lewis, R. Russell, M. Huggard, and B. McGurk, "Predicting effective bandwidths of ATM and Ethernet traffic," in *Proc. 13th UK Teletraffic Symp.* Strathclyde Univ., Glasgow, Mar. 1996.

[20] S. Crosby, I. Leslie, J. Lewis, R. Russell, F. Toomey, and B. McGurk, "Practical connection admission control for ATM networks based on on-line measurements," in *Proc. IEEE ATM'97*, Lisbon, June 1997.

[21] (1999) Measure web pages. [Online]http://www.cl.cam.ac.uk/Research/SRG/netos/old-projects/measure/

[22] (1999) The UCB/LBNL/VINT network simulator, version 2. [Online]http://www-mash.cs.berkeley.edu/ns/

[23] A. Kuznetsov. (1999) IP route tools v2. [Online]ftp://ftp.inr.ac.ru/ip-routing/iproute2-current.tar.gz

[24] Chesapeake Computer Consultants, Inc. (1999) Test TCP. [Online]http://www.ccci.com/tools/ttcp/f1.html

[25] R. Russell *et al.*. (1999) Linux IP firewalling chains. [Online]http://www.rustcorp.com/linux/ipchains/

[26] S. Ostermann. (1999) tcptrace. [Online]http://jarok.cs.ohiou.edu/software/tcptrace/tcptrace.html

[27] Internet Engineering Task Force, "Requirements for Internet hosts—communication layers," RFC1122, R. Braden, Ed., Oct. 1989.

[28] J. Lewis and R. Russell, *An Introduction to Large Deviations for Teletraffic Engineers*. Dublin, Ireland: Dublin Institute for Advanced Studies, Nov. 1997.

[29] P. Barham, S. Crosby, T. Granger, N. Stratford, M. Huggard, and F. Toomey, "Measurement-based resource allocation for multimedia applications," in *Proc. SPIE: Multimedia Computing Networking 1998*: SPIE, 1998, vol. 3310.

Richard Mortier (S'99) received the B.A. degree in mathematics in 1996 and the Diploma in computer science in 1997, both from Cambridge University.

He is currently studying for the Ph.D. degree in the area of network resource allocation and control through pricing, with the Systems Research Group at Cambridge University Computer Laboratory. He is interested in many areas of systems, especially resource allocation and control in computer networks, operating systems, and distributed systems.

Ian Pratt received the Ph.D. degree in computer science and was elected a Fellow of King's College, Cambridge, in 1996.

He is currently a member of faculty at the University of Cambridge Computer Laboratory. As a member of the Lab's Systems Research Group for over five years, he has worked on number of influential projects, including the Fairisle ATM LAN, the Desk Area Network Workstation, and the Nemesis operating system. His research interests cover a broad range of Systems topics, including computer architecture, networking, and operating system design.

Christopher Clark received the B.A. degree in computer science as a Foundation Scholar of Queens' College at the University of Cambridge in 1998.

His interest in systems has been pursued through research investigating application of the Measure Project of the Systems Research Group at the Cambridge University Computer Laboratory. He is currently designing advanced network control systems for CPlane Inc., CA.

Simon Crosby (M'98) received the B.Sc. degree in math and computer science from the University of Cape Town, South Africa, the Masters degree in computer science from the University of Stellenbosch, South Africa, and the Ph.D. degree in computer science from Cambridge University.

He is co-founder and CTO of CPlane Inc., a Los Altos, CA-based startup developing a real-time service platform that delivers innovative communications-centric applications in multi-vendor, multitechnology networks. Before founding CPlane, he was a Lecturer at the Cambridge University Computer Laboratory, in the Systems Research Group. His research interests include network optimization, network control and routing, and network performance. He is chair of the Switch Control Working Group of the Multi-Service Switching Forum, an international standardization forum for converged services networks.