MPAT: Aggregate TCP Congestion Management as a Building Block for Internet QoS

Manpreet Singh, Prashant Pradhan* and Paul Francis
Cornell University, *IBM T.J. Watson Research Center
{manpreet,francis}@cs.cornell.edu, ppradhan@us.ibm.com

Abstract

Today Internet QoS is deployed piecemeal—typically at known bottleneck links like enterprise access links or wireless links. A more comprehensive, end-to-end QoS deployment, for instance across large enterprise networks or the global Internet, remains elusive. There is growing interest in the idea of using overlay networks to provide differential QoS services (improve performance for some flows at the expense of other flows). A necessary building block is the ability to provide differential service over a single overlay link that traverses many IP router hops. This paper presents MPAT, the first truly scalable algorithm for fairly providing differential services to TCP flows that share a bottleneck link. Unlike known schemes, our approach preserves the cumulative fair share of the aggregated flows even where the number of flows in the aggregate is large. Specifically we demonstrate, primarily through experiments on the real Internet, that congestion state can be shared across more than 100 TCP flows with throughput differentials of 95:1. This is up to five times better than differentials achievable by known techniques. Indeed, MPAT scalability is limited only by the delay-bandwidth product of the aggregated flows. With this tool, it is now possible to seriously explore the viability of network QoS through overlay network services.

1 Introduction

After the heady optimism of the vision of a universal QoS in the form of RSVP, we have come to understand that realistically QoS can only be deployed piecemeal. Rather than a single ubiquitous QoS architecture, what is evolving is an arsenal of tools that can be opportunistically deployed where benefits can clearly be demonstrated. These benefits are most clear where a bottleneck resource (e.g. a single bottleneck link) can be identified, there is an identifiable set of user or traffic classes that would benefit from arbitration of that bottleneck, and the organization with a vested interest in providing QoS has control over the bottleneck resource.

One example is an enterprise network using IP telephony over the Internet [17] , where the bottleneck is the access link between the enterprise locations and the Internet. Differentiation between voice and non-voice traffic provides clear benefits, and this traffic can be identified by a router, say by looking for RTP packets. The enterprise has control over the routers on either end of the access link, either directly or by coordinating with its ISP. Another common example is in wireless networking, either 802.11 or cellular, where there may be different types of traffic or different classes of users (gold, silver, bronze customers, or emergency services) vying for scarce radio spectrum. Here, QoS mechanisms may be implemented at several layers in the the protocol stack (for instance, [12]), and the wireless network provider controls these protocols.

In these diverse examples, a single organization (enterprise, wireless network provider) was able to apply a QoS tool (router queuing, radio access control) to an identifiable bottleneck resource (enterprise access link, radio spectrum). These examples represent the "low hanging fruit", if you will, of QoS tools—those cases where the opportunity and motivation are clear. What about the more difficult scenarios where there is an underlying network of some complexity, and no single organization has the motivation or wherewithal to implement QoS services in that network?

An interesting approach to creating a network with controllable properties, on top of an unstructured Internet, is the overlay network. Overlay networks have been produced for reducing network delays [15], improving network resilience [8], and providing QoS to network applications like games [9]. The basic building block for providing QoS over an overlay network is the ability to provide QoS over each logical overlay link. Flows between the two ends of an overlay link share the same underlying network path, thus sharing the bottleneck link. Hence mechanisms to arbitrate the bottleneck link bandwidth to provide QoS can be deployed at the ends of the overlay link.

Note that the question of whether overlay networks is a valid

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

approach to providing QoS is an open question, and this paper does not answer that question. In particular, if the overlay changes the characteristics of the individual TCP flows that pass through it, and if multiple overlays try to "compete" under these circumstances, then the overlays may lose their effectiveness, or worse may destabilize the Internet. What is clear, however, is that if we are to make progress in understanding what can legitimately be achieved with overlays, we need to understand the characteristics of a single hop in the overlay. More specifically, in order to co-exist with Internet traffic (and with other similar overlays), such a QoS mechanism should satisfy certain properties. We state these properties below, assuming there are N flows belonging to an overlay using a logical overlay link.

- *Fairness:* The overlay flows should not steal bandwidth away from other flows (overlay or non-overlay) sharing the same bottleneck. In particular, other TCP or TCP-friendly flows using that bottleneck link should get the same bandwidth as they would with N standard TCP flows using the overlay. This requirement ensures that the overlay's QoS mechanism is "transparent" to the background flows.

- *Utilization:* The overlay flows should be able to hold the fair share of the bottleneck link bandwidth entitled to them. In particular, the total bandwidth available to the overlay's flows should be equal to the total fair share of N TCP flows. This ensures that the overlay flows do not lose bandwidth to the background traffic.

- *Scalability:* The above properties should hold for large values of N.

The key technical contribution of this paper is an aggregate TCP congestion management scheme that scales well by the number of flows in the aggregate (we have demonstrated up to N=100 using experiments on the real Internet), that provides large differentiation ratios (we have demonstrated up to 95:1), and that does so fairly. We call the scheme MPAT, for Multi-Probe Aggregate TCP, because it maintains an active AIMD loop (i.e. a "probe") for every TCP flow in the aggregate. This can be contrasted with mulTCP[2], which tries to obtain N fair shares by running a single, more aggresive AIMD loop. A secondary contribution of this paper is to provide more experimental data and insights on the existing schemes (primarily mulTCP). Even though the scalability and stability of mulTCP is known to be limited, we feel it is important to provide this data for two reasons. First, contrasting MPAT with mulTCP allows us to clarify and verify our intuitions as to why MPAT performs well. Second, mulTCP, in spite of its stated limitations, is nevertheless still being proposed for overlay QoS [9] and so can be considered the incumbant. Thus it is important to make more direct comparisons between MPAT and mulTCP.

The rest of this paper is organized as follows. Section 2 discusses the existing techniques to provide QoS to TCP or TFRC flows over a single path or bottleneck link, and demonstrates that they fall short of meeting the desirable criteria for such a QoS mechanism. Section 3 presents the proposed aggregate TCP approach in detail. Section 4 presents a detailed performance evaluation of MPAT using experiments in controlled settings as well as over the real Internet, and compares MPAT with proposed techniques. Section 5 concludes the paper, detailing our key findings.

2 Existing Approaches

One class of approaches (e.g. pTCP [4]) that provide network QoS in a best-effort network, try to open extra simultaneous TCP connections, instead of a single connection, to give more bandwidth to an application. By contrast, MPAT does not create extra TCP connections. It tries to provide relative QoS among only the TCP flows that the applications normally create. The resulting behavior of schemes like pTCP is clearly not desirable, and in fact CM[6] proposes to explicitly forbid this kind of behavior by using one active AIMD loop (i.e. one congestion "probe") for multiple flows between two given endpoints. Further, this approach does not scale to higher performance ratios between flows, since a large number of flows active at a bottleneck lead to significant unfairness in TCP. [20]. Of course, the scalability of both MPAT and pTCP is limited by the delay-bandwidth product of the aggregated flows. But pTCP would reach this limit much before MPAT. As an example, we later show in Section 4.5 (using experiments on the real Internet) that *MPAT can give 95 times more bandwidth to one application over another using only three TCP connections*. On the other hand, schemes like pTCP would need to open 95 parallel TCP connections for this.

Key et. al. [7] propose to set different 'flow control' limits for various flows in order to provide different bandwidth to each flow. Please note that such schemes can only give lower bandwidth to a flow (as compared to its fair share), thus violating the *utilization* condition described in Section 1.

TCP Nice [3] is a way of providing only two-level QoS. Our scheme can provide multi-level QoS. While TCP Nice can give lower bandwidth to the background traffic, it cannot give higher bandwidth to the more critical traffic. Our system can give more bandwidth to one TCP flow at the expense of lower bandwidth to another flow, in such a way that the sum total of the bandwidth that the two flows get is same as the fair share of two TCP flows.

The idea of providing performance differentiation between

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

TCP flows sharing a path has also been discussed in the context of aggregate congestion management approaches like TCP Session [5], Congestion Manager (CM) [6], TCP Trunking [14], [13]. A TCP trunk is a TCP connection between two routers, which is shared by multiple TCP flows active between the two routers. The end-to-end flows terminate at the end points of the trunk, where data from all the flows is buffered. The bandwidth available on the trunk is the bandwidth available on the TCP trunk, which evolves using the standard TCP AIMD [19] algorithm. The data from various TCP flows can then be sent over the trunk according to any chosen scheduling policy. CM[6] is a congestion management scheme that provides unified congestion management to TCP as well as non-TCP flows, decouples congestion control from reliability, and ensures that end-points of a connection cannot hog more than their fair share of network bandwidth by opening multiple connections between them. CM defines an aggregate (termed "macroflow") as a set of TCP flows between a source and a destination host (termed "microflows"). Both CM and TCP Session keep one AIMD bandwidth estimation loop active per aggregate, and hence the bandwidth available to the aggregate is that entitled to one TCP flow. Microflows can share this bandwidth according to any chosen scheduling policy. Thus, in case of a logical overlay link, flows between the endpoints of the link could be aggregated into a CM macroflow or a TCP trunk, and performance differentiation could be provided between them.

However, by virtue of using one AIMD loop per trunk or per CM macroflow, both TCP Session and CM, in their current form, do not satisfy the *utilization* requirement. If N flows constitute an aggregate (trunk or macroflow), and the aggregate shares a bottleneck link with M background flows, then with equal sharing within the aggregate, the share of the bottleneck link bandwidth received by the aggregate's flows is $1/(N(M+1))$. If the N flows compete for bandwidth like standard TCP flows, each of the flows would be entitled to a share of $1/(M+N)$. This has also been mentioned in Chapter 7 of [5].

The above problem could be addressed by using a variant of the AIMD loop that acts like N TCP flows, as proposed in [5]. When a TCP flow has a bottleneck link where the loss probability p characterizes the congestion state at the link, TCP's AIMD algorithm allocates the flow a bandwidth given by $B = K/(RTT * sqrt(p))$, where K is proportional to $\sqrt{\frac{\alpha}{\beta} * (1 - \frac{\beta}{2})}$. To achieve performance differentiation among these flows, one possible approach is to play with the parameters α and β in the AIMD algorithm. This is the approach used by mulTCP [2]. The idea behind mulTCP is that one flow should be able to get N times more bandwidth than a standard TCP flow by choosing $\alpha = N/2$ and $\beta = 1/2N$. Thus, the congestion window increases by N (as opposed to 1) when a congestion window worth of packets is successfully acknowledged. Losses leading to fast retransmit cause the window to be cut by (1 - β), and losses leading to a timeout cut down the window to 1. Analytically, a mulTCP flow achieves a bandwidth N times that of a standard TCP flow experiencing the same loss rate. It is thus possible for CM or TCP trunking to use one mulTCP AIMD loop per aggregate to address the utilization problem. An equivalent TFRC [16] variant of mulTCP exists, where the throughput equation of mulTCP can be used in conjunction with a TFRC rate adjustment loop. A TFRC variant of mulTCP is used by OverQoS[9] to estimate the bandwidth entitled to N TCP flows on a logical overlay link.

Note however, that the loss process induced by a single mulTCP flow is quite different from that generated by N independent TCP flows. Hence, it is not clear if a mulTCP flow, especially for large N, would continue to behave like N TCP flows in the network. Also note that the 'amplitude' of mulTCP's AIMD control loop increases with N, leading to an increasingly unstable controller as N grows. This is in contrast to N control loops of N independent TCP flows, where each has a small amplitude and thus tends to be more stable.

Further, it has been noted in [2] that a mulTCP flow cannot act exactly like N independent TCP flows because timeout losses force the entire mulTCP window to be cut down to 1, whereas with N independent TCP flows, such a loss would cut down only one TCP connection's window to 1. In [2], it is shown that this limits the value of N for which a mulTCP flow can achieve as much bandwidth as N independent TCP flows (the recommended value for N is 4 [2]).

The above discussion indicates that QoS approaches based on mulTCP may not meet the fairness and utilization requirements for large N, in turn violating the scalability requirement. We experimentally investigated this hypothesis. Please refer to [1] for the detailed experimental analysis. In the next section, we present an aggregate congestion management scheme that satisfies the fairness, utilization as well as scalability requirements mentioned in section 1.

3 QoS through aggregate congestion management

3.1 An illustration using two flows

Before we discuss how our MPAT scheme exactly works, let us start with a simple example. Consider the following scenario: we have two TCP flows running simultaneously between the same end-hosts, thus sharing the same bottleneck link and experiencing similar delay. For ease of illustration, we refer to

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

the first flow as the *red* flow and the second flow as the *blue* flow. [1]

Our goal is to provide performance differentation among these two flows under the following constraints: Firstly, we should not affect other background flows running in the network at the same time (in terms of bandwidth, delay, loss rate, etc). This is referred to as the *fairness* property in Section 1. Secondly, the sum total of bandwidth acquired by the red flow and the blue flow should be equal to the fair share of two standard TCP flows going through the same bottleneck link at the same time. This is referred to as the *utilization* property in Section 1. The reason this problem is hard is that the fair share of a TCP flow keeps changing dynamically with time, depending upon the number of background flows going through the bottleneck link, and the end-hosts do not have an explicit indication of the amount of such cross-traffic.

Suppose we want to apportion the available bandwidth [2] among these two flows in the ratio 4:1. Assume that the congestion windows for each of the two flows is 5. Thus, the network allows us to send 10 packets every Round Trip Time (RTT). In the case of standard TCP, we would have sent 5 red packets and 5 blue packets every RTT. But the network does not really care how many red and blue packets we actually send, as long as the total number of red and blue packets is 10. If we *aggregate* the congestion information of both these flows at the end-host (the sender side), we can transmit 8 red packets and 2 blue packets every RTT, thus giving four times more bandwidth to the red flow over the blue flow. This would also make sure that we do not hurt the background traffic at all.

The above step allows us to split the instantaneous bandwidth among the two flows in the ratio 4:1. But will we be able to achieve that over a long period of time? The answer is NO, and the reason is as follows. Since the probability of each packet getting dropped in the network is the same, the red flow would experience a higher loss rate than the blue flow. This would force the red flow to cut down its congestion window more often than the blue flow. In the long run, the total bandwidth acquired by the two flows would be much less than the fair share of two TCP flows, thus violating the *utilization* property.

To overcome this problem, the fundamental *invariant* that we try to maintain at all times is that *the loss rate experienced by each congestion window should be the same as in standard TCP*. Standard TCP sends data packets, and receives congestion signals (either explicitly in terms of ECN, or implicitly in terms of duplicate acks, fast retransmissions, timeouts, etc) back from the receiver. If we had used standard TCP for each of the two flows, both the red window and the blue window would (on an average) experience equal number of congestion signals. In order to maintain this property, we first separate reliability from congestion control, as proposed in CM[6]. Next, *we decouple the actual growth of congestion window from the identity of the flow whose packets advance the congestion window*.

In the above example, when we get 8 red acks, we send three of these to the blue congestion window. In other words, we assign 5 acks to the red window, and 5 to the blue window (3 red + 2 blue). This ensures that each of the two congestion windows experiences similar loss rate, even though we can split the available bandwidth in the ratio 4:1. Please note that since we separate reliability from congestion control, it is the red flow that is responsible to maintain the reliability of each of the 8 red packets (in terms of buffering, retransmission, etc). When we get 8 red acks, we separate each of them into 'reliability ack' and 'congestion signal'. For the purpose of reliability, we send each of the 8 red acks to the red flow. But for the purpose of congestion control, we apply only 5 of these red acks to the red window, and apply rest 3 to the blue window. This ensures that the sum total of bandwidth acquired by both the flows is equal to the fair share of two TCP flows, thus satisfying the *utilization* property.

3.2 The general case

A source of N TCP flows sharing a bottleneck link is entitled to a total share of the link bandwidth given by the sum of the fair shares of each of the N TCP flows. The source should thus be able to arbitrate this total share among the N flows according to their performance requirements. Each TCP connection opened by an application has a corresponding control loop which uses the AIMD algorithm to adjust its congestion window in response to feedback from the network. The key idea behind our aggregate congestion management scheme is to *keep as many AIMD control loops active in the network as the number of TCP flows in the aggregate*, but to decouple application flows from their congestion windows. We call this scheme MPAT, for Multi-Probe Aggregate TCP, to emphasize the fact that we keep N AIMD control loops (i.e. "probes") active.

Thus, in an aggregate of N flows, the N application data streams are decoupled from the N congestion windows that are each evolving using the AIMD algorithm. The N AIMD control loops allow us to hold the fair share of N TCP flows, while an additional step of mapping application packets to congestion windows allows us to arbitrate the aggregate bandwidth to provide the desired QoS. Since the identity of pack-

[1] Please note that in contrast to schemes like pTCP, we are not intentionally opening these flows in order to hog more network bandwidth. These are the flows that applications normally create.

[2] In this example, by 'available bandwidth', we refer to the fair share of two TCP flows.

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

ets driving a given AIMD loop is irrelevant, this remapping is transparent to the network. Thus, the aggregate appears as N standard TCP flows to the network, providing appropriate fairness to any background TCP or TFRC traffic. Please note that this is different from all the existing schemes[6, 2, 14, 13], which keep only one bandwidth estimation probe active in the network, and hence suffer from problems of scalability and fairness.

The following section describes MPAT in detail.

3.3 The MPAT Algorithm

Consider N TCP connections running simultaneously as part of an aggregate, sharing the same bottleneck link in the network. Let these flows be labeled as f_i, $1 \leq i \leq N$. Let C_i represent the congestion window of flow f_i. We introduce another variable A_i which denotes the MPAT window for flow f_i. Let C denote the aggregate congestion window, given by the sum of the congestion windows of all flows. Let x_i denote the fraction of the total bandwidth allocated to flow f_i, such that $\sum_i x_i = 1$. The shares x_i are derived from the performance requirement of the flows f_i, and could change dynamically with time.

Note that C represents the product of the bandwidth available to the aggregate and the round-trip delay (RTT) on the logical overlay link. While TCP would allocate C among the N flows roughly equally in every RTT, MPAT would allocate C in accordance with the performance requirements of the flows. In other words,

$$A_i = x_i * C \tag{1}$$

The actual number of packets that flow f_i is allowed to send every RTT is $min(A_i, W_i)$, where W_i is the minimum of the sender and receiver window sizes for flow f_i. With standard TCP, flow f_i would be allowed to send $min(C_i, W_i)$ packets every RTT.

C_i	Congestion Window for flow i
A_i	MPAT Window for flow i
C	$\sum_i C_i$
x_i	Bandwidth share of flow i
W_i	Minimum of sender and receiver window size for flow i

Table 1: Symbols and their meanings

Each connection i maintains a mapping of the sequence numbers of its packets to the identifier of the congestion window through which the packets were sent. We refer to this mapping as $seqno2id_i()$. This mapping is maintained as a list ordered by increasing sequence number. For each congestion window i, we also maintain the inverse of this mapping, i.e. the list of packet sequence numbers sent through this window. We refer to this mapping as $id2seqno_i()$. This mapping is also maintained as a list ordered by increasing timestamps (i.e., the time at which the packet was sent). We use a variable per congestion window that keeps track of the number of outstanding packets that have been sent through that window.

3.3.1 Transmit Processing

Whenever connection f_i sends out a packet with sequence number s, it tries to find a connection (say j) with open congestion window. Congestion window j is said to be open if the number of outstanding packets sent through it is less than C_j. Note that the packets sent using congestion window j could belong to f_j, or to any other flow in the aggregate. In practice, j could be the same as i. Connection i then stores the mapping of seqno s to congestion window j in $seqno2id_i()$. We also store the inverse mapping in $id2seqno_j()$

3.3.2 Receive Processing

Our implementation is based on TCP-SACK. When we receive an acknowledgement for a packet with sequence number s, belonging to flow f_i, we use the mapping $seqno2id_i()$ to find the congestion window j through which the packet was sent. We then look at the inverse mapping $id2seqno_j()$ to find if the ack received is in sequence or not. We then apply the standard AIMD algorithm to congestion window j. Thus, if the ack is in sequence, we linearly increase C_j to C_j + 1/C_j. If it is a duplicate ack (in case of SACK, this would be the ack for a later packet in sequence), we enter the fast retransmit phase, halving C_j if we get three duplicate acks (in case of SACK, three out-of-sequence acks).

3.3.3 Mapping sequence numbers to congestion windows

The choice of the mapping of a sequence number to a congestion window is critical, since it affects the number of fast retransmit-induced window halvings that an MPAT aggregate receives over a given interval of time, and hence the total bandwidth gained by it. Recall that standard TCP-SACK considers multiple losses within a single congestion window as one loss event for fast retransmit, and hence cuts down its window by half only once. To understand how this affects MPAT, consider the following example: There are two TCP flows running simultaneously as part of an MPAT aggregate. Connection 1 has a window size of 20, and has packets 1 through 20 outstanding in the network. Suppose packets 8 and 9 get dropped. Since both these packets belong to the same window, standard TCP-

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

SACK would treat them as one signal for fast retransmit, and hence cut down its window only once.

MPAT could treat this situation as either one or two fast retransmit signals, depending upon the mapping $seqno2id_1()$. Consider the case when sequence numbers 8, 10-12 are mapped to C_1, and sequence numbers 9, 13-15 are mapped to C_2. The acks for packets 10-12 will be charged to C_1, which will treat all three of them as duplicate acks for packet 8, and hence cut down its window by half. But the acks for packets 13-15 will be charged to C_2, which will again treat them as duplicate acks for packet 9, and hence cut down its window by half. This example shows how two packet losses within the same window could lead to two fast retransmit signals. On the other hand, if both packets 8 and 9 were mapped to C_1, this would lead to only one fast retransmit signal. In this case, flow 2 would treat acks for packets 13-15 as if they were received in sequence, and hence would *not* reduce its window.

We tried out three different algorithms to map the packet with sequence number s, belonging to flow f_i, to a congestion window :

- If sequence number $s-1$ was sent through congestion window k, send the current packet s through window k if it is open. Else, randomly pick an open congestion window to send the packet. This algorithm turns out to be aggressive to the background flows since consecutive packet losses are charged to the same congestion window, and get absorbed into one loss event for fast retransmit, reducing the number of window reductions.

- If sequence number $s-1$ was sent through congestion window k, randomly choose an open congestion window different from k. This scheme turns out to be more conservative than standard TCP, since it forcibly maps multiple losses on a flow to different congestion windows, causing each one of them to be cut down.

- Map s to an open congestion window picked uniformly at random. This option turns out to be a reasonable compromise between the two extremes. In fact, it turns out to be conservative when one of the flows has a very large share of the aggregate congestion window, since it still spreads out the losses of packets on that flow to multiple congestion windows. We use this option in our implementation.

Whenever any of the congestion windows in the aggregate changes, the aggregate congestion window also changes, and hence we must also update the MPAT windows A_i of all connections using equation 1. Note that this means that whenever the congestion window for one of the flows is cut down, the loss in available bandwidth gets distributed proportionally among the MPAT windows of all the flows. This has the effect of reducing the inter-connection variance in throughput, as we shall see later in section 4.6.

3.3.4 TFRC variant

There exists a TFRC[16] variant of MPAT, in which instead of N AIMD control loops, we have N rate adjustment loops, each driven by the standard TCP throughput equation for one TCP flow. Note that loss probabilities will be measured by congestion window, i.e. over data sent using a given rate adjustment loop, irrespective of which application flow the packets came from. Loss events will be mapped to congestion windows in the same way as described above. The TFRC variant enjoys the same transparency properties as the AIMD version, in that each rate adjustment loop is oblivious to the identity of the packets driving it. As a result, the TFRC variant of MPAT appears as a set of N independent TFRC flows to the network. All of our experiments in this paper are with the AIMD version of MPAT.

3.4 Implementation

We have prototyped our proposed aggregate congestion management scheme in Daytona , a user-level TCP stack running on Linux. Daytona can be linked as a library into an application, and offers the same functions as that provided by the UNIX socket library. Daytona talks to the network using a raw IP socket, thus avoiding any processing in the operating system's TCP stack. All TCP processing is done at user-level, and pre-formed TCP packets are handed to the IP layer to be sent over a raw socket.

Our user-level implementation allowed us to run wide-area experiments on diverse locations in the Internet, as well as in controlled public network testbeds like Emulab [10], without requiring control over the operating system.

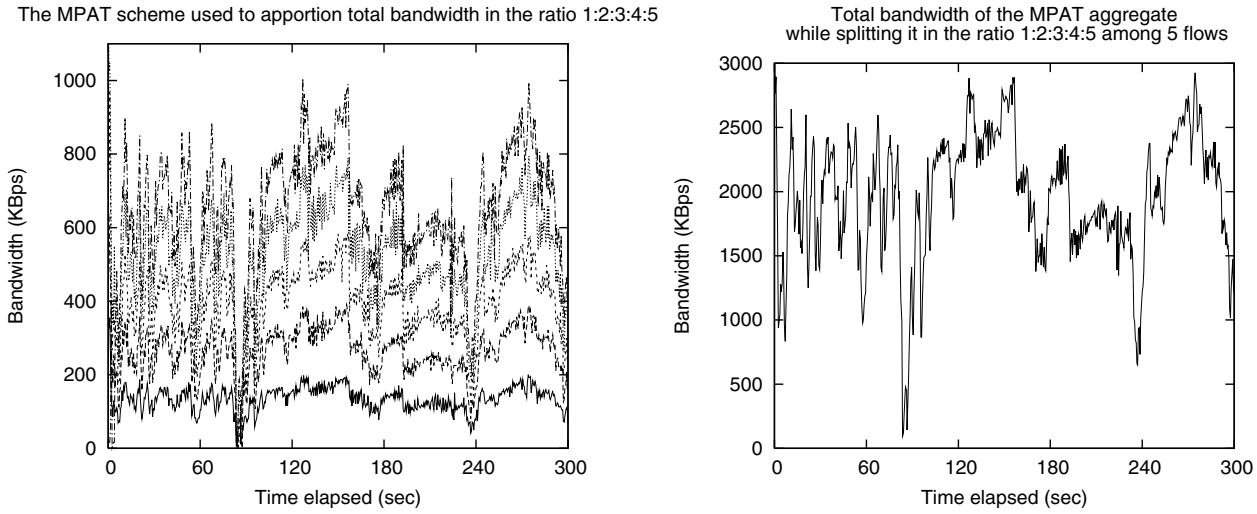
A key goal in MPAT is to ensure that we hold the fair share of N TCP flows in the process. Thus, each AIMD control loop should be kept running as long as there are packets available to drive it. Note that if a flow with a high share of the aggregate congestion window gets send or receive window limited, or has a period of inactivity, then it may leave some of its share of the aggregate congestion window unused. Left alone, the aggregate would lose this unused share to the background traffic. By using a work-conserving scheduler on the aggregate, however, we make sure that other flows in the aggregate take up such unused bandwidth, if they have data to send.

4 Evaluation

In this section, we conduct extensive experimentation to evaluate the fairness, utilization and scalability properties of MPAT.

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY



(a) MPAT can apportion bandwidth among its flows, irrespective of the total bandwidth available.

(b) The MPAT aggregate can hold its total fair share

Figure 1: MPAT can apportion bandwidth in the desired ratio, while holding its total fair share.

We have conducted experiments both in controlled settings, and over the Internet under stable conditions, reverse traffic, and transient congestion. We built our own wide-area network testbed consisting of a network of nodes in diverse locations (including US, Europe and Asia). Our main goal in choosing these nodes is to test our system across wide-area links which we believe have losses. For this reason, we made sure that the whole path is not connected by Internet2 links (known to have very few losses). We changed the routing tables of our campus network to make sure that the path to our destination did not take any of the Internet2 links. We also did experiments on the Emulab network [10], which provides a more controlled setting for comparing our scheme with other approaches. All of our experiments used unconstrained send and receive windows on both ends of the TCP connections, so that flow control does not kick-in before congestion control. As background traffic we use long lived TCP connections and bursty web traffic. We had a Maximum Segment Size (MSS) of 1460 bytes in all our experiments. Due to lack of space, we omitted some of the graphs. Please refer to [1] for detailed experimental results.

4.1 Scalability

Using experiments on both the Emulab network and the real Internet, we found that the MPAT scheme can hold the fair share of 100 TCP flows running simultaneously as part of an aggregate, irrespective of how the aggregate bandwidth was apportioned among the aggregate's flows. The limits on scalability beyond 100 flows arise due to other factors like minimum amount of bandwidth needed per connection, bottleneck shifting from network to memory, etc.

As noted in section 2, mulTCP scales only up to 20-25 flows in the Emulab setting. In a real Internet experiment, we found mulTCP to scale upto 10 to 15 flows.

4.2 Performance differentiation

The MPAT scheme can be used to apportion the total available bandwidth among different flows in any desired ratio. Even though the total fair share allocated to the aggregate keeps changing with time, MPAT can maintain the target performance differential at all times. We could give a maximum of 95 times more bandwidth to one TCP flow over another.

The real limits to scalabiliy for large performance differential arise from the fact that every connection needs to send a minimum of 1-2 packets every RTT. In the future, we plan to change this to 1 packet every k RTTs, as proposed in [3].

As an example, Figure 1(a) shows the bandwidth allocated to five TCP flows running as part of an MPAT aggregate between Utah and our campus network on the east coast (RTT approximately 70 msec). There were five more long-running standard TCP flows in the background (in both directions). We tried to split bandwidth in the ratio 1:2:3:4:5. The graph is for a period of 5 min, with data samples taken every 500msec (ap-

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE
IEEE COMPUTER SOCIETY

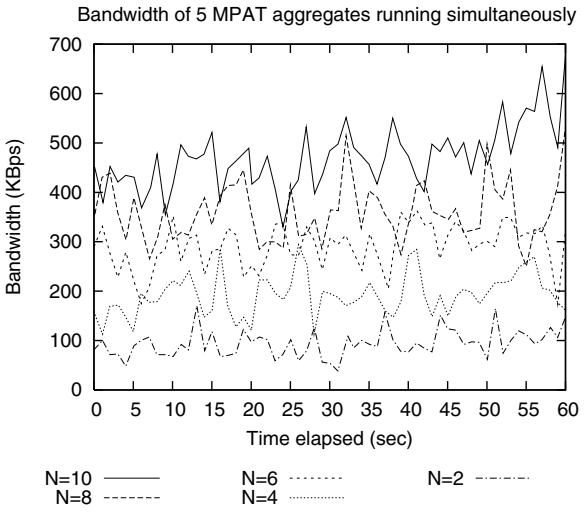


Figure 2: Competing MPAT aggregates cooperate with each other.

prox 7 RTTs). The average amount of bandwidth that each of the five flows got was 135 KBps, 265 KBps, 395 KBps, 530 KBps and 640 KBps respectively. We can see that the MPAT scheme is very effective at maintaining the desired target differential among different flows at all times, irrespective of the total share of bandwidth that the aggregate is entitled to. The average bandwidth of a standard TCP flow running in background during this time period was 400 KBps. During the process of splitting bandwidth, the MPAT aggregate holds its total fair share of approximately 2000 KBps, as seen in Figure 1(b), thus satisfying the *utilization* criterion.

4.3 Fairness to background flows

As noted earlier, an MPAT aggregate is naturally fair to background TCP traffic since the remapping of application packets to congestion windows is transparent to the N AIMD control loops in the aggregate, which means that the network effectively sees N standard TCP flows.

4.4 Interaction between multiple MPAT flows

To analyse the behavior of multiple competing virtual overlay links using MPAT, we study the bandwidth achieved by multiple MPAT aggregates running simultaneously. Again, as noted earlier, the behavior of an MPAT aggregate with N flows is similar to N independent TCP flows running without any aggregation. This suggests that each MPAT aggregate should be able to get a share of the bottleneck bandwidth depending upon the number of flows in each aggregate, and the number of background TCP flows.

To verify this hypothesis, we ran 5 MPAT aggregates, each with a different value of N, together with 5 background TCP flows between Utah and our campus network on the east coast. The five MPAT aggregates use N = 2, 4, 6, 8 and 10. We then replaced each aggregate with the same number of independent TCP flows. The experiment was repeated during different times of day. Figure 2 shows the bandwidth for each of the five MPAT aggregates, with data points sampled every 1sec. We measured the number of fast retransmits, number of timeouts and bandwidth for each flow within the MPAT aggregate over a period of 390 seconds. We also measured the corresponding data for the non-aggregated case. Please refer to [1] for the detailed experiment numbers. In both cases, we found the achieved bandwidth and loss rates seen by the background TCP flows to be similar. The bandwidth obtained by each aggregate is always proportional to the number of flows it had, and when the flows within an aggregate were run independently, the sum of bandwidths achieved by these flows was similar to the bandwidth achieved by the aggregate. This shows that MPAT is not a selfish scheme. Competing MPAT flows cooperate with each other, and with the background traffic. The experiment also illustrates that MPAT adequately satisifies the utilization property.

4.5 Adaptation to changing performance requirements

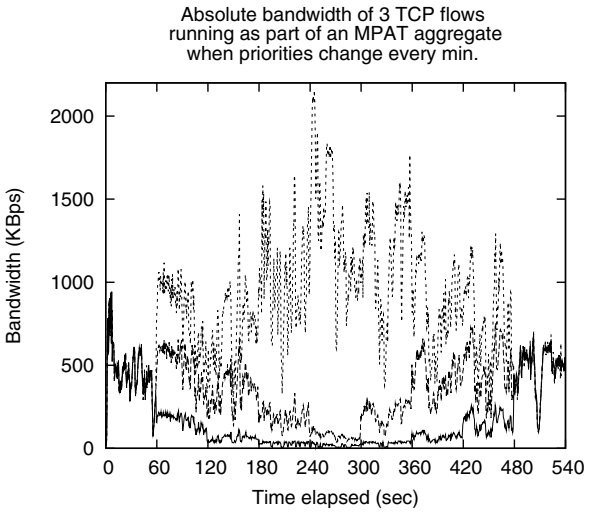
When the performance requirements of various flows within an aggregate change, our scheme simply needs to update the MPAT windows A_i for all flows using equation 1, and thus should be able to respond very quickly (typically 2-3 RTTs) to dynamically changing performance requirements with time.

To test this claim, we ran an MPAT aggregate consisting of three flows between Utah and our campus network on the east coast, with 5 more long-running standard TCP flows in the background (in both directions). We changed the performance requirements every minute. Figure 3(a) shows how the absolute bandwidth for each of the three connections varies with time for a period of nine minutes. Data points have been sampled every 500 msec (approx 7 RTTs). Figure 3(b) shows relative bandwidth of the three flows at all times, along with the target performance differential. We can see that MPAT very quickly adapts to changing performance requirements. Note that during the time interval t = 240 to t = 300 sec, we were able to split bandwidth in the ratio 95:1.

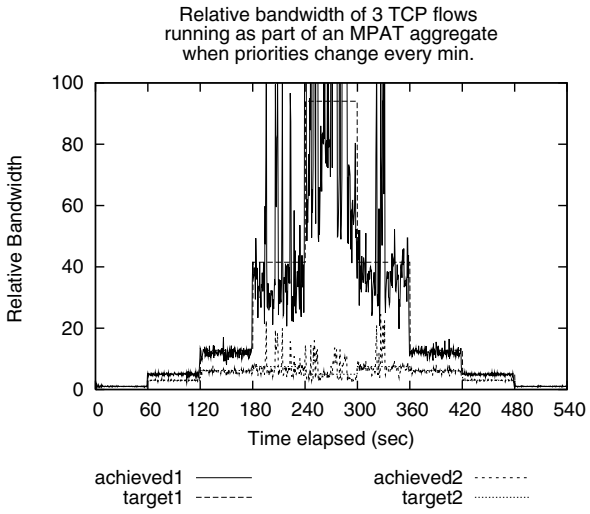
4.6 Reduced variance

The total bandwidth of an MPAT aggregate is equivalent to the sum of N independent standard TCP flows, each of which uses

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE
IEEE COMPUTER SOCIETY



(a) This figure shows how the absolute bandwidth achieved by flows that are part of an MPAT aggregate (N=3) adapts itself when priorities keep changing dynamically with time.



(b) This figure shows how the relative bandwidth achieved by flows that are part of an MPAT aggregate (N=3) adapts itself when priorities keep changing dynamically with time.

Figure 3: MPAT Scheme can adapt itself very quickly to dynamically changing performance requirements.

an AIMD control loop. Multiplexing of these control loops smooths out the bandwidth variations within each AIMD loop, thus reducing the variance in throughput of an MPAT aggregate drastically. This is in contrast to mulTCP that exhibits increasing variance with N. We verified this using experiments on the real Internet. Please refer to [1] for detailed experiment graphs.

4.7 Adaptation to Transient Congestion

The MPAT scheme must be robust to transient congestion in the network, and more generally, to variations in the fair share of bandwidth available to the aggregate. With transient congestion, an MPAT aggregate must learn about a change in its total fair share using the AIMD algorithm. The change is apportioned among the aggregate's flows quickly through an adjustment of the MPAT windows A_i of all flows, ensuring that the desired bandwidth ratio is maintained at all times.

To study the responsiveness of MPAT to transient congestion, we conducted experiments on the Emulab network over an emulated link of bandwidth 90 Mbps and RTT 50msec. We ran an MPAT aggregate consisting of 10 flows with different performance requirements. Transient congestion was created by introducing constant bit rate (CBR) as well as TCP traffic in the background. We also used 10 long-running standard TCP flows in both directions. At t=120 sec, we introduce a CBR flow in the background with rate 30Mbps. Please refer to [1] for the detailed graphs. MPAT cuts down its total bandwidth in about 4-6 RTTs (200-300 msec), while still apportioning bandwidth within the aggregate in the desired ratio at all times. At t=240sec, we removed the background CBR traffic, and MPAT reclaims its fair share of the remaining bandwidth.

When background TCP traffic was introduced to create transient congestion, it takes about 15-20 (1 sec) RTTs for both MPAT and the background TCP flows to settle into their respective fair shares This is because the new TCP flows begin in slow-start mode, and are also adapting to their fair share together with MPAT. As earlier, MPAT always apportioned bandwidth within the aggregate in the desired proportion.

4.8 Bursty background traffic

Most of the results we described above are for long-running TCP flows in the background. We also tested our scheme with bursty web traffic running in background. We did this using a *wget* loop downloading web pages in the background, together with MPAT. While bursty traffic increased the absolute number of retransmits and timeouts *proportionally* for all aggregates and the background traffic, we did not see any qualitative change in the results. In other words, MPAT exhibited the same scaling, fairness and utilization properties with bursty traffic.

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY

5 Conclusions

The paper demonstrates for the first time the viability of providing differential services, at large scale, among a group of TCP flows that share the same bottleneck link. We demonstrate this through a range of experiments on the real Internet which show that an MPAT aggregate can hold its fair share of the bottleneck bandwidth while treating other flows fairly (either individual TCP flows or other MPAT aggregates). We also demonstrate that within an aggregate, MPAT allows substantial differentiation between flows (up to 95:1). This suggests that MPAT is therefore an appropriate candidate technology for broader overlay QoS services, both because MPAT provides the requisite differentiation and scalability, and because MPAT can co-exist with other TCP flows and with other MPAT aggregates. This result opens the door to experimentation with more easily deployable network QoS schemes such as the overlay.

Having said that, this paper does not make the broader conclusion that overlay QoS works. In order to do that, we must understand the end-to-end behavior of flows (TCP and otherwise) over a multihop overlay. This means among other things that we must understand the interactions between the individual overlay hops (be they TCP or TFRC) of the same overlay, between aggregates on different overlays, and between all of these and the end-to-end flow control of the TCP connections running over the overlay. It is also important to understand how an overlay QoS could be combined with RON-type overlay functionality used to actually enhance performance (not just provide differentiation).

There may also be other uses for MPAT aggregation. For instance, in the back end systems of a web service data center, MPAT could be used to provide differential service among different kinds of customers (gold, silver, bronze). This is possible in part because MPAT only needs to be deployed at the sending end of a TCP connection. For this to work, however, the TCP flows must share a bottleneck. This in turns requires that the server implementing MPAT can determine which flows, if any, share the bottleneck. These possible uses of MPAT provide exciting avenues for future research.

References

[1] Manpreet Singh, Prashant Pradhan and Paul Francis, *MPAT: Aggregate TCP Congestion Management as a Building Block for Internet QoS*, IBM Research Technical Report, http://www.cs.cornell.edu/∼manpreet/mpat-qos.pdf

[2] J. Crowcroft and P. Oechslin, *Differentiated End-to-End Internet Services using a Weighted Proportional Fair Sharing TCP*, ACM Computer Communication Review, 28(3):53-69, July 1998.

[3] A. Venkataramani, R. Kokku and Mike Dahlin, *TCP NICE: A mechanism for background transfers*, OSDI 2002.

[4] H. Y. Hsieh, K.H. Kim and R. Sivakumar, *On Achieving Weighted Service Differentiation: an End-to-end Perspective*, IWQoS 2003.

[5] V. Padmanabhan, *Addressing the Challenges of Web Data Transport*, Ph.D. Thesis, University of California at Berkeley, Sept 1998. http://www.cs.berkeley.edu/∼padmanab/phd-thesis.html

[6] H. Balakrishnan, H. S. Rahul and S. Seshan, *An Integrated Congestion Management Architecture for Internet Hosts*, SIGCOMM 1999.

[7] P. Key, L. Massoulie and B. Wang, *Emulating Low-priority Transport at the Application Layer: A Background Transfer Service*, Sigmetrics 2004

[8] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris, *Resilient Overlay Networks*, SOSP 2001.

[9] L. Subramanian, I. Stoica, H. Balakrishnan and R. Katz, *OverQoS: An Overlay Based Architecture for Enhancing Internet QoS*, NSDI 2004.

[10] Emulab test-bed. http://www.emulab.net

[11] Planetlab test-bed. http://www.planet-lab.org

[12] 3rd Generation Partnership Project 2 (3GPP2), *Quality of Service: Stage 1 Requirements,* 3GPP2 S.R0035, www.3gpp2.org

[13] A. Bestavros and O. Hartmann, *Aggregating Congestion Information Over Sequences of TCP Connections*, In BUCS-TR-1998-001, Boston University, Dec 1998.

[14] H.T. Kung and S.Y. Wang, *TCP Trunking: Design, Implementation and Performance*, ICNP 1999.

[15] S. Savage, *Sting: a TCP-based network measurement tool*, USITS 1999.

[16] S. Floyd, M. Handley, J. Padhye and J. Widmer, *Equation-based Congestion Control for Unicast Applications*, SIGCOMM 2000.

[17] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang and W. Weiss, *An architecture for differentiated services*, RFC 2475, Oct 1998.

[18] J. Padhye, J. Firoiu and J. Kurose, *Modelling TCP Throughput: A Simple Model and its Empirical Validation*, SIGCOMM 1998.

[19] D. Chiu and R. Jain, *Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks*, Journal of Computer Networks and ISDN Systems, 17(1):1-14, June 1989.

[20] D. Lin and R. Morris, *Dynamics of Random Early Detection*, SIGCOMM 1997.

Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04)
1092-1648/04 $ 20.00 IEEE

IEEE COMPUTER SOCIETY