

Using Differentiated Services Mechanisms to Improve Network Protocol and Application Performance

Venkata N. Padmanabhan
Microsoft Research
padmanab@microsoft.com

Abstract

The differentiated services architecture (diffserv) includes lightweight mechanisms for service differentiation in the Internet. The primary motivation for diffserv has been enabling differentiation at a coarse granularity such as individual ISP customers. In this paper, we argue that the same or similar lightweight mechanisms could be employed fruitfully at a much finer granularity such as individual packets. The key benefit this would offer is that the sending host could exercise control on a much finer time-scale than traditional end-to-end mechanisms permit. We discuss several instances in the context of end-to-end protocols and applications where such enhanced control could help improve performance. We focus specifically on the TCP protocol and the Web browsing application, neither of which has received much attention in the traditional QoS research literature.

1 Introduction

Recently considerable effort has gone into developing an architecture for differentiated services in the Internet (*diffserv* [BBC+98]). This effort has been motivated, on the one hand, by the need for service differentiation as the Internet increasingly becomes commercialized and, on the other hand, by the sluggish deployment of integrated services (*intserv* [BCS94]). A salient feature of *diffserv* is its scalability, which is achieved by moving much of the complexity out of the core of the network into edge devices that process lower volumes of traffic. End hosts or edge devices set the *diffserv* bits (*DS bits*) in the IP header of packets. These bits then invoke specific *per-hop behaviors* (*PHBs*) at routers in the network interior. Examples of *PHBs* include *priority dropping* (with FIFO queuing) and *priority scheduling*. The key efficiencies of *diffserv* compared to *intserv* are that the service offering is on aggregated traffic rather than on a per-flow basis (i.e., all packets with a similar setting of the *DS bits* are treated similarly), the routers do not maintain per-flow state, and signaling happens only on a slow “administrative” timescale rather than on a flow timescale (which is in contrast to RSVP [BZB+97]).

Much of the attention thus far has been focussed on how *diffserv* may be used by Internet Service Providers (ISPs) to provide differentiated service at a coarse granularity such as individual customers. In this paper, we take the position that the same mechanisms can also be used for a very different purpose, i.e., to improve the performance of network protocols and applications. The key benefit offered by the *diffserv* mechanisms is that the sending host can exercise control on a much finer timescale (i.e., on the order of the queuing delay at a router) than traditional end-to-end mechanisms permit (i.e., on the order of the network round-trip time (RTT)). Network protocols and/or applications could control the setting of the *DS bits* on a packet-by-packet basis with a view to optimizing performance. We argue that this could benefit not only real-time applications (such as streaming audio and video) but also protocols/applications (such as TCP and Web browsing) that have traditionally been considered to be “non-realtime” and hence have not been the subject of much QoS research.

We begin in Section 2 by discussing the use of *diffserv* in the context of transport protocols, in particular TCP. Then, in Section 3, we discuss applications, primarily Web browsing, that could benefit from *diffserv*. In Section 4, we discuss some general issues and concerns pertaining to our proposal. We conclude in Section 5.

Note that when we refer to *diffserv* in the remainder of this paper, we are not necessarily restricting ourselves to the specific proposals under discussion by the IETF. Rather, we use the term to loosely refer to the set of lightweight service differentiation mechanisms that are similar in spirit to the IETF proposals.

2 Transport Protocols

The task of a transport protocol is to transfer data end-to-end from a source host to one or more destination hosts. In doing so in the context of the Internet, the protocol has to contend with the shared nature of the network and the inherently unreliable data delivery service provided by the IP layer. For this reason, transport protocols employ *congestion control* and *loss recovery* mechanisms.

Since TCP is by far the most widely used transport protocol in the Internet, it is the focus of much of our discussion. Since its inception, TCP has been used for a wide range of applications (bulk transfers, transactions, etc.) and over a wide variety of networks (dialup, cable modem, etc.). In some instances, there has been a strain on TCP's congestion control and loss recovery mechanisms [Jac88], leading to poor performance. We now discuss how some of these problems can be alleviated using *diffserv*.

2.1 Short Transfers and TCP Fast Start

One of the challenges posed by short transfers, such as Web transfers, is that TCP's congestion control algorithm does not have the opportunity to probe the network for long enough to determine the available network bandwidth. A transfer may terminate while the congestion window is still quite small, thereby under-utilizing bandwidth. While persistent-connections [PM94] in the context of the Web and HTTP do help, connection lengths still tend to be short (e.g., [Mah97] reports an average length in the range 26-32 KB) and there is the *slow-start restart* problem [Hei97] when the persistent connection resumes activity after an idle period.

We have proposed *TCP fast start* [Pad98,PK98] as a possible solution to this problem. The basic idea is that the TCP sender reuses congestion information gathered in the recent past to avoid repeated slow start. To avoid a large burst of packets, the sender uses timers to space packets apart. TCP fast start can speed up short transfers significantly. For instance, if the cached window size is 25 KB, a 20 KB transfer can be completed in less than one RTT. In contrast, standard slow start would require at least 5 RTTs (and even more with delayed acks).

A fundamental problem, however, is that the cached congestion information may have been invalidated by changes in the network conditions. For example, "flash crowds" may cause an uncongested network to become heavily congested within a short period of time. So a fast start attempt during the congested period that draws upon information cached during the uncongested period could cause heavy packet loss, thereby degrading performance significantly both for the connection itself and for competing traffic. (Alternative schemes proposed in [Tou97] and [VH97] suffer from a similar problem.)

Our proposal is to have the sender use the DS bits to mark the fast start packets (except for the first one, which would have been sent by slow start anyway) as low priority for the purposes of dropping. In the event of congestion, routers would preferentially drop the fast start packets compared to regular traffic, thereby shielding the latter from the ill-effects of an over-aggressive fast start. As shown in Figure 1, priority dropping ensures that under adverse circumstances (i.e.,

when the network transitions rapidly from an uncongested state to a congested state), fast start impacts other traffic much less than it would otherwise have.

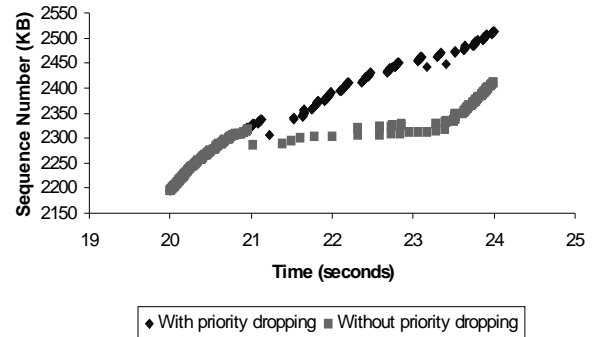


Figure 1 The impact of fast start on a competing connection when priority dropping is in use (upper curve) versus when it is not in use (lower curve).

However, we have addressed only part of the problem. The loss of (several) fast start packets could cause the connection that is attempting fast start to itself suffer significant performance degradation. (This could happen because TCP's loss recovery mechanisms perform poorly in the face of a burst loss.) To alleviate this problem, the sender aborts fast start *immediately* upon detecting the loss of multiple packets (i.e., it does not wait for a retransmission timeout). It falls back on standard slow start (with an initial window size of 1 segment), *without* incurring any of the usual congestion control penalties associated with packet loss (e.g., timer backoff, halving of the slow start threshold, etc.). The reason such an aggressive recovery is permissible is that priority dropping would have ensured that the original fast start attempt, although based on stale information, did little damage to the network at large (as illustrated in Figure 1). So it is okay for the sender to "forget" that it ever attempted fast start and to fall back on the default behavior, i.e., slow start, without incurring a penalty.

Our experiments in [Pad98] show that fast start coupled with priority dropping performs well under a wide range of conditions. The fundamental reason is that priority dropping relieves the sender from having to be overly conservative. The sender can use cached congestion information and yet be confident of significant latency reduction when the information is valid (up to 3X in our experiments) and little degradation when it is stale.

2.2 Coping with Packet Loss

We now turn to problems that occur when a TCP connection suffers packet loss during vulnerable periods, and discuss how *diffserv* may help alleviate these problems.

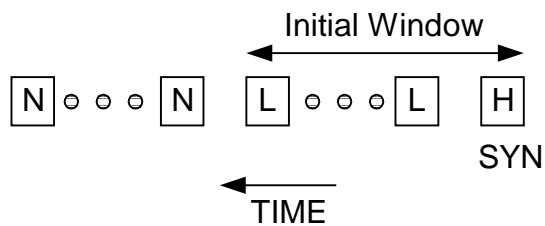


Figure 2 Packet priorities when TCP fast start is used in conjunction with a high-priority SYN packet (Section 2.2).

Short connections are particularly vulnerable to packet loss because often an insufficient number of duplicate acknowledgements is generated to trigger TCP fast retransmission. So the sender typically has to fall back on expensive retransmission timeouts. The situation is similar during the start up phase of a long connection. This problem can be alleviated by assigning the vulnerable packets a high drop priority, so that their chances of being dropped are minimized.

The initial SYN packet of a connection is particularly vulnerable because of the rather large timeout period (often 6 seconds [Ste94]) used for connection establishment. So it might be worthwhile to accord the SYN packet a high drop priority (even if the other packets in the initial window are not).

Using the letters H, L, and N to denote high, low and normal (default) drop priorities, a combination of the policies advocated in this section and TCP fast start might result in a packet sequence such as: HL...LN...N, i.e., a high-priority SYN packet, followed by low-priority fast start packets (in the initial window), followed by normal packets (in subsequent windows). This is illustrated in Figure 2. If an initial window size larger than 1 segment is deemed appropriate [AFP98], the sequence would be more like H...HL...LN...N.

Finally, retransmitted packets are also quite vulnerable because the loss of a retransmission often leads to a timeout. So it may be beneficial to assign a high drop priority to such packets as well.

2.3 Alternative Congestion Response

TCP's response to implicit/explicit congestion notifications (packet loss, ECN, etc.) is to decrease the offered load by shrinking its window significantly. Since window growth, especially during the linear phase, can be quite slow, a drastic reduction in the window size may be undesirable if the congestion is, in fact, due to a momentary traffic spike, say due to a UDP burst. It may be better for the sender to *in-effect* decrease the offered load by lowering the drop priority for a fraction of its packets while still maintaining its window at a similar size as before. If sustained congestion is observed, the sender can then shrink its window, just as standard TCP would.

In some sense, this approach buys us the best of both worlds. If congestion is due to a momentary spike, the connection would perform as though it had never reacted to the congestion notification. If congestion is sustained, the sender's action would immediately contribute to alleviating the congestion (for the normal and higher priority packets).

2.4 Asymmetric-Bandwidth Networks

Asymmetric-bandwidth access networks, such as those based on cable modems and ADSL, are becoming increasingly popular because they represent a cost-effective means of providing high-bandwidth connectivity in the direction of interest, i.e., towards users. However, as we have reported in [BPK97,Pad98], the presence of upstream traffic can degrade downstream throughput significantly even when the high-bandwidth downstream channel is uncongested. The problem is that the queuing of data packets at the upstream link could hold up acks that are critical to sustaining the downstream transfer. This is illustrated in Figure 3.

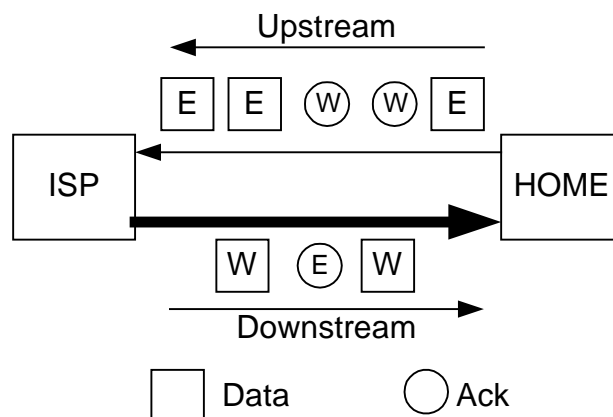


Figure 3 An asymmetric-bandwidth network with an email transfer (E) in the upstream direction happening concurrently with a Web download (W) in the downstream direction. With FIFO scheduling, large email data packets hold up acks of the Web connection, thus slowing down the download significantly.

As a solution, we proposed that acks be scheduled at a higher priority than data packets at the upstream router (*acks-first scheduling*, which can be implemented using *diffserv*). This can lead to a very significant improvement in performance, as reported in [Pad98]. For instance, with a 10 Mbps downstream link and a 28.8 Kbps upstream link, the download time for a 40 KB Web page decreases from 12.4 seconds to 3.67 seconds.

2.5 Use of Diffserv in Other Contexts

Although much of our discussion has been couched in terms of TCP, the fundamental reason why *diffserv* is beneficial is, in many instances, independent of the specific protocol. For example, the repair request and response packets in Scalable Reliable Multicast (SRM) [FJL+95], are critical to quick loss recovery (just as retransmitted packets are in TCP), so it may be beneficial to assign them a high drop priority.

We recently became aware of a unicast transport protocol called Blitz [Kim95] that uses *diffserv*-like mechanisms to implement non-feedback-based congestion control for high-speed networks. This is similar in flavor to our TCP fast start work but in a non-TCP context.

There have also been proposals to use *diffserv* mechanisms to provide probabilistic rather than deterministic differentiation. Examples include ERED [FKS+97] and RIO [CW97] in the context of RED queue management [FJ93] where the idea is make the dropping probability a function of a packet's *diffserv* markings.

3 Applications

We now briefly discuss how *diffserv* may be exploited in the context of applications, primarily Web browsing.

3.1 Adapting Dynamically To User Actions

Web browsing is fundamentally an interactive application. So ideally the process of downloading a Web page should adapt dynamically to user preferences as reflected by his/her actions. For instance, if the user scrolls down a Web page whose download is in progress, then the download of the images that fall within the newly visible portion of the page should be sped up while that of images not in view should be slowed down. Similarly, if the user clicks on a new hyperlink halfway through a download, the download of the new page should begin immediately.

In [Pad98], we presented the *TCP session* abstraction to address part of the problem, i.e., to alter scheduling on the sender (server) host itself when a message indicating a change in the user's preferences is received from the client. However, two other problems need to be addressed. First, the client's message may get delayed, for instance, due to queuing at the upstream link of an asymmetric-bandwidth network (as in Section 2.4). Second, even if the server were to start transmitting the newly-requested data immediately, there could be a long delay before the client receives the data because it may be queued behind previously transmitted data (for instance, a single 1 KB packet would cause a delay of 280 ms over a 28.8 Kbps link).

These problems may be alleviated by assigning a high scheduling (and drop) priority to the packets of interest.

3.2 Web Prefetching

Prefetching [Bes95, PM96] has been suggested as a technique to mask Web latency by anticipating user requests and downloading objects ahead of time. While being beneficial when the "guess" is correct, prefetching could hurt if the user never actually requests a prefetched object. The cost of an unnecessary prefetch is not only the CPU and storage resources expended, but also network bandwidth and buffer space consumed (which, in turn, impact the packet loss rate experienced by other traffic).

So, given the tentative nature of the prefetched data, it may be worthwhile to assign a low drop (and possibly scheduling) priority to the prefetch packets so that their impact on more "legitimate" and urgent traffic is minimal.

3.3 Other Applications

Priority dropping has long been proposed for video transmission as a means to shield critical data in the event of network congestion. With layered encoding, packets of the base layer are assigned a higher drop priority than those of the enhancement layers. Even with a non-layered encoding, such as MPEG, the more critical I-frames could be assigned a higher drop priority than the P and B frames, as proposed in PET [ABE+94]. We believe a similar technique could be used for audio. For instance, the low-bandwidth "redundant" audio data proposed in [HSH+95] could be carried in separate packets that are assigned a higher drop priority than the regular packets.

4 Discussion

The use of *diffserv* in the manner we advocate in this paper raises several issues.

The setting of the DS bits on a packet-by-packet basis adds complexity and increases the chances of bugs. To ensure that the added complexity is not in vain, only those instances where the use of *diffserv* would yield significant benefits should be implemented. Our experimental evaluation of TCP fast start and acks-first scheduling in asymmetric networks shows that *diffserv* is indeed very valuable in those instances. A similar thorough evaluation is needed for the other proposals made in this paper.

To minimize the chances of inadvertent bugs being introduced, the use of the *diffserv* mechanisms should be localized in a small, well-defined subset of the code. So, for instance, if changes were restricted only to the TCP code, then all applications that use TCP could derive benefit without the risk of introducing any (new) bugs.

Using *diffserv* in the manner we advocate could cause undesirable side-effects. For instance, acks-first scheduling could cause starvation of upstream data packets. To avoid this, we regulate the number of (high-priority) acks using ack congestion control [BPK97]. In general, the fraction of packets with priority other than normal should be kept small since the priority of a packet is only meaningful *relative* to other packets of different priorities.

Priority scheduling could be quite disruptive to TCP because it can cause packet reordering and significant variation in RTT from one packet to the next. So it may not be desirable to assign differential scheduling priorities to packets within a single connection while it may be okay to do so across connections. Note that priority dropping does not suffer from this problem because it retains FIFO ordering.

It unclear what the set of “ideal” *diffserv* mechanisms might be. Even priority dropping can have several variants, for instance, based on the queue length threshold at which dropping is triggered. But regardless of the specific mechanisms, we would like there to be at least three service levels – normal, better than normal, and worse than normal – to support the algorithms proposed in this paper.

A related issue is that *diffserv* may not be supported end-to-end, at least in today’s Internet. Even if it is, there may not be an agreement on the mechanisms across domains and/or packets may be re-marked at inter-domain boundaries. However, the absence of end-to-end support does not necessarily invalidate the proposals made in this paper. In the case of optimizations that involve assigning a subset of packets (e.g., TCP SYN packets) a better-than-normal service level, the absence of end-to-end *diffserv* support would render the situation no worse than if *diffserv* mechanisms were not used at all (i.e., no packets were marked). For certain other optimizations, such as that discussed in Section 2.4, it may be sufficient if *diffserv* were supported by the access network alone. In the case of TCP fast start, we need priority dropping support at the router leading to the bottleneck link, which is likely to be the location where packet losses occur. If the bottleneck tends to shift over time, then we would need *diffserv* support at all the potential bottlenecks.

Finally, there is the concern that *diffserv* mechanisms may be a disincentive for “good” behavior on the part of protocols and applications. In our view, the real disincentive is just the fact that the Internet is a *shared* network without hard boundaries separating users. Network users have an incentive to be greedy even in the absence of *diffserv*. (The analysis in [BBS98] appears to support this contention.) The use of *diffserv* as we advocate does not alter this situation in any fundamental way. However, it does make the task of ensuring compliance and identifying misbehavior more challenging. We believe that this latter task can be

made tractable by hiding the *diffserv* “knobs” from the end user, by ensuring application and protocol compliance through extensive testing, and by employing policing mechanisms as discussed in [FF97].

5 Conclusions

In this position paper, we have advocated the use of the Internet’s (proposed) differentiated services mechanisms to improve the performance of network protocols and applications. The key idea is to exploit *diffserv* to enable the sending host to control packet dropping and scheduling at a much finer timescale than traditional end-to-end mechanisms permit. We have made the case that the *diffserv* mechanisms can be very useful even in what are traditionally considered as “non-realtime” contexts, such the TCP protocol and the Web browsing application. We have designed, implemented, and evaluated two of our proposals – TCP fast start and acks-first scheduling – extensively, and have obtained promising results.

Acknowledgements

We thank the anonymous reviewers for their valuable comments.

References

- [ABE+94] A. Albanese et al., “PET – Priority Encoding Transmission”, Proc. ACM FOCS, November 1994.
- [AFP98] M. Allman, S. Floyd, C. Partridge, “Increasing TCP’s Initial Window”, RFC 2414, IETF, September 1998.
- [BBB+98] Y. Bernet et al., “A Framework for Differentiated Services”, Internet Draft, IETF, February 1999 (working draft).
- [BBC+98] S. Blake et al., “An Architecture for Differentiated Services”, RFC 2475, IETF, December 1998.
- [BBS98] S. Bajaj, L. Breslau, S. Shenker, “Uniform versus Priority Dropping for Layered Video”, Proc. ACM SIGCOMM, September 1998.
- [BCS94] R. Braden, D. Clark, S. Shenker, “Integrated Services in the Internet Architecture: An Overview”, RFC 1633, July 1994.
- [Bes95] A. Bestavros, “Using Speculation to Reduce Server Load and Service Time on the WWW”, Proc. ACM International Conference on Information and Knowledge Management, 1995.
- [BPK97] H. Balakrishnan, V. N. Padmanabhan, R. H. Katz, “The Effects of Asymmetry on TCP Performance”, Proc. IEEE/ACM Mobicom, September, 1997.
- [BZB+97] R. Braden et al., “Resource Reservation Protocol (RSVP) Version 1 Functional Specification, RFC 2205, IETF, September 1997.
- [CW97] D. Clark, J. Wroclawski, “An Approach to Service Allocation in the Internet”, Internet Draft, IETF, July 1997 (working draft).
- [FF97] S. Floyd, K. Fall, “Router Mechanisms to Support End-to-End Congestion Control”, Technical Report, Lawrence Berkeley National Laboratory, USA, February 1997.

- [FJ93] S. Floyd, V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, August 1993.
- [FJL+95] S. Floyd et al., "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing", Proc. ACM SIGCOMM, August 1995.
- [FKS+97] W. Feng, D. Kandlur, D. Saha, K. Shin, "Understanding TCP Dynamics in an Integrated Services Internet", Proc. NOSSDAV, May 1997.
- [Hei97] J. Heidemann, "Performance Interactions between P-HTTP and TCP Implementations", ACM SIGCOMM Computer Communication Review, April 1997.
- [HSH+95] V. Hardman, M. A. Sasse, M. Handley, A. Watson, "Reliable Audio for Use Over the Internet", Proc. INET, 1995.
- [Jac88] V. Jacobson, "Congestion Avoidance and Control", Proc. ACM SIGCOMM, August 1988.
- [Kim95] H. Kim, "A Non-Feedback Congestion Control Framework for High-Speed Networks", Ph.D. Thesis, University of Pennsylvania, USA, 1995.
- [Mah97] B. Mah, "An Empirical Model of HTTP Network Traffic", Proc. IEEE Infocom, April 1997.
- [PM94] V. N. Padmanabhan, J. C. Mogul, "Improving HTTP Latency", Proc. 2nd International WWW Conference, October 1994.
- [PM96] V. N. Padmanabhan, J. C. Mogul, "Using Predictive Prefetching to Improve WWW Latency", ACM SIGCOMM Computer Communication Review, July 1996.
- [Pad98] V. N. Padmanabhan, "Addressing the Challenges of Web Data Transport", Ph.D. Thesis, University of California at Berkeley, USA, September 1998. <http://www.cs.berkeley.edu/~padmanab/phd-thesis.html>.
- [PK98] V. N. Padmanabhan, R. H. Katz, "TCP Fast Start: A Technique for Speeding Up Web Transfers", Proc. IEEE Globecom Internet Mini-Conference, November 1998.
- [Ste94] W. R. Stevens, "TCP/IP Illustrated Volume 1", Addison-Wesley Publishers, 1994.
- [Tou97] J. Touch, "TCP Control Block Interdependence", RFC 2140, IETF, April 1997.
- [VH97] V. Visweswaraiyah, H. Heidemann, "Improving Restart of Idle TCP Connections", Technical Report 97-661, University of Southern California, USA, November 1997.