The Case for Informed Transport Protocols

Stefan Savage Neal Cardwell Tom Anderson
{savage,cardwell,tom}@cs.washington.edu
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195, U.S.A.

Abstract

Wide-area distributed applications are frequently limited by the performance of Internet data transfer. We argue that the principle cause of this effect is the poor interaction between host-centric congestion control algorithms and the realities of today's Internet traffic and infrastructure. In particular, when the duration of a network flow is short, then using end-to-end feedback to determine network conditions will be extremely inefficient. We propose an incremental approach to the problem, in which congestion information is shared among many co-located hosts and transport protocols make informed congestion control decisions. We argue that the resulting system can potentially improve the performance experienced by each network user as well as the overall efficiency of the network.

1 Introduction

While there has been considerable research invested in mitigating the effects of inadequate memory performance[22, 5], disk performance [23, 24], and LAN performance [34], for the growing number of Internet-based applications, these are secondary problems at best. Requesting a simple 10KB object from a Web server 35ms away is likely to proceed slower than 200kbps in today's Internet, and even under ideal conditions and with infinite bandwidth, it *cannot* transfer significantly faster than 300kbps. The principle problem is that the congestion control algorithms used by the popular Transmission Control Protocol (TCP) are poorly matched to the traffic and infrastructure characteristics of today's Internet. Unlike local resources, such as memory or disk blocks, network resources are shared outside the view of the host operating system. Consequently, the congestion control algorithms used by TCP are ignorant about the availability of network resources and must repeatedly probe the network to "discover" its characteristics. This design worked well when each individual data transfer was long, comprised a significant fraction of the traffic along any given path, and experienced few packet losses. Today, in contrast, the Internet is increasingly dominated by "short flows", "fat pipes", and "frequent losses". In this environment, TCP's lack of information not only limits the performance of each flow but also reduces the overall efficiency of the network.

We contend that the best way to address these problems is to relax the traditional view that the network is a "black box". Instead, we believe that end-to-end congestion control mechanisms can be significantly enhanced by sharing congestion information among many co-located hosts. Our observation is that while any individual host possesses only limited information about the network, the union of this information across hundreds or thousands of hosts can provide a far more complete picture. Other work has explored the advantages of sharing congestion state between flows *within* a single host [27, 21, 32] but to our knowledge the benefits of inter-host sharing are unexplored (although the possibility is mentioned in [21]).

In the rest of this paper we'll review TCP's default congestion control mechanisms, discuss how they interact with current traffic patterns, and argue that both the host and the network could benefit from sharing congestion information among many hosts. Finally, we'll present evidence that current Internet traffic patterns have enough locality to benefit from such an approach.

2 Background

TCP is currently the dominant transport protocol in the Internet and is the foundation of applications such as Web browsing, E-mail, file transfer and news distribution. It is a reliable, connection-oriented protocol that uses a sliding window to explicitly flow-control the use of receiver buffer space. Since it was first introduced [26], there have been several enhancements to TCP designed to improve its behavior during times of network congestion [20, 13]; the most significant of these was developed by Van Jacob-

son [12] in response to a "congestion collapse" event in the late 1980's. We provide a rough overview of the essential congestion control algorithms used in modern TCP implementations below. We have taken the liberty of describing the algorithms in terms of packets for simplicity, although they are usually implemented in terms of bytes (a more detailed description may be found in Steven's well known books on TCP [29, 30]).

- *Slow start.* When a host first starts a TCP flow it is completely ignorant about the state of the network and does not know the proper rate at which to send packets. To quickly "learn" this information, the slow start algorithm increases the amount of unacknowledged data that may be sent, called the "congestion window" or *cwnd*, until a loss is detected or until the "slow start threshold", or *ssthresh*, is exceeded. When a flow starts *cwnd* is initialized to a single packet and *ssthresh* is set to the largest possible value. *cwnd* is incremented by one packet for each in-sequence acknowledgment that is received, leading to an exponential growth rate. When a loss is detected, as described below, it is assumed that this is an indication that the host is sending too fast and must slow down.

- *Timeouts and fast retransmit.* There are two mechanisms for detecting a loss. The first is the expiration of the timeout timer TCP sets when a packet is sent. The timeout value is chosen somewhat conservatively to accommodate changes in measured round-trip time caused by increased load on the network. After a timeout, the sender must set its *cwnd* to one and re-initiate the slow start procedure.

  The second loss indication is the arrival of three "duplicate acknowledgments". When the receiver receives an out-of-sequence packet, it responds by sending an acknowledgment for the last *in-sequence* packet it received. Consequently, if a packet is lost then subsequent packets arriving at the receiver will result in a duplicate acknowledgment being returned. Because this effect can also be triggered by packet reordering in the network, the sender waits until it has received three consecutive duplicate acknowledgments before it interprets this as an indication of a loss. In this event, TCP retransmits the lost packet immediately, hence the name for the algorithm, *fast retransmit*. The principle advantage of this mechanism over timeouts is that it can detect a loss much more quickly.

  After any loss, *ssthresh* is divided by two. If *cwnd* ever exceeds *ssthresh* then TCP enters the congestion avoidance phase.

- *Congestion avoidance.* After finishing slow start, or recovering from a loss, TCP continues to probe the network to see if the path capacity has changed. The congestion avoidance algorithm increases the congestion window by 1/*cwnd* for each in-sequence acknowledgment that is received, resulting in an overall linear increase.

Since the introduction and wide-scale deployment of these algorithms, there has been considerable effort devoted to improving on their behavior [11, 3, 35, 36, 6, 17, 1, 15]. Most, if not all, of the problems addressed by these extensions can be attributed to TCP's ignorance of prevailing network conditions. In the following section, we describe how this lack of information interacts poorly with today's Internet.

3 Short Flows, Fat Pipes, Frequent Losses

Numerous measurements show that Internet traffic is now dominated by flows involving small Web objects 10-20KB in size [10, 31]. The average Web document is less than 40KB [16] so flow length is unlikely to be changed significantly by any simple modification to HTTP. At the same time, the capacity of network links is growing dramatically. Finally, while TCP was originally designed assuming that the average packet loss rate was less then 1% [4], current measured packet loss rates exceed 5% [1, 25]. This "short flows, fat pipes, frequent loss" trend undermines a number of the assumptions on which TCP's congestion control algorithms are based.

- *Connection establishment.* It is assumed the packet loss rate is low enough that connection establishment (SYN) packets are almost always delivered reliably. In the event that they are not, it is assumed that the resulting delay is well-amortized over the long lifetime of the connection. Unfortunately, with a uniform loss rate of 5%, the probability of a SYN packet being lost is quite high; $1-(1-.05)^2$, or 10% of the connection attempts made. If the host requests a web page with 5 or 6 inline images, there are even odds that at least one of their SYN packets will be lost. Further, when TCP starts a connection it has no information about the path round-trip time, and so it uses a conservative timeout (three seconds in many implementations) that is considerably longer than the lifetime of most flows.

- *Reaching equilibrium.* It is assumed that the duration of each network flow is long enough that it can reach equilibrium using end-to-end feedback. In this situation, TCP's expected bandwidth can be approximated as:

$$BW < \left(\frac{MSS}{RTT}\right)\frac{1}{\sqrt{p}}$$

where MSS is the maximum number of bytes in a packet, RTT is the round-trip time, and p is the probability that a packet is lost [18].

However, when flows are short, as they are with most Web requests, there are not enough packets sent and received to reach a correct estimate of the available bandwidth or the round-trip time. Consequently, TCP spends most or all of its time in slow start. In such circumstances the bandwidth experienced by a TCP flow is bounded by:

$$BW < \frac{B}{RTT \cdot \lceil \log_{1.5}\left(\frac{B}{2 \cdot MSS} + 1\right) \rceil}$$

where B is the length of the flow in bytes, RTT is the round-trip time, and MSS is the maximum number of bytes in a packet. The intuition for this bound is that the number of round-trip times necessary to send B bytes is related to the log of B because of slow start's exponential growth.

During the time it is in slow start, a TCP flow is usually sending slower than its fair share, and any excess link capacity that is available will be wasted. When it exits slow start, a TCP flow can send as much as twice the available bandwidth into the network, resulting in unnecessary packet losses. As network links increase in capacity, the time it takes to reach equilibrium increases, and the range of transfers which experience this problem increases as well.

- *Detecting congestion.* It is assumed that the average queue length at the bottleneck router does not change too quickly, so that a host can usefully correlate its own sending rate with the observation that a packet was lost. This assumption is important for congestion avoidance, so a flow can moderate its sending when load increases and send faster when load decreases. However, in a traffic mix containing mostly short flows, the load on the bottleneck queue is very bursty, leading to very short term congestion events, so a host's packets are increasingly likely to be lost even when the host is not sending too quickly.

- *Responding to congestion.* Finally, it is assumed that network traffic is *elastic* and when congestion causes the network to drop packets, that this will result in an exponential decrease in the offered load leading to overall stability [12]. Unfortunately, for the short flows common in today's networks, the quality of the overall congestion "signal" can dominated by sampling error; the average short TCP flow does not see any packet losses at all and behaves in a purely *inelastic* manner. Floyd et al. [9] propose that "TCP-friendly" congestion control algorithms should never send faster than:

$$BW < \frac{1.5\sqrt{2/3} \cdot MSS}{RTT \cdot \sqrt{p}}$$

We observe that by this definition, aggregates of small TCP flows can be TCP-unfriendly.

In the next section we describe how we can use expanded network information to address many of these problems.

4 Informed Congestion Control

Our design, similar to SPAND [28], uses a gateway that promiscuously monitors all traffic leaving an organization's network. For each destination network the gateway tracks a smoothed average of the bandwidth, round-trip time, and packet loss rate, as well as the variance for each metric, and metric to represent the currency of the data (e.g. the last timestamp). There are several implementation strategies for integrating this information into TCP's congestion control algorithms. First, the host can request performance estimates directly for a given destination. Alternatively, if the gateway is configured "in-line" between the outgoing WAN connection and the organization, it can piggy-back performance estimates on acknowledgments to the sender (a la Explicit Congestion Notification (ECN) [8]). Finally, the gateway can be configured as a transparent TCP proxy and implement the congestion control policy directly by delaying or accelerating the rate at which packets enter the network. In this paper we do not favor a particular implementation strategy, nor do we distinguish between server-side gateways (those aggregating flows from a popular cluster of servers) and client-side gateways (those aggregating the flows to a co-located set of clients).

In the remainder of this section we revisit the problems described previously in Section 3:

- *Connection establishment: Early retransmission.* Instead of using an overly conservative initial timeout, an informed TCP could leverage the aggregate measurements taken at the gateway, and set its initial timeout according to the round-trip time and variance measured by other hosts to the same destination. Consequently, when a SYN packet is lost, it can be retransmitted after a reasonable amount of time.

- *Reaching equilibrium: JumpStart.* Instead of probing the network to find its fair share, an informed host could start sending immediately at its fair share as determined by the total achieved rate seen to the same destination divided by the total number of simultaneous flows to that destination. This avoids the startup penalty imposed on hosts as well as the penalty imposed on the network from overshooting the correct

rate. However, as the time since the last network measurement decreases, so too we must our confidence in the measurement (although preliminary work in [2] provides some encouraging analysis of the temporal and spatial locality of network performance measurements). There are many ways to moderate the use of this bandwidth estimate, such as letting the rate decay exponentially after a timeout has passed, using rate based pacing [33] to prevent large numbers of back-to-back packets from being sent, or employing packet-pair measurements [14] to increase the confidence in the bandwidth estimate.

- *Managing congestion: Economies of scale.* As more flows are aggregated together, this should improve the correlation between the sending rate and observed packet losses. Moreover, as detailed in [21], it is possible to detect losses by observing duplicate acknowledgments *across* connections. At the same time, when congestion is detected the required reduction in rate can be spread among all the individual short flows, leading to a system that is significantly more fair and elastic. A key research problem is deciding the policy for how to relate a packet loss to the overall rate sent to a particular destination. A naive approach would be to simply treat the many flows as a single large flow. As the number of simultaneous flows becomes large this approach could become quite unfair to the individual flows relative to competing Internet traffic. At the same time, an approach that preserves the behavior of small flows could too too aggresive since flows using the gateway will reach their fair share more quickly than competing Internet flows.

5 Potential for sharing information

The potential benefit arising from sharing congestion information between hosts is directly related to the amount of temporal and spatial locality in the traffic stream. If most flows share the same bottleneck router at the same time, then can be an enormous benefit to using an integrated solution. If most flows use different bottlenecks, or only reuse bottlenecks after long periods, then there is unlikely to be much benefit. The truth is undoubtedly somewhere in-between. In this section, we provide some preliminary evidence that the amount of bottleneck locality at network borders is large.

5.1 Detecting sharing

The first problem we encounter is determining which networks flows share a bottleneck. This is difficult in the current Internet architecture because there is no easy way to determine where a packet was dropped. However, there are several approximations based on destination IP address that are likely to be conservative estimates. The first is "host locality", that is, flows that share a destination host IP address in common. Since flows leaving an organization take the same path through the network to a particular host, it follows that they must also share the same bottleneck link. Another approximation is "network locality", that is, flows that share the same destination network. This is somewhat more difficult to determine because the network part of an IP address is of variable length. However, few networks are allocated at a granularity smaller than 256 hosts, so a reasonable approximation is to assume that IP addresses sharing the most significant 24 bits belong to the same network. Determining a better estimate of which flows share a bottleneck is an open problem; several heuristics have been presented in the literature [2, 28].

5.2 Trace descriptions

In our analysis of locality we use traces of network traffic between the Internet and two large universities.

The first trace, was collected at roughly 3:00pm on April 16, 1998 from the network border of Harvard University. It comprises 30 minutes of data produced via the `tcpdump` program and constitutes all 83,500 TCP connections initiated by Harvard clients during this period; 92% of these flows and 76% of their bytes are for connections to port 80, the standard HTTP port. The anonymization scheme used in this trace is applied uniformly across the IP addresses so we will only describe its host locality.

The second trace was collected on November 23, 1998 between 2:30pm and 3:23pm at the network border of the University of Washington. It contains the 390,000 HTTP transactions originating from UW clients during this period and does not include other TCP flows such as FTP, SMTP, or telnet. This omission slightly biases the data to underrepresent the total amount of locality, but since HTTP traffic generally dominates the number of TCP flows [31] this is unlikely to have significant impact on the results. While the host addresses are anonymized in this trace, the anonymization function is applied independently to each octet of the address. Consequently, using the 24-bit heuristic described earlier, it is possible to extract measures of network locality as well as host locality.

5.3 Overall flow locality

Figure 1 shows the locality in flows from the two traces. For each new flow to a particular destination host or network, this represents the amount of time, T, since that destination was last accessed. If the new flow overlaps an existing flow to the same destination, then T is zero. There

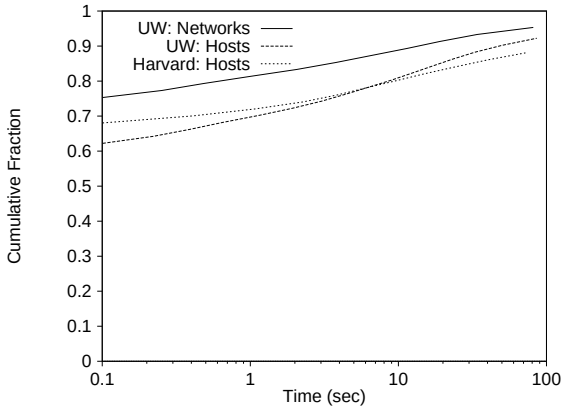

Figure 1. The cumulative fraction of T, the time since a destination host or network has last accessed.

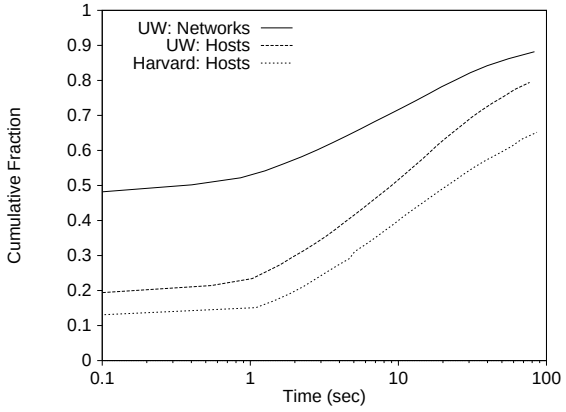

Figure 2. The cumulative fraction of T, the time since a destination host or network has last accessed. Multiple flows between the same source and destination are coalesced into a single flow.

is a striking amount of locality. For example, for 69% of flows to UW clients and 71% of flows to Harvard clients, the most recent flow from the same server ended less than 1 second ago. Furthermore, for 80% of flows to UW clients, the most recent flow from the same server *network* to a UW client ended less than 1 second ago. This suggests that many flows may be able to benefit from the congestion information accumulated by previous flows.

5.4 Inter-host flow locality

In Figure 1, intra-host and inter-host locality is intermixed. Inter-host locality is commonly caused by web browsers that launch multiple concurrent TCP sessions to the same site to request independent objects in a web page. While this kind of locality can be well exploited by our approach, we believe it is a legacy effect. New versions of HTTP [7, 19], new TCP implementations [32, 21] and new host-level congestion control interfaces [27] are likely to reduce this source of flow locality.

In Figure 2 we artificially extrapolate the impact of such changes by treating multiple flows between a source and a destination as a single flow, which we will refer to as a "persistent" flow. In particular, we coalesce all pairs of flows from a server to a client between which there was an idle period of one second or less. This corresponds to previous trace-based techniques for aggregating HTTP flows [16]. Examining Figure 2 we find while intra-host locality is a major factor, there is still significant inter-host locality. Considering only persistent flows, the fraction of flows which come from a server that has been seen in the last 1 second, is 15%-20%. For network locality the same value is just over 50% (although this might drop if we used a network-centric construction for persistent flows). Overall, this is an encouraging result; even assuming the complete deployment of host-based mechanisms to limited independent flows to the same destination, there is still significant potential for sharing congestion information *between* hosts.

6 Conclusion

Careful management of network resources is critical to acheiving better performance for today's network workloads using high-speed wide area networks. However, managing network resources is uniquely challenging because end-hosts cannot directly observe or control network dynamics. The high level of locality in network traffic suggests that by aggregating the observations of many different hosts within an organization, end hosts may be able to improve the performance and network friendliness of application data transport.

Acknowledgments

We would like to thank Robert Morris for providing the Harvard University packet traces on short notice, Alec Wolman and Geoff Voelker for collecting the University of Washington HTTP traces and providing code for manipulating them, and Brian Bershad for reading an earlier draft of this position paper.

References

[1] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, E. Amir, and R. H. Katz. TCP Behavior of a Busy Internet Server: Analysis and Improvements. In *Proceedings of the IEEE INFOCOMM '98*, San Francisco, California, Mar. 1998.

[2] H. Balakrishnan, S. Seshan, M. Stemm, and R. H. Katz. Analyzing Stability in Wide-Area Network Performance. In *Proceedings of the 1997 ACM SIGMETRICS Conference*, Seattle, WA, June 1997.

[3] L. S. Brakmo and L. L. Peterson. TCP Vegas: End-to-end Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, Oct. 1995.

[4] D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proceedings of the ACM SIGCOMM '88*, pages 106–114, Palo Alto, CA, Sept. 1988.

[5] S. Eggers, J. Emer, H. Levy, J. Lo, R. Stamm, and D. Tullsen. Simultaneous Multithreading: A Platform for Next-generation Processors. *IEEE Micro*, 17(5):12–18, Sept. 1997.

[6] K. Fall and S. Floyd. Simulated-based Comparisons of Tahoe, Reno and SACK TCP. *ACM Computer Communications Review*, 26(3):5–21, July 1996.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. RFC 2068, Jan. 1997.

[8] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communications Review*, 24(5):10–23, Oct. 1994.

[9] S. Floyd and K. Fall. Promoting the Use of End-to-end Congestion Control in the Internet. Unpublished Draft, Feb. 1998.

[10] S. D. Gribble and E. A. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, Dec. 1997.

[11] J. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *Proceedings of the ACM SIGCOMM '96*, pages 26–30, Palo Alto, California, Aug. 1996. Hoe95Thesis is thesis version.

[12] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of the ACM SIGCOMM '88*, pages 314–329, Palo Alto, CA, Aug. 1988.

[13] P. Karn and C. Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. In *Proceedings of the ACM SIGCOMM '94*, pages 2–7, Stowe, Vermont, Aug. 1987.

[14] S. Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of the ACM SIGCOMM '91*, pages 3–15, 1991.

[15] D. Lin and H. Kung. TCP Fast Recovery Strategies: Analysis and Improvements. In *Proceedings of the IEEE INFOCOMM '98*, San Francisco, California, Mar. 1998.

[16] B. A. Mah. An empirical model of HTTP network traffic. In *Proceedings of IEEE INFOCOM '97*, Apr. 1997.

[17] M. Mathis and J. Mahdavi. Forward Acknowledgment: Refining TCP Congestion Control. In *Proceedings of the ACM SIGCOMM '96*, pages 281–291, Palo Alto, California, Aug. 1996.

[18] M. Mathis, J. Semke, and J. Mahdavi. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM Computer Communications Review*, 27(3):67–82, July 1997.

[19] J. C. Mogul. The Case for Persistent-Connection HTTP. *ACM Computer Communications Review*, 25(4), Oct. 1995.

[20] J. Nagel. Congestion Control in IP/TCP Internetworks. *ACM Computer Communications Review*, 14(4), Oct. 1984.

[21] V. N. Padmanabhan. *Addressing the Challenges of Web Data Transport*. PhD thesis, University of California at Berkeley, Department of Electrical Engineering and Computer Science, Sept. 1998.

[22] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent RAM. *IEEE Micro*, 17(2), Apr. 1997.

[23] D. Patterson, G. Gibson, and R. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the International Conference on the Management of Data*, pages 109–116, June 1989.

[24] R. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed Prefetching and Caching. In *Proceedings of the 15th ACM Symposium. on Operating Systems Principles*, Dec. 1995.

[25] V. Paxson. End-to-end Internet Packet Dynamics. In *Proceedings of the ACM SIGCOMM '97*, pages 139–152, Cannes, France, Sept. 1997.

[26] J. Postel, Editor. Transmission Control Protocol — DARPA Internet Program Protocol Specification. RFC 793, Sept. 1981.

[27] H. Rahul, H. Balakrishnan, and S. Seshan. An End-System Architecture for Unified Congestion Management. In *Proceedings of 7th Workshop on Hot Topics in Operating Systems*, Mar. 1989.

[28] S. Seshan, M. Stemm, and R. H. Katz. SPAND: Shared Passive Network Performance Discovery. In *Proceedings of 1997 USENIX Symposium on Internet Technologies and Systems*, Monterey, Calinfornia, Dec. 1997.

[29] W. R. Stevens. *TCP/IP Illustrated*, volume 1. Addison Wesley, 1994.

[30] W. R. Stevens. *TCP/IP Illustrated*, volume 2. Addison Wesley, 1995.

[31] K. Thompson, G. J. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6):10–23, Nov. 1997.

[32] J. Touch. TCP control block interdependence. RFC 2140, Apr. 1997.

[33] V. Visweswaraiah and J. Heidemann. Rate Based Pacing for TCP. Unpublished Draft, 1998.

[34] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proceedings of the Nineteenth International Symposium on Computer Architecture*, pages 256–266, Gold Coast, Australia, May 1992.

[35] Z. Wang and J. Crowcroft. A New Congestion Control Scheme: Slow Start and Search (Tri-S). *ACM Computer Communications Review*, 21(1), Jan. 1991.

[36] Z. Wang and J. Crowcroft. Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm. *ACM Computer Communications Review*, 22(2), Apr. 1992.