Decongestion Control

Barath Raghavan and Alex C. Snoeren
University of California, San Diego
{barath,snoeren}@cs.ucsd.edu

ABSTRACT

Congestion control is fundamental to network design. Today's networks enjoy traffic stability, performance, and fairness in large part due to the use of TCP and TCP-like congestion control protocols. In such protocols, end hosts temper their transmission rates based upon packet losses, delay, and other observations to explicitly avoid persistent congestion.

We propose an alternative view on network congestion: it may not be necessary to keep the network uncongested to achieve good performance and fairness. We argue that a protocol that relies upon greedy, high-speed transmission has the potential to achieve better performance and fairness than TCP while simultaneously guaranteeing protection against misbehaving end hosts and obviating large router buffers.

1 INTRODUCTION

One of the key problems in network design is ensuring that available capacity is fairly and efficiently shared between competing end points. Traditionally, fairness has been achieved either in the network itself using fair queuing at routers [8], or through the cooperation of end hosts using a common congestion control protocol such as TCP. Unfortunately, both approaches have significant drawbacks: fair queuing is expensive to implement, while end-host congestion control is typically far from optimal and, critically, relies on the goodwill of end hosts for success [23, 25].

While the Internet has relied upon end-host cooperation for some time, ill-conceived or intentionally-aggressive end-point behavior can drive a network without fair queuing into congestion collapse. This scenario is a classic tragedy of the commons; individual selfish behavior can drive the system to a globally pessimal state, yet there is no incentive for any user to unilaterally back off. Thus, in game-theoretic terms, the Nash equilibrium of the network congestion-control game is sub-optimal [1, 13, 14, 24, 30]. Researchers have proposed a number of router-based enforcement mechanisms to avoid congestion collapse that vary in both complexity and effectiveness: some maintain per-flow state to provide near perfect fairness [3, 8, 9, 21, 26, 27], while others simply throttle the most aggressive senders [17, 20].

We observe that most of the complexity of both fair queuing and end-host-based congestion control is due to the perceived need to avoid dropping packets: in both models, well-behaved flows should experience little or no packet loss. With the advent of efficient, high-speed erasure coding [15, 18], we argue that packet loss no longer needs to be avoided. In fact, modern coding techniques can achieve high throughput even in the face of arbitrary packet loss. Hence, we propose a novel congestion control paradigm based on fair *dropping*—as opposed to queuing—and erasure-coded data streams that is both efficient and fair.

Rather than attempting to keep the network uncongested, the goal of our proposed approach, which we term *decongestion control*, is to ensure that all available capacity is used whenever it is needed by anyone. If packet drops are not of concern, then it is straightforward to align the interests of all parties: each sender simply sends as fast as possible. If congested routers drop packets in a fair manner [19], each flow will receive its max-min fair throughput. Better yet, if flows use efficient erasure coding, they will achieve goodput almost equal to their throughput, fully utilizing network bandwidth.

Of course, there is no (non-malicious) reason for a sender to transmit faster than a path's maximum unloaded capacity; even if no other flows were present, the sender's flow would be limited by this value. The main tasks of a decongestion control protocol, then, are to enable each sender to apportion its link capacity between destinations, and to determine at what *coding* rate to transmit. While packet drops are expected, the actual drop rate (and, thus, throughput) along any given path is unknown *a priori* and will vary from destination to destination. Depending on the coding method used, it may be advantageous for senders to adjust the coding rate in response to changes in path delivery rate.

While simple in spirit, there are more intricacies and ramifications of any congestion control approach than space allows. Hence, we do not attempt to detail the full design and implementation of a decongestion control protocol here. Instead, we present a case for decongestion by enumerating the key potential benefits, briefly sketching the basics of a possible design, and concluding with a partial list of challenges that must be addressed by a real implementation.

2 BENEFITS

Decongestion controlled networks have several attractive features over and above fairness and efficiency. Because packet drops are inconsequential, routers can be simple and provisioned with smaller queues. The resulting decreased fluctuation in traffic arrival rates and predictable traffic patterns similarly simplify traffic engineering. Finally, because goodput is only dependent on packet delivery rates, the most effective malicious behavior is flooding (as opposed to timing or protocol-based attacks), which is precisely the prescribed behavior when a sender has only one flow.

2.1 Fairness and efficiency

A key challenge facing traditional end-host congestion control algorithms is determining the appropriate fair share for each flow. TCP uses an additive increase/multiplicative decrease mechanism to converge to a flow-fair allocation. Unfortunately, this allocation can take a long time to converge on high capacity and/or long-delay paths and, even in the best case, oscillates around the optimal rate. This issue is particularly acute during slow start, when the sender needs to rapidly (re-)discover an appropriate rate. While numerous modifications to TCP have been proposed to improve slow-start, they still must rely upon complex mechanisms to help TCP rapidly discover additional available capacity should it become available during congestion avoidance.

With decongestion control, in contrast, senders always transmit at the maximum available rate; fairness is ensured by appropriate dropping policies at congested routers. Should available capacity increase at any router due to, for example, the completion of a flow, the remaining flows instantaneously take advantage of the freed link resources. The ability of a flow to translate increased throughput into increased goodput of course depends on the coding mechanism employed.

Tuning the coding rate between sender and receiver is not a new class of problem, however. For example, in TCP efficiency is managed by an end-to-end control loop (*i.e.*, receive window announcements) that ensures the sending rate does not exceed the receiver's ability to consume the data. We propose to use a similar mechanism described in Section 3.1 to dynamically adjust the coding rate based on recent throughput rates. A key distinction between adjusting the coding rate and changing the transmission rate, however, is that the coding rate has no impact on other flows. Hence, changes in available capacity (and, therefore, throughput) are likely to be less frequent since traffic rates fluctuate only on flow arrival and departure events, in contrast to TCP's sawtooth which probes for additional capacity and halves its flow transmission rate upon packet loss.

Our fundamental efficiency concern is that downstream packet drops will lead to wasted capacity at upstream links, thereby decreasing the overall throughput of the network. Kelly *et al.* use the term *dead packets* to refer to packets that will eventually be dropped before reaching their destination [10]. We conjecture, however, that the slow-access/fast-core structure of the Internet may alleviate the impact of dead packets in typical topologies. Conventional wisdom states that packet loss typically occurs at access links—not in the core of the network—so most flows will be thinned out before they reach the core. Further, dead packets in the core—those that will be dropped at receivers' access links—may be inconsequential in many cases. Previous studies have shown that existing research networks (in particular, Abilene) have over-subscription factors less than 2—that is, the access links can only deliver roughly twice as much traffic as can be serviced by the transit links at each access router [28]. We hope to empirically quantify the decrease in efficiency due to dead packets in real topologies.

2.2 Simplified core infrastructure

Much of the complexity in today's routers stems from the elaborate buffering schemes necessary to ensure loss-free forwarding at line rate. In addition, TCP's sensitivity to packet reordering complicates parallelizing router switch fabrics. Adding fair queuing or similar policing mechanisms to high-speed routers even further complicates matters. By decoupling loss rate and local packet order from the end-to-end congestion control protocol, decongestion control enables significantly simpler router designs. Idealized decongestion control only requires a fair dropping mechanism, which can be efficiently implemented with a single FIFO queue [19].

In addition to their inherent complexity, a significant portion of the heat, board space, and cost of high-end routers is due to the need for large, high-speed RAM for packet buffers. Previous work has shown that erasure coding can reduce the need for queuing in the network; in particular, for networks with large numbers of flows, coding schemes can provide similar goodput with coding buffer sizes on the same order as router buffers [4]. Hence, we suspect that such a minimalistic router design would require little buffering, which, in addition to reducing cost, also decreases the variance and maximum-possible end-to-end queuing delay. While recent work has shown that smaller router buffers may suffice for large TCP flow aggregates [2], smaller router buffers make TCP more vulnerable to bursty DoS attacks [11].

We suspect that decongestion control can also simplify traffic engineering. Decongestion control in no way affects the sources or sinks of data flows, and, therefore, does not impact traffic patterns. However, due to its

inherently greedy sender behavior, some links will be driven in excess of their capacity. In contrast to today's networks, where overloaded links cause TCP goodput to plummet (due to high delays, packet loss, and timeouts), overload does not require re-engineering paths; links are equally efficient at full capacity as they are when underutilized. The net result is that engineering for overprovisioned capacity would be largely unnecessary in such a network, though backup links are still needed to cope with maintenance and failure.

2.3 Incentive compatibility

Perhaps the most compelling benefit of decongestion control is its ability to sidestep many aspects of greed and malicious behavior.

It is nearly impossible for users to unilaterally increase their goodput by injecting more packets into a network dominated by decongestion control flows, since senders are transmitting at maximum rate anyway—the most effective way for a sender to increase its goodput is to adjust its coding rate, which, as previously mentioned, has no impact on other flows. This is in contrast to TCP, whose throughput can be gamed in a number of ways, perhaps most famously by the misbehaving-receiver attacks of Savage *et al.* [23].

Decongestion control is similarly more robust to malicious behavior due to its time independence. Senders adjust coding rates based upon reported throughputs—not individual packet events—so they are not as sensitive to short-term packet behaviors as TCP. In particular, there is little opportunity to launch well-timed bursty "shrew" attacks [11]. Our goal is to reduce all attacks to bandwidth attacks: ideally, there should be nothing more effective a malicious source can do than send traffic at a high rate. Unlike shrew attacks, flooding attacks are easy to detect and defend against.

3 DESIGN

Next we consider an initial approach to designing a decongestion control protocol, Achoo. At its most basic level, Achoo sends erasure-coded packets as fast as possible between a sender and a receiver. Packets are labeled with unique, monotonically-increasing sequence numbers, and the receiver periodically acknowledges packet reception with information about the rate of reception. Ideally, all routers implement a fair dropping policy to ensure that each flow receives its fair share of link bandwidth. We conjecture, however, that enforcing fairness only at access routers would provide an acceptable level of global fairness. (Recall that TCP itself is known to be unfair to flows with varying RTTs, loss rates, etc.) The design of such a dropping mechanism is beyond the scope of this paper, but a variant of AFD [19] or a similar mechanism suffices.

Fair dropping is most important when multiple bottlenecks are involved. In the absence of fair-dropping routers, Achoo flows traversing multiple congested routers will suffer: as a flow's packets traverse each link, they compete with other flows' packets, and as a result, lose some rate. Since short flows compete at fewer links, their packets will experience a lower loss rate, and thus, yield a higher steady-state goodput. However, a fair dropping scheme prevents this path-length induced unfairness: a flow's packets are only dropped if the flow is above its max-min fair share at each router, otherwise its packets are allowed through. Thus, once a flow has been throttled to its path fair share by an upstream router, downstream routers will ensure that the remaining packets reach their destination unhindered.

Achoo's transmission behavior is controlled by two components at the sender: the decongestion controller and the transmission controller. At a high level, all data to be sent is divided into *caravans*. Each caravan consists of n fixed-size (1Kb, say) data blocks; we pick n dynamically. The role of the decongestion controller is to select the appropriate rate of transmission, rate of coding, and caravan size. The transmission controller is responsible for ensuring the delivery of each caravan of data as instructed by the decongestion controller.

3.1 Decongestion controller

The decongestion controller has three fundamental tasks: selecting the caravan size, picking the appropriate level and type of coding, and balancing transmission rates across destinations. The space of options for each of these is large: caravans can be anywhere from 1 packet to thousands of packets, coding can vary from the simple (duplicate transmission or XORs) to the complex (LT coding), and available link capacities range from tens of kilobits to many gigabits.

To select the right caravan size, the controller starts with a fixed-size caravan and begins the transmission loop. When a caravan is successfully delivered, the controller doubles the size of the next caravan. If after some fixed timeout (likely a function of the RTT) there is insufficient data in the socket buffer to fill a caravan, the caravan size is halved. In this way, the controller quickly discovers the rate at which the source is generating data.

Once the caravan size has been identified, the decongestion controller must select the type and rate of coding to use for each caravan. Many strategies can be used, each with different guarantees and tradeoffs. For now we consider a simple approach in which small caravans consist of duplicate data (ordinary redundancy) and large caravans use rateless erasure codes. This effectively trades off both the cost and latency associated with erasure coding while harnessing its strengths for larger caravans of data. Because rateless codes can be expensive

to implement, we intend to experiment with variable-rate XOR coding for modest size caravans. Senders will adjust the rate of coding in response to the successful delivery rates reported by the receivers.

The final role of the decongestion controller is to apportion the available access link capacity across flows. Each physical interface has a maximum achievable rate (which is generally far in excess of the available wide-area capacity). The job of the controller is to determine which flows can put that capacity to most effective use. Initially, the nth flow on an interface is given $1/n$ of the link capacity and the transmission rates of the other flows are decreased proportionally.

The reception rates of all flows are constantly monitored; it is possible that the current transmission rates for some of these flows are insufficient to capture available capacity (we call them *unbottlenecked*). In this case, the controller considers conducting transmission rate experiments to determine which other flows are bottlenecked and, therefore, are not making effective use of their current transmission rate.

Experiments are conducted when a flow starts a new caravan. If the reception rate for the last caravan in the flow was less than the transmission rate, the controller attempts a rate decrease and monitors the resulting end-to-end delivery rate. If a transmission rate decrease results in no decrease in delivery rate, the decrease is kept, and the newly available capacity is distributed among all unbottlenecked flows. However, if the experiment results in a goodput decrease then the previous rate is retained.

Capacity is similarly reapportioned whenever a flow finishes. Note that transmission rates are only increased for unbottlenecked flows—bottlenecked flows are not deliberately increased, but are driven slightly over their bottleneck capacity in steady state, so any increase in path capacity will be immediately reflected in delivery rate (and the flow reclassified as unbottlenecked if necessary). Of course, if a flow is not able to make use of additional rate, its receive rate will drop below the transmission rate, subsequently subjecting the now bottlenecked flow to decrease experiments. In the case where no flow is able to make effective use of the additional capacity, it may be held in reserve.

In the normal case when link access bandwidth exceeds the bottleneck capacity for all of a sender's flows, this procedure keeps each flow overdriven but converges to (just above) the lowest rate at which the maximum end-to-end goodput is achievable for each flow; in this way, the sender wastes as few resources and still maximizes its welfare. Controlling transmission rate on the order of caravans allows for bulk flows to have more stable transmission rates, since they likely send large caravans, whereas short-lived or interactive flows may need rapid rate adjustment and will have small caravans.

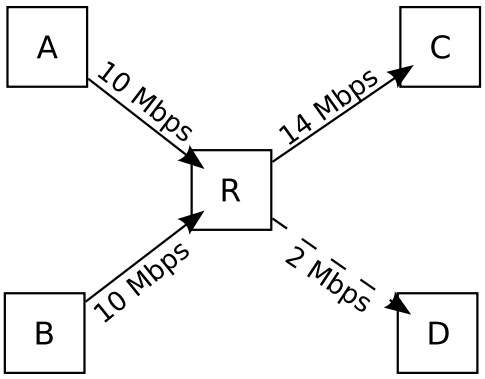

Figure 1: An example topology.

3.2 Transmission controller

The job of the transmission controller is simple: to ensure that each caravan is delivered successfully. Our approach is straightforward. Each caravan is streamed (using the rate and coding specified by the decongestion controller) until the sender receives an ACK indicating that the entire caravan has been successfully received and decoded. The transmission controller can then start sending the next caravan. More advanced designs might pipeline the transmission of caravans to eliminate the effect of network latency on inter-caravan spacing.

Interactive sessions or flows are distinguished by their sporadic data transmission. The decongestion controller uses the lack of enough data (a full caravan) as a signal to decrease caravan size: this ensures that interactive sessions transmit data in smaller units, and thus, with decreased latency. Also, since the type of coding used can depend upon the caravan size, interactive sessions can use simpler codes.

3.3 An example

To aid in an intuitive understanding of Achoo, consider the topology shown in Figure 1 depicting four end nodes, A, B, C, and D. A and B attempt to send data to C simultaneously using Achoo, which results in two flows of 10 Mbps each arriving at a router R. R implements fair dropping, so both A's flow and B's flow will achieve an end-to-end goodput of about 7 Mbps. After some time, A decides to start a flow to D, which forces it to divide its 10 Mbps between two flows. Achoo initially divides the available capacity evenly between the two flows, allowing the B-C flow to consume the remaining 9 Mbps on the R-C link. Since the link from R to D only has a capacity of 2 Mbps, if A sends at 5 Mbps to D, it will saturate the R-D link and declare the flow bottlenecked.

Conversely, the controller at A will notice that the A-C flow is unbottlenecked, and conduct bandwidth decrease experiments on the A-D flow until the A-C flow becomes bottlenecked again at 7 Mbps. In steady state, A will

overdrive its flows to both C and D in expectation of any additional capacity that may be freed up. Indeed, should the B-C flow cease, A would rapidly capture 8 Mbps for its A-C flow by conducting additional bandwidth decrease experiments on the A-D flow.

4 CHALLENGES

We consider a few of the many open questions and for each discuss the tradeoffs and issues faced by decongestion control.

4.1 What about coding overhead?

While coding provides essential functionality—resilience against loss—it also increases the end-to-end delay, packet overhead, and computational cost of decongestion control. Fortunately, different coding approaches can be used to suit the operating regime. For example, for short or interactive flows—ones with small caravans—packet duplication or simple XOR coding may yield low latency and low coding cost. As caravan size increases or there is more network contention, stronger coding schemes become necessary: flows with medium sized caravans (with sufficient computational resources) can use zero-overhead schemes such as Reed-Solomon codes, whereas large bulk flows may need to use rateless codes such as LT codes [15] or online codes [18].

4.2 What about the control channel?

Functionally, connection establishment and teardown for Achoo is the same as that for TCP. A server opens a listening socket on a particular port and establishes a new Achoo flow with each remote party sending a SYN. Similarly, once either party has decided to close the connection, they can send a close message to begin the connection teardown process. The challenge, then, is to ensure that control messages are not lost in the fray of competing data packets. For bi-directional flows, caravan ACKs can be piggybacked on coded data packets. For unidirectional flows and SYN/FINs, however, the receiver must independently determine the transmission and coding rate to use for the reverse channel.

Because the control channel contains small, independent messages, simple duplication will generally be an appropriate coding method. Determining the transmission rate, however, is more problematic. In TCP, the SYN and FIN handshakes are entirely sender-driven (a receiver only retransmits a SYN(FIN)/ACK upon receipt of a duplicate SYN/FIN). Unfortunately, with decongestion control, a single SYN/ACK packet is unlikely to be successfully delivered across a congested path. Hence, a receiver will likely need to dedicate some portion of its transmission rate for a control channel to each sender it is communicating with. A reasonable starting point is the rate currently being used by the sender to transmit its SYNs (which are piggybacked on the first data caravan for low-latency transmission of short flows). It is possible, however, that the return path is more severely congested, which would require the receiver to transmit the SYN/ACK at a faster rate.

Requiring the receiver to respond at a higher rate than the sender enables an obvious denial-of-service attack. Thus, we adopt the TCP method of requiring at least equal effort from the sender, so the receiver only increases its ACK rate in response to a commensurate increase in sender rate. Note that the transmission rate is likely far higher than the packet delivery rate at the receiver. The receiver determines the sending rate by observing the rate of change in sequence numbers of the packets it receives (each coded data packet has a unique, monotonically increasing sequence number). Implemented naively, however, the sender could lie, causing the receiver to expend undue effort.

4.3 What about unconventional routing?

In recent years researchers have proposed using alternative routing approaches such as overlay routing, intelligent multihoming, and source routing to improve performance and reliability. In such systems, flows can be redirected along different paths as traffic conditions change; additionally, in some designs, packets can be sent across multiple paths simultaneously. Achoo's relative insensitivity to instantaneous packet loss and reordering may allow for more aggressive route changes and/or multipath mechanisms than TCP would tolerate. However, decongestion, like congestion control, is path-specific, so appropriate coding and transmission rates need to be discovered after route changes.

5 RELATED WORK

The literature on congestion control and erasure coding is far too vast to adequately address here. Focusing specifically on the relationship between erasure coding and congestion control, researchers have compared ARQ schemes with FEC schemes [12] and integrated TCP and FEC [16, 22]. Naturally, FEC can be placed below, inside, or above TCP—thus, FEC can hide losses from TCP, be used to prevent retransmissions, or be applied to application-layer datagrams. Fountain codes [15, 18] famously highlighted the feasibility of non-ARQ based transport [5, 6] for broadcast and bulk data transmission. None of these schemes, however, have explored the practicality and ramifications of an entirely FEC-based congestion control on network design.

Several years ago, Davies proposed isarithmic networks in which the network is always fully utilized, just with *empties* when end hosts have no data to transmit [7]. Tracking empties proves problematic, however: just as

a token ring protocol requires a token-recovery mechanism, an isarithmic network needs some way to ensure that empties are not lost forever. Another proposal similar to ours, in that it deliberately overdrives network hosts (though not necessarily links), argues for clients of DDoS victims to increase their request rates to drown out the attackers [29], but transport flows remain congestion-controlled through TCP.

6 CONCLUSION

Given recent advances in coding techniques, we believe the time has come to consider networks that operate efficiently with high steady-state loss rates. Such a decongestion-controlled network could have simple routers, fair bandwidth allocation, low latency, stable traffic patterns, and incentive compatibility. We are currently designing and implementing Achoo and aim to quantify the potential benefits of decongestion control. Also, we note that many of the issues we are addressing have direct analogues in traditional congestion-controlled environments, so even if decongestion control is not adopted wholesale, the exercise may help us to re-evaluate tradeoffs made in the current Internet.

7 ACKNOWLEDGMENTS

We thank Rene Cruz, Bill Lin, and Scott Shenker for their thoughtful discussions. This work is funded in part through the UCSD Center for Networked Systems and a National Science Foundation Graduate Fellowship.

REFERENCES

[1] A. Akella, R. Karp, C. Papadimitrou, S. Seshan, and S. Shenker. Selfish behavior and stability of the internet: A game-theoretic analysis of TCP. In *Proceedings of ACM SIGCOMM*, 2002.

[2] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proceedings of ACM SIGCOMM*, 2004.

[3] J. C. R. Bennett and H. Zhang. WF2Q: Worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM*, 1996.

[4] S. Bhadra and S. Shakkottai. Looking at large networks: Coding vs. queueing. In *Proceedings of IEEE INFOCOM*, 2006.

[5] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. In *Proceedings of ACM SIGCOMM*, 2002.

[6] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A Digital Fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM*, 1998.

[7] D. W. Davies. The control of congestion in packet switching networks. In *Proceedings of ACM Problems in the optimizations of data communications systems*, 1971.

[8] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings of ACM SIGCOMM*, 1989.

[9] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, 1995.

[10] T. Kelly, S. Floyd, and S. Shenker. Patterns of conjestion collapse, 2003. unpublished.

[11] A. Kuzmanovic and E. W. Knightly. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of ACM SIGCOMM*, 2003.

[12] S. Lin, D. J. C. Jr., and M. J. Miller. Automatic-repeat-request error-control schemes. *IEEE Communications Magazine*, 22(12), 1984.

[13] L. López, G. del Rey Almansa, S. Paquelet, and A. Fernández. A mathematical model for the TCP tragedy of the commons. *Theor. Comput. Sci.*, 343(1-2):4–26, 2005.

[14] L. López and A. Fernández. A game theoretic analysis of protocols based on fountain codes. In *Proceedings of IEEE ISCC*, 2005.

[15] M. Luby. LT codes. In *Proceedings of IEEE FOCS*, 2002.

[16] H. Lundqvist and G. Karlsson. TCP with end-to-end forward error correction. In *Proceedings of International Zurich Seminar on Communications*, 2004.

[17] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM CCR*, 32(3):62–73, 2002.

[18] P. Maymounkov. Online codes. Technical Report TR2002-833, New York University, 2002.

[19] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *ACM SIGCOMM CCR*, 33(2):23–39, 2003.

[20] R. Pan, B. Prabhakar, and K. Psounis. CHOKe - A stateless queue management scheme for approximating fair bandwidth allocation. In *Proceedings of IEEE INFOCOM*, 2000.

[21] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

[22] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM SIGCOMM CCR*, 27(2):24–36, 1997.

[23] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. *ACM SIGCOMM CCR*, 29(5):71–78, 1999.

[24] S. Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. In *Proceedings of ACM SIGCOMM*, 1994.

[25] R. Sherwood, B. Bhattacharjee, and R. Braud. Misbehaving TCP receivers can cause Internet-wide congestion collapse. In *Proceedings of ACM CCS*, 2005.

[26] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *Proceedings of ACM SIGCOMM*, 1995.

[27] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high speed networks. In *Proceedings of ACM SIGCOMM*, 1998.

[28] R. Vasudevan, Z. M. Mao, O. Spatscheck, and J. van der Merwe. Reval: A tool for real-time evaluation of DDoS mitigation strategies. In *Proceedings of USENIX*, 2006.

[29] M. Walfish, H. Balakrishnan, D. Karger, and S. Shenker. DoS: Fighting fire with fire. In *Proceedings of ACM HotNets*, 2005.

[30] H. Zhang, D. Towsley, and W. Gong. TCP connection game: A study on the selfish behavior of TCP users. In *Proceedings of IEEE ICNP*, 2005.