Aggregate-Based Congestion Control

Ratul Mahajan, Steven M. Bellovin, Sally Floyd,
John Ioannidis, Vern Paxson, and Scott Shenker[*]

ICSI Center for Internet Research (ICIR) AT&T Labs – Research

Abstract

Recent events have illustrated the Internet's vulnerability to both denial of service (DoS) attacks and flash crowds in which links (or servers) in the network become severely congested. In both DoS attacks and flash crowds, the congestion is neither due to a single flow, nor due to a general increase in traffic, but to a well-defined subset of the traffic — an *aggregate*. This paper proposes aggregate-based congestion control (ACC) to protect the network from such aggregates. Our approach involves mechanisms for detecting and controlling high-bandwidth aggregates at the congested router, and a cooperative mechanism *pushback*, using which these aggregates can be controlled upstream. These mechanisms, while certainly not a panacea, should provide some relief from flash crowds and flooding-style DoS attacks.

1 Introduction

The current Internet infrastructure is highly vulnerable to both denial of service (DoS) attacks and flash crowds. During these, all traffic traversing the congested links experiences significantly degraded service over an extended period of time. In this paper, we view flooding-style DoS attacks and flash crowds as congestion events, and propose protection mechanisms to minimize their adverse impact.

The goal of a DoS attack is to impact a resource such as a link, a router, or a server in a way that service to legitimate users is denied or degraded. Flooding-style DoS attacks direct large amounts of traffic at the resource to overload it. The Internet is particularly vulnerable to distributed denial of service (DDoS) attacks, in which the attack traffic comes from a large number of sources. A series of DDoS attacks occurred in February 2000 to considerable media attention, resulting in a high packet loss rates for several hours [7, 15]. DDoS attacks have also been directed against network infrastructure rather than against individual servers [16].

Flash crowds occur when a large number of users try to access the same server simultaneously. Apart from overloading the server itself, the traffic due to flash crowds can also overload the network links and thereby interfere with other, unrelated traffic on the Internet. For example, degraded Internet performance was experienced during a Victoria's Secret webcast [2] and during the NASA Pathfinder mission. The "Slashdot effect" often leads to flash crowds.

While the intent and the triggering mechanisms for DDoS attacks and flash crowds are quite different, from the network's perspective these two events are quite similar. The persistent congestion is neither due to a single well-defined flow, nor due to an *undifferentiated* overall increase in traffic. Instead, there is a particular *aggregate* of packets causing the overload, and these offending packets may be spread across many flows.

Congestion caused by aggregates cannot be controlled by conventional flow-based protection mechanisms [3, 12, 14, 20] because the aggregate can be composed of numerous flows, each of which possibly being low-bandwidth. In this paper we propose aggregate-based congestion control (ACC) that operates at the granularity of aggregates, which falls between the traditional granularities of flow-based control (classifies packets into flows) and active queue management (does not differentiate between packets at all).

More specifically, an *aggregate* is a collection of packets from one or more flows that have some property in common. This property could be anything from the destination or source address prefix to a certain application type (*e.g.*, streaming video). Other examples of aggregates are TCP SYN packets and ICMP ECHO packets. An aggregate could be defined by a very broad property such as TCP traffic, or a very narrow one such as HTTP traffic to a specific host.

ACC mechanisms enable a congested router to identify the responsible aggregate(s) and control its throughput; this helps prevent service degradation experienced by other traffic. The congested routers can also use *pushback*, a cooperative ACC mechanism to control an aggregate upstream. Pushback prevents upstream bandwidth from being wasted on packets that are only going to be dropped downstream. In addition, for a DDoS attack, if most traffic is concentrated at a few upstream links, pushback can protect other traffic within the aggregate.

ACC mechanisms are intended to protect the network from persistent and severe congestion due to a rapid increase in traffic from one or more aggregates. We envision that these mechanisms would be invoked rarely, and emphasize that they are not substitutes for either adequate provisioning or flow-based congestion control. We believe that introducing control mechanisms at the granularity of aggregates would provide important protection against flash crowds, flooding-style DDoS attacks, and other forms of aggregate-based congestion.

The organization of this paper is as follows. In Section 2, we present an overview of the ACC mechanisms. We describe our current design in Section 3. Section 4 shows the utility and dynamics of the ACC mechanims using simulations. We discuss related work in Section 5, and conclude in Section 6.

[*]Ratul Mahajan is at University of Washington (work done while at ICIR); Steven M. Bellovin and John Ioannidis are at AT&T Labs – Research; Sally Floyd, Vern Paxson and Scott Shenker are at ICIR.

2 ACC Mechanisms

The ACC mechanisms can be thought of as taking the following sequence of decisions:
1. Am I seriously congested?
2. *If so*, can I identify the responsible aggregate(s)?
3. *If so*, to what degree do I limit the aggregate(s)?
4. Do I also use pushback?
5. When do I stop?

Each of these questions requires an algorithm for making the decision, mostly independent of each other. Each is also a natural point to inject policy considerations. The class of possible policies is very large; in this paper we assume simple policies in order to develop and understand the basic mechanisms. In this section we focus on the important considerations for answering each of these questions, and in the next we present a possible design.

2.1 Detecting Congestion

The ACC mechanisms should be triggered only when the output queue experiences sustained severe congestion. One can detect this by monitoring the loss rate at the queue, and looking for an extended high loss rate period. History of the loss rate pattern at the router can also be employed to distinguish between typical and unusual congestion levels, maybe even taking into account the time of day.

2.2 Identifying Responsible Aggregates

Since no preset definition of aggregates that may cause congestion exists, we need to identify them once serious congestion is detected. This is a tricky problem to solve in a general fashion for three reasons. First, there are many possible dimensions in which traffic may cluster to form aggregates: by source or destination address (a server experiencing a flash crowd), address prefix (a flooding attack targeting a site), or a specific application type (a virulent worm that propagates by email). Second, if the congestion is due to a DDoS attack, the attacker may vary her traffic to escape detection. Third, there may exist no offending aggregate at all, because the congestion is undifferentiated, such as that caused by an under-provisioned network or routing around a failure.

Analogous to the term *attack signature* for describing various forms of malicious activities, we use the term *congestion signature* to denote the aggregate(s) identified as causing congestion. If the congestion signature is too broad, such that it encompasses traffic beyond that in the true high-bandwidth aggregate, then we refer to the signature as incurring *collateral damage*. To minimize the collateral damage, identifying a narrow congestion signature is an important goal.

We propose that routers identify aggregates by applying clustering to a random sample of their traffic. Clustering should be based only on the fields in the packet header that can be trusted at the congested router. Moreover, the clustering algorithm should not identify any aggregate during undifferentiated congestion.

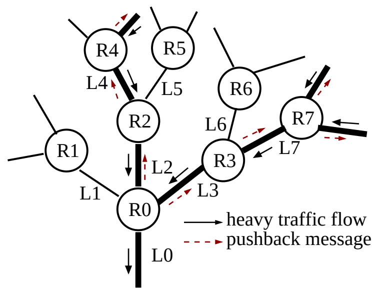

Figure 1: Pushback takes rate-limiting closer to the source(s).

Traffic history can also be used to verify that a particular aggregate is in fact responsible for the recent congestion. There are links in the network that are dominated by a particular aggregate in the normal case, and these links may remain dominated by that aggregate even during severe congestion caused by failures or other aggregates. The ISP can also use policy if it wants to protect such aggregates.

2.3 Determining the Rate Limit for Aggregates

We now turn to the question of to what degree the router should limit an aggregate. We argue that there is no useful, policy-free equivalent of max-min fairness when applied to aggregates; for example, we cannot give each destination prefix an equal share of the bandwidth. We also cannot completely shut off the identified aggregate unless we are sure it is a DDoS attack and that the congestion signature contains no legitimate traffic. We make protecting the other traffic on the link be the basis for deciding the rate-limit. The rate-limit for the identified aggregate(s) should be chosen such that a minimum level of service can be guaranteed for the remaining traffic, for example, by bounding the loss rate.

2.4 Pushback

Pushback is a cooperative mechanism that can be used to control an aggregate upstream. In pushback, the congested router asks its adjacent upstream routers to rate-limit the aggregate. Since the neighbors sending more traffic within the aggregate are more likely to be carrying attack traffic, this request is sent only to the *contributing* neighbors, *i.e.*, those that send a significant fraction of the aggregate traffic. The receiving routers can recursively propagate pushback further upstream.

Apart from saving upstream bandwidth through early dropping of packets that would have been dropped downstream at the congested router, pushback helps to focus rate-limiting on the attack traffic within the aggregate. Figure 1 illustrates this. Assume that *L0* is highly congested due to a high-bandwidth aggregate, and *R0* identifies the responsible aggregate. Local ACC can protect the traffic not belonging to the aggregate, but not the (likely) legitimate traffic within the aggregate coming from *L1*. In this case pushback will propagate from *R0* to *R2* and *R3*, and subsequently to *R4* and *R7*,[1] thus protecting traf-

[1]The path taken by pushback is the reverse of that taken by the aggregate, incidentally providing a form of traceback.

fic from *L1*, *L5* and *L6*.

Pushback is an optional mechanism, whose invocation is especially useful in two situations. First, when the sending rate of the aggregate remains much higher than the imposed limit. This implies that the router has not been able to control the aggregate locally by increasing its loss rate in an effort to encourage end-to-end congestion control. Second, when there is an indication that a DDoS attack is in progress. For instance, if most packets within the aggregate are destined for a notorious UDP port, the router can be fairly certain that it is not witnessing a flash crowd but a DDoS attack. Pushback can also be invoked by a router on the behest of a directly connected server, which can use application level information to distinguish between attacks and flash crowds [10].

Before invoking pushback, the ACC mechanism divides the rate-limit for the aggregate among the contributing neighbors. In the general case, all the contributing neighbors do not contribute the same amount; a link carrying more traffic belonging to the aggregate is more likely to be sending attack traffic; hence more traffic should be dropped from it. After determining the limit for each contributing neighbor, a pushback request message is sent to them. The recipients begin rate-limiting the aggregate with the specified limit. Pushback is propagated further upstream in a similar manner.

It is not necessary that all traffic in the congestion signature at an upstream router be traversing the congested router. For example, assume that the congestion signature determined by the congested router is traffic destined for 12.0.0.0/8; it is possible that an upstream router does not send all traffic in the /8 towards the congested router but only a subset of it. Hence, when propagating pushback, the congestion signature should be restricted (using the routing table) only to the traffic that traverses the congested router [13].

2.5 Reviewing Rate-limiting

Rate-limiting decisions are revisited periodically, to revise the limit on the rate-limited aggregates based on the current conditions, and to release some aggregates altogether if they starts to behave. These decisions are easy when rate-limiting is purely local (no pushback), as the router can continuously monitor its congestion as well the aggregates' arrival rate. However, we do need to worry about an attacker predicting this decision in order to evade ACC.

For pushback, however, the decision is more difficult; the router must distinguish between not seeing much traffic from the aggregate due to upstream rate-limiting, and the aggregate ceasing to be high-bandwidth. Disambiguating these two cases requires feedback from upstream, by the congested router estimating the real sending rate of the aggregate. Starting from the routers that did not propagate pushback, each rate-limiting router sends feedback to the downstream router reporting the arrival rate estimate for that aggregate. Routers that propagated pushback receive feedback from their upstream neighbors, which they consolidate to get an estimate of the arrival rate where they are located. This estimate is propagated downstream. Finally, the congested router takes rate-limiting decisions based on the feedback from its adjacent upstream routers.

To prevent attacks that send traffic intermittently to evade ACC mechanisms, it is imperative that the aggregate release time be much larger than the detection time.

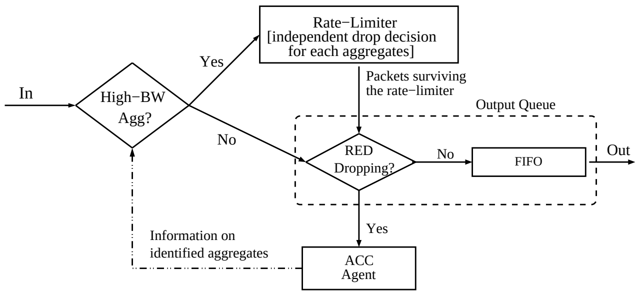
Rate−Limiter
[independent drop decision for each aggregates]
Packets surviving the rate−limiter
Output Queue
In
High−BW Agg?
Yes
No
RED Dropping?
No
FIFO
Out
Yes
Information on identified aggregates
ACC Agent

Figure 2: Architecture of an ACC-enabled router.

3 Design

In this section, we present our preliminary design of the ACC mechanisms. We present a brief description here; the details are documented in [13].

Figure 2 shows the architecture of an ACC-enabled router. Upon arrival, packets belonging to the identified aggregates go through the rate limiter, where some of them are dropped to enforce the rate limit. Other packets go directly to the output queue.

The ACC agent monitors congestion and identifies aggregates when it detects severe congestion. It also makes rate-limit refresh decisions. Since it does not sit in the fast forwarding path, it can be implemented in a box attached to the router rather than the router itself.

The rate limiter enforces the rate limit using a virtual queue [8] with a service rate equal to the specified limit. It also estimates the arrival rate of each rate-limited aggregate using exponential averaging [20], which is used in making refresh decisions. The rate limiter can also work more intelligently. Suppose it observes that the rate of a particular type of packets (ICMP ECHO, for example) within the congestion signature is significantly above normal. It is very likely that these packets are being used to launch the attack; the rate limiter can drop such packets more aggressively thus providing relief to other packets within the aggregate.

Our current design uses very simple algorithms for each of the problems specified in Section 2. Severe congestion is detected if the drop rate over the last $T_{monitor}$ seconds goes above a configured threshold of p_{high}.

Instead of installing a separate packet sampling mechanism, we use the RED [6] drops for aggregate identification because RED provides a reasonably fair distribution of drops [5]. Currently, we use only destination addresses for aggregate identification since most attacks and flash crowds have a common destination (prefix): that of the victim. Aggregates are identified by first clustering the commonly seen destination addresses (32-bit) in the drop history into 24-bit prefixes,[2] and then getting a longer prefix that contains most of the drops (to minimize collateral damage). This gives us a list of high-bandwidth aggregates, which we sort based on their sending rate (estimated using the number of drops seen for an aggregate) and the RED loss rate.

From the list of aggregates obtained above, we rate-limit one or more of them. The number of aggregates to control and their limits are computed such that enough traffic is shedded from the rate-limited aggregate(s) to bring down the RED loss rate (experienced by the other traffic) below a configured threshold of p_{target}. Moreover, this limit cannot be less than the throughput of any non-rate-limited aggregate. We currently do not have a technique to disambiguate between undifferentiated and aggregate-based congestion; instead, we place a bound on the number of aggregates that can be rate-limited simultaneously.

The ACC agent invokes pushback for an aggregate if the loss rate suffered by the aggregate in the rate limiter is very high. With the help of the rate limiter, the ACC agent estimates the arrival rate from each upstream neighbor and sends pushback requests to the contributing neighbors after dividing the rate limit among them. We divide the total rate limit, after adjusting for the non-contributing neighbors, among the contributing neighbors in a max-min fashion based on an estimate of their individual contribution. For example, assume that there are three contributing links with arrival rates of 2, 5, and 12 Mbps, the total incoming rate from all the non-contributing neighbors is 1 Mbps, and the rate-limit is 11 Mbps. Setting aside 1 Mbps for the non-contributing neighbors, we divide the remaining 10 Mbps among the three contributing links as 2, 4, and 4 Mbps. The upstream routers rate-limit the aggregate and decide whether to propagate pushback using similar algorithms, and periodically send feedback downstream.

The rate-limiting decisions are reviewed every $T_{refresh}$ seconds. Using the arrival rate from the rate limiter for local ACC or the feedback messages for pushback, along with the current RED loss rate level, the ACC agent decides on a new rate limit for each rate-limited aggregate. The new rate limit is used by the rate limiter and also pushed upstream after division if pushback is being used. If the arrival rate of an aggregate stays well below the limit for a small number of refresh intervals, the aggregate is released altogether. Note that continuing to rate-limit after the aggregate stops being high-bandwidth does not hurt, since no packets would be dropped by the rate limiter.

[2]This is based on the observation that most sites operate within a small range of IP addresses. Multiple aggregates may be identifi ed for sites with a spread-out IP address range.

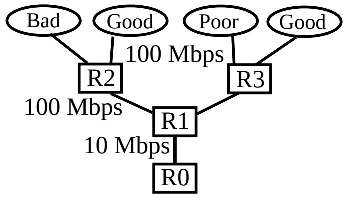

Figure 3: **Simple topology.** Link $R1$-$R0$ is congested.

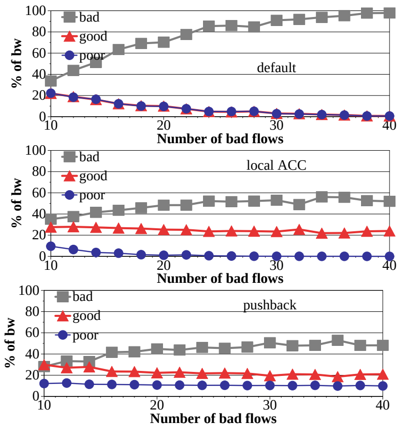

Figure 4: Throughput of different aggregates.

4 Evaluation

In this section, we illustrate the basic underlying functionality of ACC using simulations done with an *ns* [17] implementation of the design outlined in Section 3. We used the following values for configured thresholds: $T_{monitor}$=1 second, p_{high}=10%, p_{target}=5%, and $T_{refresh}$=5 seconds. These simulations do not pretend to use realistic topologies or traffic, but are intended as a first step toward a more rigorous design and evaluation of the mechanisms.

We use the following informal terminology to describe the simulation setup. The *bad* sources send attack traffic to their victim. The *poor* sources send legitimate traffic to the victim. Thus, the poor traffic incurs collateral damage when all traffic going to the victim is identified as the responsible aggregate. The *good* sources send traffic to destinations other the victim.

4.1 ACC Mechanisms

Figure 3 shows the topology for a simulation intended to show the dynamics of the ACC mechanisms. Each good and poor source generates traffic using seven infinite-demand TCP flows. The bad source uses a UDP flow with an on-off sending pattern with equal on and off times, chosen randomly between 0 and 40 seconds. Each bad flow sends at 1 Mbps during the on periods. A collection of these flows is a variable rate non-congestion-controlled traffic, harder to tackle because of its unpredictable sending rate. The number of bad flows is varied to model different levels of aggressiveness of the bad aggregate.

Figure 4 shows the results of the simulation without ACC (default), with only local ACC, and with pushback. In the default

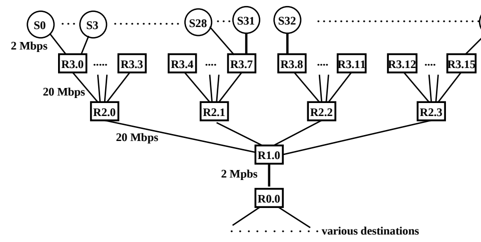

Figure 5: The topology used for DDoS attack and flash crowd simulations. $R1.0 - R0.0$ is congested.

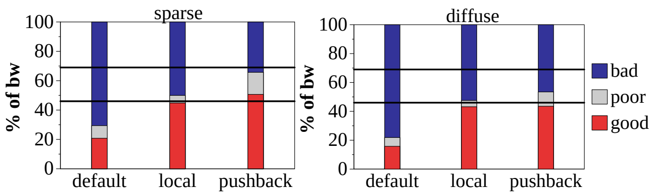

Figure 6: Bandwidth allocation at the congested link during DDoS attacks.

case, the bad aggregate consumes most of the bandwidth, and the good and the poor traffic suffer as a result. Local ACC controls the throughput of the bad aggregate to protect the good traffic, but fails to protect the poor traffic. Because local ACC cannot differentiate between the two, it penalizes the poor traffic along with the bad traffic. In contrast, pushback protects not only the good traffic, but also the poor traffic by pushing rate-limiting upstream where the bad and the poor sources can be differentiated.

4.2 DDoS Attacks

The simulations in this section illustrate the utility of ACC in the face of DDoS attacks, with both sparsely-spread and highly-diffuse attack sources. We use the topology shown in Figure 5. The link capacities are such that, apart from the congested link itself, congestion is limited to the access links.

Ten good sources and four poor sources are picked at random in the topology, each of which spawn Web-like traffic (using the Web-traffic generator in *ns*). The number of bad sources depends on the simulation scenario. The sparse-attack scenario contains four randomly chosen bad sources, each sending on-off UDP traffic (as above) but with an on-period sending rate of 2 Mbps. The diffuse attack scenario contains 32 UDP sources with an on-period sending rate of 0.25 Mbps.

Figure 6 shows the results for both the simulation scenarios. The horizontal lines represent the throughput of the good and the poor traffic in the absence of any bad traffic. Without ACC, the bad aggregate gets most of the bandwidth in both scenarios. Local ACC protects the good traffic but not the poor traffic. Pushback protects the poor traffic also, but that ability is reduced in the face of diffuse attacks; the poor traffic manages about 50% less throughput in the diffuse-attack scenario than in the sparse one. This is mainly because when the attacks are diffuse, even pushback cannot differentiate between the poor and the bad sources. In fact, it is possible to launch a highly diffuse attack in which each bad source generates less traffic than an average poor source, making it hard to distinguish between the two.

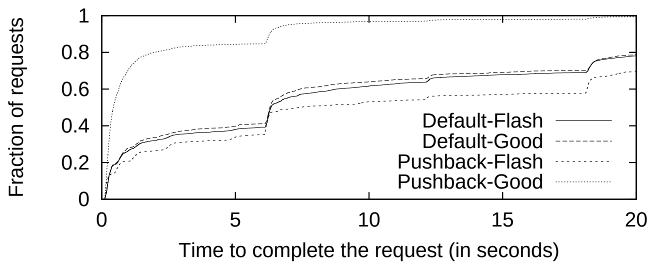

Figure 7: **Time to complete a request during a flash crowd.**

4.3 Flash Crowds

This section shows a simulation with a flash crowd, using again the topology shown in Figure 5. The "flash" traffic comes from 32 randomly picked sources, each sending Web-like traffic to the same destination. The good traffic comes from ten sources, each sending Web-like traffic to different destinations.

Figure 7 shows the cumulative distribution function of the transfer completion time for the good and the flash traffic in the default case without ACC, and in the pushback case with pushback and local ACC. The hump around the 6-second mark is due to timeout for transfers whose SYN or SYN/ACK packet was lost. With pushback, 80% of the good transfers complete within a few seconds, compared to less than 40% in six seconds in the default case. The use of ACC and pushback significantly benefits the good traffic, while resulting in only a moderate degradation for the flash traffic. With pushback, the drop rate for the good traffic is reduced from 30% to just 6% (p_{target}=5%), while the drop rate for the flash traffic is increased only by 3%, to 33%. Because this simulation has much more flash traffic than good traffic, even a small increase in the drop rate for the flash traffic frees up a significant amount of link capacity.

5 Related Work

Two common mechanisms to counter DDoS attacks are ingress filtering [4] and traceback [1, 18, 19]. ACC is orthogonal to both of them; the focus of ACC is neither to stop the attacks as in ingress filtering nor to find the sources of these attacks as in traceback, but to control the damage while the attack is in progress, whether or not the attack relies on spoofed source addresses.

Web-caching infrastructures, content distribution networks (CDNs), and multicast are powerful mechanisms for preventing flash crowds from congesting the network. However, even a combination of these techniques may not be sufficient to prevent all occurrences of network congestion due to flash crowds. The ACC mechanisms can provide additional protection whenever congestion is caused by flash crowds.

The goals of flow-based congestion control (FCC) [3, 20, 12, 14] are similar to those of ACC. In fact, we have borrowed several ideas from various FCC mechanisms. Pushback, in particular, is similar to credit-based flow control [11] in that both mechanisms send messages specifying how much traffic within a certain category the upstream should send. However, FCC operates at a different granularity, and cannot control aggregate-based congestion because an aggregate could be composed of many low-bandwidth flows.

6 Conclusions and Future Work

Congestion caused by aggregates differs in some fundamental aspects from that caused by individual flows, and hence requires different control mechanisms in the network. We have proposed both local and cooperative mechanisms for aggregate-based congestion control. Initial simulations have shown that these mechanisms are promising directions to control both DDoS attacks and flash crowds.

Much needs to be investigated about the ACC mechanisms. Apart from evaluating the trade-offs involved in various design choices to implement them, we need to understand the pitfalls and limitations of ACC itself. For example, pushback can potentially hurt innocent sources close to an attack source if it is not propagated upstream enough to differentiate between the two.

Other open issues include implementation complexity and deployability of ACC. A complex mechanism with high resource requirements can become a DoS mechanism itself. A technique to incrementally deploy pushback is presented in [13]; a prototype implementation can be found in [9].

Empirical answers to various issues are required to guide the design and evaluation the ACC mechanisms. These issues concern the ability to accurately identify aggregates in the network, distinguish between flash crowds and DoS attacks, and distinguish between undifferentiated and aggregate-based congestion. Pushback is most effective when the attack tree (the union of links used by the attack traffic) has a low branching factor, as this enables better localization of malicious sources. A study of DDoS attack trees observed in practice would be very useful in this context.

Finally, we expect the ACC mechanisms to be heavily influenced by policy. We plan to investigate the kinds of policies that these mechanisms need to support. Examples include protecting some aggregate even if it is high bandwidth, punishing some aggregate as soon as congestion sets in, providing relative fairness among aggregates, and restricting maximum throughput of an aggregate.

Acknowledgments

The original idea for pushback came from an informal DDoS research group consisting of Steven M. Bellovin, Matt Blaze, Bill Cheswick, Cory Cohen, Jon David, Jim Duncan, Jim Ellis, Paul Ferguson, John Ioannidis, Marcus Leech, Perry Metzger, Vern Paxson, Robert Stone, Ed Vielmetti, and Wietse Venema. We also thank Randy Bush, Eddie Kohler, and Ed Vielmetti for useful feedback.

References

[1] S. M. Bellovin, M. Leech, and T. Taylor. ICMP Traceback Messages. Internet-draft: draft-ietf-itrace-01.txt, Oct. 2001.

[2] J. Borland. Net Video Not Yet Ready for Prime Time. CNET news, Feb. 1999. http://news.cnet.com/news/0-1004-200-338361.html.

[3] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *SIGCOMM*, 1989.

[4] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofi ng. RFC 2827, May 2000.

[5] S. Floyd, K. Fall, and K. Tieu. Estimating Arrival Rates from the RED Packet Drop History, Apr. 1998. http://www.icir.org/fbyd/end2end-paper.html.

[6] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *Transactions on Networking*, Vol. 1(4):pp. 397–413, Aug. 1993.

[7] L. Garber. Denial-of-Service Attacks Rip the Internet. *IEEE Computer*, vol. 33(4):pp. 12–17, Apr. 2000.

[8] R. J. Gibbens and F. P. Kelly. Resource Pricing and the Evolution of Congestion Control. *Automatica, invited paper for special issue on control in communication networks*, 1999.

[9] J. Ioannidis and S. Bellovin. Implementing Pushback: Router-Based Defense Against DDoS Attacks. In *Proceedings of NDSS '02*, Feb. 2002.

[10] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *WWW*, May 2002.

[11] H. T. Kung, T. Blackwell, and A. Chapman. Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation and Statistical Multiplexing. In *SIGCOMM*, Aug. 1994.

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

[13] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling High-Bandwidth Aggregates in the Network (Extended Version). http://www.icir.org/pushback/.

[14] R. Mahajan, S. Floyd, and D. Wetherall. Controlling High-Bandwidth Flows at the Congested Router. In *ICNP*, Nov. 2001.

[15] Denial of Service Attacks Disrupt Internet. Matrix.Net, Feb. 2000. http://www.matrix.net/company/-news/20000209_dos.html.

[16] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. *USENIX Security Symposium*, Aug. 2001.

[17] NS Web Page: http://www.isi.edu/nsnam.

[18] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *SIGCOMM*, Aug. 2000.

[19] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-Based IP Traceback. In *SIGCOMM*, Aug. 2001.

[20] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *SIGCOMM*, 1998.