Controlling High Bandwidth Aggregates in the Network

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

The current Internet infrastructure has very few built-in protection mechanisms, and is therefore vulnerable to attacks and failures. In particular, recent events have illustrated the Internet's vulnerability to both denial of service (DoS) attacks and flash crowds in which one or more links in the network (or servers at the edge of the network) become severely congested. In both DoS attacks and flash crowds the congestion is due neither to a single flow, nor to a general increase in traffic, but to a well-defined subset of the traffic – an *aggregate*. This paper proposes mechanisms for detecting and controlling such high bandwidth aggregates. Our design involves both a local mechanism for detecting and controlling an aggregate at a single router, and a cooperative *pushback* mechanism in which a router can ask upstream routers to control an aggregate. The presentation in this paper is a first step towards a more rigorous evaluation of these mechanisms. While certainly not a panacea, these mechanisms could provide some needed relief from flash crowds and flooding-style DoS attacks.

1 Introduction

In the current Internet, when a link is persistently overloaded all flows traversing that link experience significantly degraded service over an extended period of time. Persistent overloads can arise for several reasons. First, persistent overloads can result from a single flow not using end-to-end congestion control and continuing to transmit despite encountering a high packet drop rate. There is a substantial literature [6, 18, 27, 20] on mechanisms to cope with such *ill-behaved* flows (where, by *flow*, we mean a stream of packets sharing IP source and destination addresses, protocol field, and source and destination port numbers). Second, as was seen on the transatlantic links a few years ago, persistent overloads can also be due to a general excess of traffic [14].

However, even when all links are adequately provisioned, and all flows are using conformant end-to-end congestion control, persistent congestion can still occur. Two examples of this are *denial of service* attacks (DoS) and *flash crowds*.

DoS attacks occur when a large amount of traffic from one or more hosts is directed at some resource of the network such as a link or a web server. This artificially high load denies or severely degrades service to legitimate users of that resource. The current Internet infrastructure has few protection mechanisms to deal with such DoS attacks, and is particularly vulnerable to distributed denial of service attacks (DDoS), in which the attacking traffic comes from a large number of disparate sites. A series of DDoS attacks occurred in February 2000 to considerable media attention, resulting in higher packet loss rates in the Internet for several hours [12]. DDoS attacks have also been directed against network infrastructure rather than against individual web servers [21].

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 offend-

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

ing packets may be spread across many flows.

Congestion caused by aggregates cannot be controlled by conventional flow-based protection mechanisms [6, 18, 20, 27] because the aggregate can be composed of numerous flows, each of which might be low-bandwidth. In this paper we propose aggregate-based congestion control (ACC)[1] that operates at the granularity of aggregates. ACC mechanisms fall between the traditional granularities of per-flow control (which looks at individual flows) and active queue management (which does not differentiate between incoming packets).

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 destination or source address prefixes to a certain application type (streaming video, for instance). Other examples of aggregates are TCP SYN packets and ICMP ECHO packets. An aggregate could be defined by a property which is very broad, such as TCP traffic, or very narrow, such as HTTP traffic to a specific host.

To reduce the impact of congestion caused by such aggregates, we propose two related ACC mechanisms. The first, *local* aggregate-based congestion control (Local ACC), consists of an *identification* algorithm used to identify the aggregate (or aggregates) causing the congestion, and a *control* algorithm that reduces the throughput of this aggregate to a reasonable level. There are many situations in which local aggregate-based congestion control would, by itself, be quite effective in preventing aggregates from significantly degrading the service delivered to other traffic.

The second ACC mechanism, *pushback*, allows a router to request adjacent upstream routers to rate-limit traffic corresponding to the specified aggregates. Pushback can prevent upstream bandwidth from being wasted on packets that are only going to be dropped downstream. In addition, for a DDoS attack, if the attack traffic is concentrated at a few upstream links, pushback can protect other traffic within the aggregate from the attack traffic.

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

The organization of this paper is as follows. Section 2 gives an overview of ACC. Section 3 describes the local component of ACC in more detail. We discuss the pushback mechanisms in detail in Section 4, and evaluate ACC using simulation in Section 5. Section 6 discusses the advantages and disadvantages of pushback, and several issues related to ACC. In Section 7 we describe other works that tackle the problems of DDoS attacks and flash crowds.

2 Overview of ACC

We can think about the ACC mechanisms as consisting of the following sequence of decisions:

1. Am I seriously congested?
2. *If so,* can I identify an aggregate responsible for an appreciable portion of the congestion?
3. *If so,* to what degree do I limit the aggregate?
4. Do I also use pushback?
5. When do I stop? When do I ask upstream routers to stop?

Each of these questions requires an algorithm for making the decision. Each is also a natural point to inject *policy* considerations. The space of possible policies is very large. 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. We do not attempt to explore such policies in this paper but assume simple policies in order to focus on developing and understanding the mechanisms.

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. The history of the loss rate pattern at the router can

[1] We note that the term "ACC" has also been used in different contexts to denote "Active Congestion Control" and "ACK Congestion Control".

be used to distinguish between typical and unusual congestion levels, maybe even taking into account the time of day.

2.2 Identifying Responsible Aggregates

When serious congestion is detected, the router attempts to identify the aggregate(s) responsible for the congestion. Identifying the offending aggregate(s) is a tricky problem to solve in a general fashion for three reasons. First, the overload may be chronic due to an under-engineered network, or unavoidable, *e.g.* as a shift in load caused by routing around a fiber cut. These lead to *undifferentiated congestion* not dominated by any particular aggregate. Second, there are many possible dimensions in which traffic might cluster to form aggregates: by source or destination address (e.g., a flash crowd of requests to access a particular server, or the replies), address prefix (a flooding attack targeting a site or a particular network link), or a specific application type (a virulent worm that propagates by email, inadvertently overwhelming other traffic). Third, if the congestion is due to a DDoS attack, the attacker may vary her traffic to escape detection.

We note that there are links in the network that are dominated by a particular aggregate(s) even in the normal case, and that might remain dominated by that aggregate even in the presence of diffuse congestion. The ISP can use policy if it wants to protect such aggregates, with the ACC mechanisms looking for other aggregates or rate-limiting these high-bandwidth aggregates only when they exceed their policy-defined limits.

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. When constructing congestion signatures, the router does not need to make any assumptions about the malicious or benign nature of the underlying aggregate (and this may not be possible in the face of a determined attacker). If the congestion signature is too broad, such that it encompasses additional traffic beyond that in the true high-bandwidth aggregate, then we refer to the signature as incurring *collateral damage*. Narrowing the congestion signature helps to reduce collateral damage.

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; no one would recommend for best-effort traffic that we give each destination prefix or application type an equal share of the bandwidth in a time of high congestion. Instead, the goal is to rate-limit the identified aggregate sufficiently to protect the other traffic on the link.

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) is chosen so that a minimum level of service can be guaranteed for the remaining traffic, for example, by bounding the loss rate.

A more Draconian measure, like completely shutting off or imposing a very low bandwidth limit for identified aggregates, is not taken because of two reasons. First, the aggregate can be a flash crowd. Second, even if the aggregate is from a DDoS attack, the congestion signature of the attack traffic will usually contain some innocent traffic too.

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

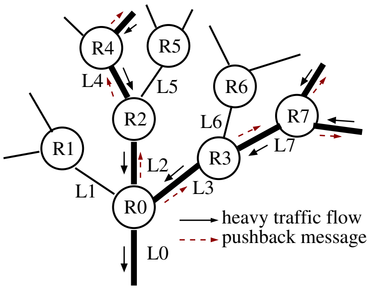

Figure 1: Pushback takes rate-limiting closer to the source(s).

R0 to *R2* and *R3*, and subsequently to *R4* and *R7*,[2] thus protecting traffic from *L1*, *L5* and *L6*.

Pushback is an optional mechanism, whose invocation is especially useful 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.

Pushback is also useful 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 [16]. The decision as to when to use pushback is likely to have a large policy component, which we do not address in this work.

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.

An unmodified congestion signature sent to an upstream router could cover some traffic that doesn't traverse the downstream 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 [19].

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 have started 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 routers, so that the congested router can estimate the real sending rate of the aggregate. To prevent attacks that send traffic intermittently to evade ACC mechanisms, the aggregate release time should be much larger than the detection time.

3 Local ACC

In this section, we present the ACC mechanisms in more detail. This section focuses on Local ACC, and the next on pushback.

Figure 2 shows the architecture of an ACC-enabled router. A filter at the entry to the regular FIFO output queue classifies a packet, and sends it to the *rate-limiter* if it belongs to an identified aggregate. Other packets proceed directly to the output queue. Relevant information (headers) about packets dropped[3] at the output queue is fed into the ACC Agent which uses it to identify high-bandwidth aggregates. The ACC Agent is not

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

[3]With an active queue management scheme (like RED) that distributes drops fairly, drops can be considered random sample of incoming traffic [10]. Alternatively, random samples can also be used.

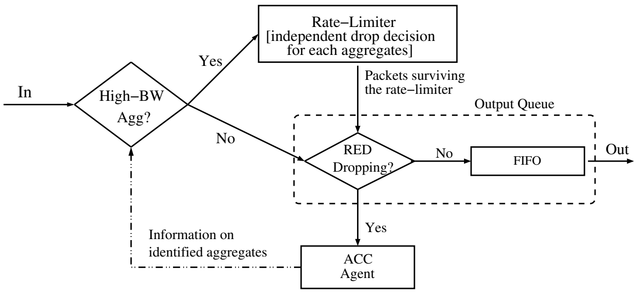

Figure 2: **Architecture of an ACC-enabled router.** Packets of high-bandwidth aggregates pass through the rate-limiter. All packets dropped by RED are passed to the ACC Agent for identifying aggregates.

in the fast path used for packet forwarding, and might not even be on the same machine.

The identification process in the ACC Agent is triggered when the output queue experiences sustained high congestion. We define sustained congestion as a drop rate of more than p_{high} over a period of K seconds. To get a rolling window effect, the drop history of the last K seconds can be kept as multiple lists of smaller periods, with the new list overwriting the oldest one [20]. Using the packet drop history of the last K seconds, the ACC Agent tries to identify a small number of aggregates responsible for the high congestion. If some aggregates are found, the ACC Agent computes the limit to which these aggregates should be restricted. The limit is computed such that the *ambient* drop rate, that is the drop rate at the output queue (not taking into account the drops in the Rate-Limiter), is brought down to below p_{target}. At the same time the limit on the rate-limited aggregate is not allowed to be less than the highest arrival rate among aggregates which are not being rate-limited. The ACC Agent is also responsible for modifying the limit imposed on various rate-limited aggregates based on changes in demand from background traffic.

The following subsections describe identification and rate-limiting in detail.

3.1 Identification of High Bandwidth Aggregates

In principle, an aggregate could be defined only in terms of the protocol field or port number; all DNS packets, for instance. However, almost all DDoS attacks and flash crowds have either a common source or a common destination prefix. As a result, we expect that most aggregate definitions will be based on either a source or destination address prefix. As is discussed later in the paper, pushback is invoked only for aggregates whose definition includes a destination address prefix.

We present a technique to identify high-bandwidth aggregates based on the destination address. The same technique could be used to identify aggregates based on the source address (though we acknowledge that source addresses cannot necessarily be trusted). This is only one of many possible algorithms for identifying high-bandwidth aggregates; more accurate and flexible algorithms are a subject of further research. More complex definitions of aggregates would require an appropriate language for expressing the aggregate definition and for passing the aggregate definition to upstream routers during pushback.

The identification technique presented below was designed with the observation that most Web sites operate in a small range of IP addresses[4]. Multiple aggregates may be identified for sites with a spread-out IP address range.

From the drop history, extract a list of high-bandwidth addresses (32-bit); for example, addresses with more than twice the mean number of drops. Now cluster these addresses into 24-bit prefixes. To minimize collateral damage, for each of these clusters try obtaining a longer prefix that still contains most of the drops. This can be easily done by walking down the prefix subtree with this 24-bit prefix at the root. At each step a heavily biased branch leads to a longer prefix with most of the weight. Next, try to merge prefixes that are closely related to each other. For example, two adjacent 24-bit prefixes can be described by a single 23-bit prefix. We now have a list of prefixes of different lengths, each describing a high-bandwidth aggregate. The next section addresses the question of which aggregates should be rate-limited.

Since access links have much less capacity than backbone links, they are more likely to be congested during DDoS attacks and flash crowds. The identification of high-bandwidth aggregates is easier in such cases. For instance, the aggregates used by the congested router could correspond to prefixes present in its routing table.

[4]The use of CDNs can result in flash crowds near many separate caches; all routers that get congested will invoke Local ACC independently. Unlike flash crowds, attacks are likely to use the IP addresses of the primary installation.

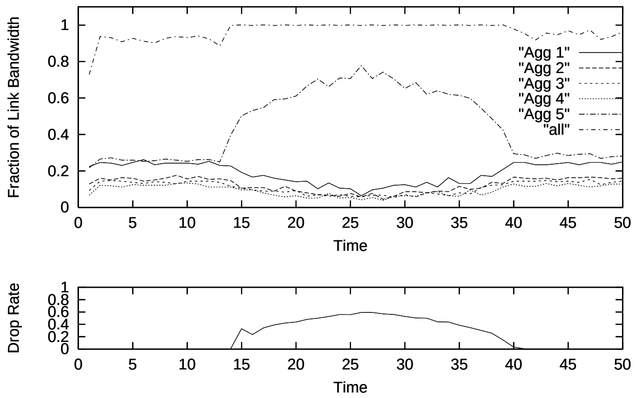

Figure 3: **A simulation without ACC.**

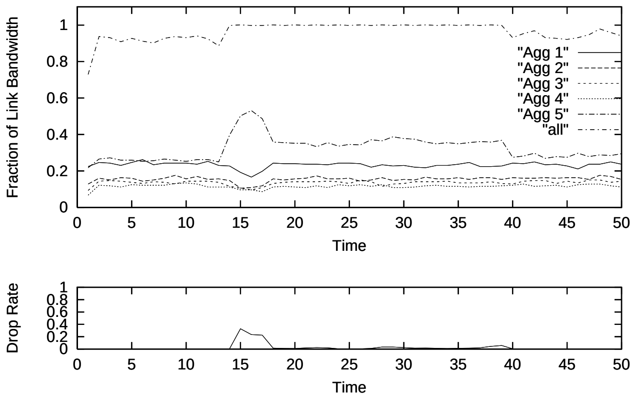

Figure 4: **The same simulation with Local ACC.**

3.2 Determining the Rate Limit for Aggregates

Using the list of high-bandwidth aggregates obtained above, the ACC Agent determines if some aggregate(s) should be rate-limited, and if so, what the limit should be. We describe this algorithm at a high level; pseudocode can be found in [19].

The ACC Agent sorts the list of aggregates, starting with the aggregate with the most drops. Next, it uses the total arrival rate at the output queue and the drop history to estimate the arrival rate of each aggregate over the last K seconds. This estimation assumes that drops are fairly distributed across aggregates. If this assumption was not true we would get aggregates that were false negatives or positives in identification. False negatives would be caught soon in a future round (aggregates cannot continue to be lucky given that RED is reasonably fair in distributing drops). Aggregates falsely identified are not harmed since their arrival rate is measured more carefully when they are rate-limited. Too many false positives are also avoided because, as discussed below, there is a limit on the number of identified aggregates.

Next, the ACC Agent calculates R_{excess}, the excess arrival rate at the output queue. This is the amount of traffic that would have to be dropped before the output queue (at the rate-limiter) to bring the ambient drop rate down to p_{target}. We now determine the minimum number of aggregates that need to be rate-limited to sufficiently reduce the total arrival rate. This is done using the constraint that the limit for rate-limited aggregates must be greater than the arrival rate of the largest non-rate-limited aggregate. The maximum number of rate-limited aggregates is at most *MaxSessions*.

Assume that the ACC Agent decides to rate-limit i aggregates. Now it computes the rate-limit L to be applied to each aggregate such that:

$$\Sigma_{k=1}^{i}(Aggregate[k].arr - L) = R_{excess},$$

where $Aggregate[k].arr$ is the arrival rate estimate of the $k-$th aggregate.

The above procedure requires that L is less than the arrival rate estimate of the i + 1-th aggregate, and that i is at most *MaxSessions*. Ideally, the Local ACC mechanisms should not rate-limit *any* aggregate during times of undifferentiated congestion caused by underprovisioned links or hardware failures. In the absence of effective methods for distinguishing between aggregate-based and undifferentiated congestion, we use the upper bound *MaxSessions* on the number of aggregates that are rate-limited simultaneously. With a better understanding of the traffic composition and behavior during DDoS attacks and flash crowds, we can tune the Local ACC mechanism such that it does not identify any aggregate in times of undifferentiated congestion.

The ACC Agent revisits its rate-limiting decisions periodically (every five seconds in our simulations), revising the rate limit L, and determining if some aggregate no longer needs to be rate-limited. The rate-limiter measures the arrival rate of rate-limited aggregates, so for the refresh phase, the ACC Agent has more precise information about the arrival rates. Aggregates that have had an arrival rate less than the limit for some number of refresh intervals are no longer rate-limited. Similarly, if congestion persists, more aggregates may be added to the list of rate-limited aggregates. No harm is done if rate-limiting continues for some time after the DDoS attack or flash crowd has subsided, because the rate-limiter only drops packets if the arrival rate is more than the specified limit. To avoid sudden changes in a flow's drop rate, the limit on an aggregate is not changed abruptly when another aggregate is added or removed from the list of rate-limited aggregates.

For an aggregate composed of congestion-controlled flows, the sending rate of the aggregate will be reduced in response to packet drops from rate-limiting at the router. However, the sending rate of the aggregate should remain above the imposed rate-limit in the long run.

3.3 Rate-limiter

The rate-limiter controls the throughput of the identified aggregates, and estimates their arrival rate using exponential averaging [27]. The rate-limiter is a pre-filter before the output queue that decides whether or not to drop each arriving packet of the aggregate based on the aggregate's arrival rate and the rate-limit.

Since the rate-limiter is in the forwarding fast path, it must be light-weight. Unlike strict lower priority queues, it does not starve the identified aggregates. Once past the rate-limiter, packets lose any identity as a member of the aggregate, and as a result are treated as regular arrivals to the output queue. Thus, rate-limiting cannot result in preferential treatment for the packets in the aggregate. In contrast, the rate-limited aggregates would get preferential treatment if they were allocated a fixed bandwidth share irrespective of the general congestion levels at the output queue. Rate-limited aggregates are protected from each other because the drop decision for each aggregate is taken independently.

We implement the rate-limiter as a virtual queue [13], which can be thought of as simulating a queue without actually queuing any packets. The service rate of the simulated queue is set to the specified bandwidth limit for the aggregate, and the queue size is set to the tolerated burst size. When a packet arrives at the rate-limiter, the rate-limiting mechanism simulates a packet arrival at the virtual queue. Packets that would have been dropped at the virtual queue are dropped by the rate-limiter, and packets that would have been queued at the virtual queue are forwarded to the real output queue.

3.3.1 Narrowing the Congestion Signature

In the discussion above, the aggregates identified by the ACC Agent are based only on source or destination addresses. The rate-limiter can do more sophisticated narrowing of the congestion signature that, in times of specialized attacks, can result in dropping more of the attack traffic within the aggregate. It can detect the more dominant signature within the aggregate, based on packet characteristics (such as port number or ICMP type code), and drop more heavily from this subset. Narrower rate-limiting can be achieved by placing another virtual queue, with a smaller service rate, in front of the aggregate's virtual queue.

Hierarchical rate-limiting described above is safe in scenarios where the attacker frequently changes her attack signature (but not the destination), as the total bandwidth available to the aggregate is still bound. Such specialized rate-limiting can be very useful for attacks such as the SYN attack [3] and the smurf attack [4].

One might perhaps argue that during flash crowds the routers should do some form of flow-aware rate-limiting, for example, dropping more heavily from SYN packets to provide better service to connections that manage to get established. However, such rate-limiting can be dangerous if used for a DDoS attack rather than a flash crowd. The attacker could simply send packets in the category being favored by the flow-aware rate-limiting (TCP data packets in the above example). Flow-aware rate-limiting is different from narrow rate-limiting mentioned above. While the latter punishes the dominant (relative to usual levels) packet type in the aggregate, the former favors a particular packet type, a strategy that can be gamed.

3.4 Simulations

We use a simple simulation with five aggregates to illustrate the effect of Local ACC. Figure 3 shows the simple simulation without ACC.[5] Aggregates 1-4 are composed of multiple CBR flows. The fifth aggregate is a variable rate source whose sending rate starts increasing at t=13, and then starts decreasing at t=25.

Figure 3 shows that, without ACC, the high-bandwidth aggregate is able to capture most of the link bandwidth. The bottom graph shows the packet drop rate in the output queue. Thus, when the sending rate of the fifth aggregate increases, the ambient drop rate increases and

[5] These simulations can be run with the commands "./test-all-pushback slowgrow" and "./test-all-pushback slowgrow-acc" in the tcl/test directory in the NS simulator. Similar simulations using TCP traffic can be run with the command "./test-all-pushback".

the bandwidth received by the other four aggregates decreases.

Figure 4 shows the same simulation with Local ACC. When the ambient drop rate exceeds the configured value of 10%, the ACC Agent identifies the fifth aggregate, and rate-limits that aggregate sufficiently to control the drop rate in the output queue. The bottom graph shows the ambient drop rate in the output queue, but does not show the drop rate in the rate-limiter for the fifth aggregate.

Similar results were obtained with TCP traffic in the background (Aggregates 1-4), and with TCP traffic as both background and high-bandwidth (the sending rate was increased by increasing the number of TCP flows).

4 The Pushback Mechanism

After detecting aggregate-based congestion, the ACC Agent must decide whether to invoke pushback, and if so, what limit should be specified to upstream routers. In this section, we address these and other pushback related issues.

4.1 Deciding when to Invoke Pushback

Pushback is invoked if the drop rate for an aggregate in the rate-limiter remains high for several seconds (because the arrival rate for the aggregate remains much higher than the limit imposed on it).[6] Pushback can also be invoked if there is other information that a DDoS attack is in progress. In some cases the packet drop history can help the router differentiate between DDoS attacks and flash crowds. For instance, if most of the packets within the aggregate are destined for a notorious UDP port, the router can be fairly certain that it is witnessing a DDoS attack and not a flash crowd. Another source of information can be the downstream server itself.[7] Pushback can be invoked by a router at the behest of a server directly connected to it.

4.2 Sending the Pushback Requests Upstream

Before invoking pushback, the ACC agents needs to divide the rate-limit for the aggregate among the upstream links. The division requires an estimate of the amount of aggregate traffic coming from each upstream link. Estimating each upstream link's contribution is easy with point-to-point links if the packets are marked in the router with the incoming interface. Section 6.2.2 explains how the upstream link's contribution can be estimated if routers connect using multi-access links, or no interface-specific marking is done in the router.

Based on its contribution, each upstream link is classified as either *non-contributing* (those that send a small fraction of the aggregate traffic) or *contributing*.[8] Because one of the motivations of pushback is to concentrate rate-limiting on the links sending the bulk of the traffic within the aggregate, the ACC Agent does not send a pushback request to non-contributing links. The assumption is that if a DDoS attack is in progress, the traffic on the non-contributing links is less likely to include attack traffic.

In the general case, contributing links do not all contribute the same amount of traffic. A link carrying more traffic belonging to the aggregate is more likely to be pumping in attack traffic. One of many possible algorithms, and the one used in our simulations, is to divide the total rate-limit among contributing links in a max-min fashion. For example, assume that we have three contributing links with arrival rates of 2, 5, and 12 Mbps, and that the desired arrival rate from the contributing links is 10 Mbps. The limits sent to each of the three contributing links would then be 2, 4, and 4 Mbps respectively.

After determining the limit for each upstream router, the ACC Agent sends a pushback *request* message[9] to them. The rate-limit specified in the request is only an *upper* bound on the bandwidth obtained upstream by that aggregate. If the upstream router itself becomes heavily congested, then it may give less bandwidth to the aggregate than the specified limit (Section 3.3).

We also note that the congested router could receive more than the desired amount of traffic in the aggregate if the non-contributing upstream neighbors (which were not sent pushback requests) start sending more traffic in the aggregate. However, since rate-limiting is also being done at the congested router, the congested link never carries more than the desired amount of aggregate traf-

[6] The high drop rate implies that the router has not been able to control the aggregate locally by preferential dropping, in an attempt to encourage increased end-to-end congestion control.

[7] The server can have some higher level or application-specific attack detection mechanism.

[8] The time constants for rate estimation are conservative to account for bursts.

[9] The pushback protocol, including timing and format of messages, is described in [8].

fic.

4.3 Propagating Pushback

On receiving a pushback request, the upstream router starts to rate-limit the specified aggregate just as it does for Local ACC, using the rate limit in the request message. The router's decision whether to further propagate the pushback request upstream uses similar algorithms to those described above. When propagating pushback upstream, the destination prefixes in the congestion signature have to be narrowed, to restrict the rate-limiting to traffic headed for the congested router only [19].

4.4 Feedback to Downstream Routers

The upstream routers that rate-limit an aggregate send pushback *status* messages to the downstream router, reporting the total arrival rate for that aggregate. Pushback status messages enable the congested router to decide whether to continue rate-limiting (and pushback).

The total arrival rate estimate of the aggregate is a lower bound on the rate the downstream router would receive if upstream rate-limiting were terminated. Because the rate-limiting (dropping) may have been contributing to end-to-end congestion control, terminating rate-limiting may result in a larger arrival rate for that aggregate.

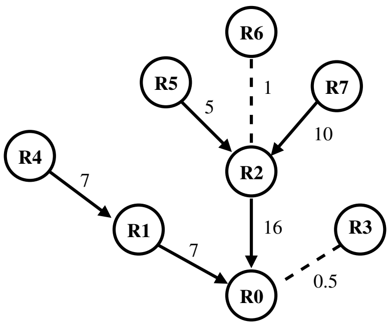

Figure 5: **Pushback status messages reporting the aggregate's arrival rate from upstream.**

The arrival rate reported in the pushback status message is the sum of the arrival rates reported in the status messages received from upstream, plus the arrival rates from the upstream non-contributing links. For example, in Figure 5, the labels for each solid line show the arrival rate estimate contained in the pushback status message. The dashed lines connect the non-contributing nodes that did not receive pushback request messages, and the labels on them show the arrival rate as estimated by the downstream router. Using the pushback status messages and its own estimate of the contribution from $R3$, $R0$ can estimate the total arrival rate for the aggregate as 23.5 Mbps. If $R0$ were to terminate the rate-limiting upstream, and invoke an equivalent rate-limiting locally, this would be roughly the arrival rate that $R0$ could expect from that aggregate.

4.5 Pushback Refresh Messages

The ACC Agent at the router uses soft state, so that rate limiting will be stopped at upstream routers unless refresh messages are received from downstream. For determining the updated rate limit in the refresh messages, the downstream router uses the status messages to estimate the arrival rate from the aggregate, and then uses the algorithms in Section 3 to determine the bandwidth limit. The arrival rates reported in the pushback status messages are also used by the downstream router in determining how to divide the new bandwidth limit among the upstream routers.

5 Simulations

This section shows a number of simulation results, obtained using the *ns* [22] simulator, that test the effect of Local ACC and pushback in a variety of aggregate-based congestion scenarios. These simulations do not pretend to use realistic topologies or traffic mixes, or to stress Local ACC and pushback in difficult or highly dynamic environments; the simple simulations in this scenario are instead intended to illustrate some of the basic underlying functionality of the ACC mechanisms as a first step towards a more rigorous evaluation.

We first introduce an informal terminology that would help us in describing the simulations. The *bad* sources send attack traffic to the victim destination D, and the *poor* sources are innocent sources that happen to send traffic to the destination D when it is under attack. In other words, packets from the poor sources represent the unmalicious traffic in the congestion signature. The *good* sources send traffic to destinations other than D.

5.1 ACC Mechanisms

Figure 6 shows the topology for simulations showing the difference in the dynamics of the ACC mechanisms. The good and the poor aggregates are each composed of seven infinite demand TCP connections. The bad sources each use a UDP flow with an on-off sending pat-

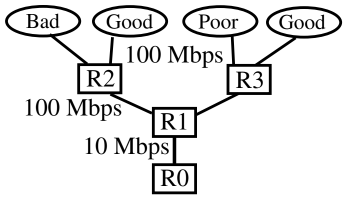

Figure 6: **Simple topology.** Link $R1$-$R0$ is congested.

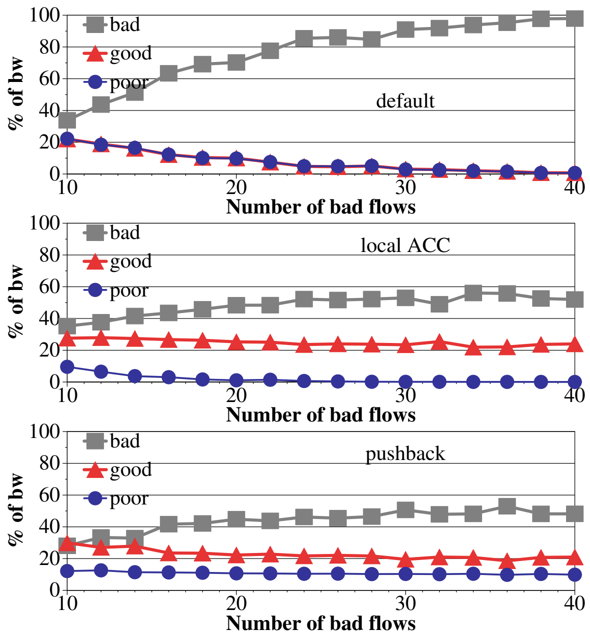

Figure 7: The throughput of different aggregates, in default (top), local ACC (middle), and pushback (bottom) scenarios.

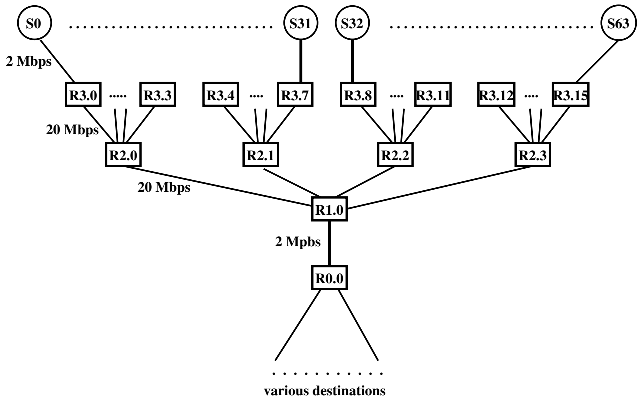

Figure 8: **The topology for sparse and diffuse attacks.** Link $R1.0 - R0.0$ is congested.

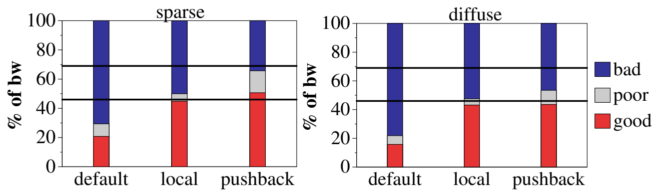

Figure 9: Bandwidth allocation at the congested link during sparse (left) and diffuse (right) DDoS attacks.

tern 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 gives 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 7 shows the results of simulations without ACC (default), with only local ACC, and with pushback. In the default 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, by pushing rate-limiting upstream where the bad and the poor sources can be differentiated, pushback protects the poor traffic as well as the good traffic.

5.2 DDoS Attacks

The simulations in this section illustrate Local ACC and pushback with both sparsely-spread and highly diffuse DDoS attacks. These simulations use the topology in Figure 8, with four levels of routers. Except for the router at the lowest level, each router has a fan-in of four. The top-most routers are each attached to four sources. The link bandwidths have been allocated such that congestion is limited to the access links from the source hosts and to the destination router.

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

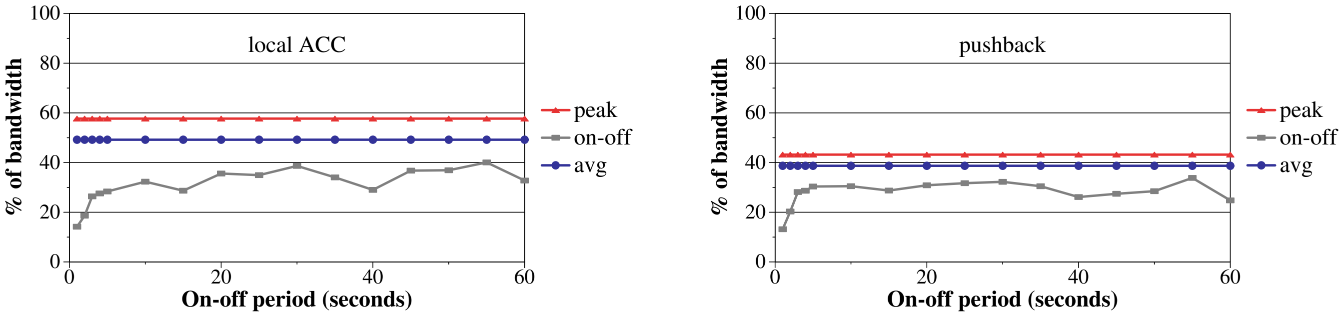

Figure 10: Throughput of the attack traffic in an on-off attack scenario.

each with an on-period sending rate of 0.25 Mbps.

Figure 9 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.

5.2.1 On-Off Attacks

ACC mechanisms can handle *on-off attacks*, in which the malicious sources don't send continuously. Plausible problems with on-off attacks are that they might generate additional control load (from the start and stop of rate-limiting) and the attacker might get away with more throughput if rate-limiting stops completely before the next *on* cycle starts. The aggregate release time after the aggregate stops being high-bandwidth is of the order of a small number of refresh intervals (Section 3.2; about 20 seconds in our simulations), so starting and stopping of rate-limiting cannot happen at a rate faster than this. Since continuing rate-limiting does not do any damage to an aggregate if it ceases being high-bandwidth, the release time could be increased to further reduce the control load in the face of such attacks. Moreover, the high-bandwidth aggregate detection time is very low, of the order of K seconds ($K = 1$ in our simulations). Thus, in an on-off attack there is a very small window of time during which the aggregate gets any significant throughput, after which the aggregate has to stop for a long time (till released) to get another such opportunity.

We simulated multiple on-off attack scenarios with different on and off periods. These included both low frequency cases, in which rate-limiting stopped and restarted between on periods, and high-frequency cases, in which rate-limiting never ceased. In these simulations all the bad sources have the same frequency and their on and off periods are synchronized. The results of the simulation are shown in Figure 10 with and without pushback. The two reference lines correspond to the cases when the attack sources were sending consistently at the peak rate and the average rate (half of peak rate). We can see that in all the cases, the on-off aggregate got much less throughput than when it was sending consistently at the peak or the average rate. The throughput for the on-off aggregate is less than the average rate scenario mainly because the on-off aggregate is controlled effectively during its on period, while it sends nothing during its off period, thus getting lower throughput on the whole.

5.3 Flash Crowds

This section shows simulations with flash crowds instead of DDoS attacks, with the "flash" traffic from 32 sources sending Web traffic to the same destination. The good traffic comes from ten other sources sending Web traffic to various other destinations, accounting for about 50% link utilization in absence of any other traffic.

Figure 11 shows the distribution of the times to complete the transfers for the good and the flash traffic respectively in the Default and Pushback mode. The distribution for Local ACC mode (not shown) was similar to the pushback. With Pushback, 80% of the good transfers

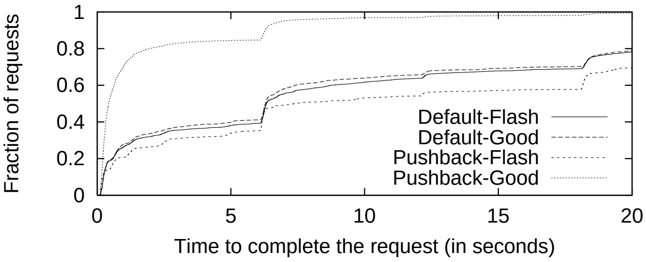

Figure 11: **Time to complete a request during a flash crowd.**

complete within a few seconds, compared to less than 40% completed in less than six seconds in the Default case. While pushback gives a significant performance gain for the good traffic, it also results in a fairly moderate degradation for the flash traffic. The time to complete a Web request can be directly correlated to the drop rate experienced. 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%, up to 33%. Because this simulation has much more flash traffic than good traffic, even a slight increase in the drop rate for the flash traffic frees up a lot of link capacity.

The hump around the 6-second mark represents short web transfers whose first SYN or SYN/ACK packet was lost, resulting in the transfer completing slightly more than six seconds later, after the retransmit timer expires. The magnitude and the location of the hump along the y-axis are good indications of the packet drop rates in the network for that aggregate. Recall that Local ACC and pushback are only invoked in scenarios of extreme congestion where the packet drop rate exceeds the configured threshold, set to 10% in our simulations, and at these levels of congestion a large fraction of transfers will have the first SYN or SYN/ACK packet dropped.

6 Discussion

6.1 Advantages and Limitations of Pushback

Pushback is not a panacea for flooding attacks. In fact, if not used carefully, it can make matters worse. This section discusses the advantages and limitations of adding pushback to ACC.

One advantage of pushback is to prevent scarce upstream bandwidth from being wasted on packets that will be dropped downstream.

When attack traffic can be localized spatially, pushback can effectively concentrate rate-limiting on the malicious traffic within an aggregate. This is very useful when source addresses cannot be trusted because then the congested router cannot narrow the congestion signature by itself.[10] In addition, if the offending traffic within an aggregate is heavily represented on some upstream link in the network, but the congested router cannot identify this subset of the aggregate based on the source IP addresses alone (i.e. the attack can be localized spatially but no concise description in terms of source prefixes exists), then pushback is necessary to narrow the attack signature *even if source addresses are genuine*.

For some DDoS attacks, pushback will not be effective in concentrating rate-limiting on the malicious traffic within an aggregate. For example, this would be the case for an attack uniformly distributed across the inbound links. Consider, for example, a reflector attack [23] based on DNS [5]. If sufficiently many reflectors are used from all portions of the network, the aggregate bandwidth will swamp the victim's link. During such an attack pushback will not be able to differentiate between the poor and the bad DNS traffic going to the destination, and will drop from both equally.

Pushback may overcompensate, particularly when it is invoked for non-malicious events such as flash crowds. If the overall demand from other traffic is reduced before the pushback refresh period expires (Section 4.5), then the upstream routers could unnecessarily drop packets from the high-bandwidth aggregate even when the downstream link becomes underutilized. With Local ACC, link underutilization is more easily avoided, as rate-limiting does not drop packets when the output queue is itself low. We reduce the possibility of overcompensation (and lower link utilization) with pushback by calculating the rate-limit of an aggregate so that the total traffic coming to the congested router is still greater than the capacity of the congested link (see the discussion of p_{target} in §3.2). Performing some of the rate-limiting just at the congested router can also help to prevent overcompensation.

In some cases, the use of pushback can increase the damage done to legitimate traffic from a source close to the

[10]If source addresses could be trusted, then in some cases the congested router could narrow the attack signature itself, by identifying both the source and the destination address prefixes responsible for the bulk of the traffic in the identified aggregate.

attacking host. As pushback propagates upstream towards the attack sources, the drop rate for the aggregate is increased. If pushback fails to reach a point where it can differentiate between the attack sources and the nearby legitimate traffic within the same aggregate, for instance, when the two sources are in the same edge network which is not pushback-enabled, the legitimate traffic at that point will share the same high drop rate as the attack traffic. This property of pushback could lead to potential DDoS attacks in which the attacker's aim is to hinder a source from being able to send to a particular destination. To be successful, an attacker would need to launch the attack from a host close to the victim source. However, the ability to compromise a machine that shares a downstream bottleneck link with the victim enables many other forms of attack anyway.

6.2 Implementation and Operational Issues

6.2.1 Implementation Complexity

The identification of aggregates can be done as a background task, or in a separate machine entirely, so the processing power required to identify aggregates should not be an issue. However, the presence of a large number of rate-limited aggregates could pose a design challenge. When a packet arrives at the output queue, the router has to determine if that packet belongs to one of the rate-limited aggregates, and if so, place it in the correct virtual queue. The time required for this lookup may increase with an increasing number of aggregates. We do not expect the limitation on the number of rate-limited aggregates to be a problem, as we envision Local ACC and pushback as mechanisms to be instantiated sparingly, in times of high congestion, for a handful of aggregates. But a deployed system needs to be robust against new attacks that could generate many rate-limited aggregates. One possible approach would be to use the routing table of the router for detecting membership in a rate-limited aggregate; however, this would restrict the definition of aggregates to destination prefixes.

6.2.2 Estimating the Upstream Link's Contribution

The distribution of the rate-limit among upstream links depends on the downstream router's ability to estimate what fraction of the aggregate comes from each upstream router. This is simple for point-to-point links; only one router is attached to each interface. However, for routers joined by LANs, VLANs, or frame relay circuits, there are multiple routers attached to an interface. The downstream router in this situation might not be able to distinguish between multiple upstream routers.

One way of dealing with this problem is to send a *dummy* pushback request to all upstream neighbors. The dummy request is similar to the real request, but the recipient does not actually rate-limit the aggregate. The only impact of this request is that the recipient will estimate the arrival rate of the specified aggregate and report it to the downstream router in status messages. These messages help the downstream router to send pushback requests with the appropriate rate-limits to contributing routers.

6.2.3 Incremental Deployment

Pushback can be deployed incrementally by deploying it only on the edges of an *island* of routers, where an island is a set of connected routers. An autonomous system (AS) is a natural island. Assume that the island has N edge routers. When one of these routers gets congested and decides to invoke pushback, it could consider the remaining $N - 1$ edge routers as its upstream links. Using dummy pushback messages it could ascertain the aggregate's arrival rate at each of these routers, and send authenticated pushback request messages accordingly. Thus, even without universal deployment the island can selectively throttle traffic coming in from certain directions.

7 Related Work

Two common mechanisms to counter DDoS attacks are ingress filtering [7] and traceback [1, 24, 26]. ACC is orthogonal to both of them; the focus of ACC is neither to stop the attacks (ingress filtering tries to stop attacks that use source address spoofing) nor to find the sources of these attacks (traceback), but to minimize the immediate damage done by high-bandwidth aggregates. In the presence of ACC mechanisms, we expect the damage control (by preferential dropping of the high-bandwidth aggregate) to trigger in much sooner than the time it takes to identify and stop the malicious sources.[11]

The network can use traceback mechanisms along with ACC mechanisms if identification of malicious sources

[11] The fact that ACC, in both its local and pushback incarnations, gently restrains aggregates to the point where they are no longer causing congestion allows ACC to respond rather quickly because the downside of an inaccurate assessment of the offending aggregate is slight. DDoS countermeasures that completely shut down the attacking traffic must be much more confident in their identification before they take action.

is desired. For instance, packets from the high-bandwidth aggregate that are not dropped can be probabilistically marked as in [24], or every incoming packet can be included in the message digest before rate-limiting as in [26].

Schnackenberg et al.[25] suggest active control of infrastructure elements. Thus, a firewall or Intrusion Detection System (IDS) that detected a DDoS attack could request that upstream network elements block the traffic. The paper defines a protocol to express the interactions between routers. It does not deal with mechanisms to identify or rate-limit malicious traffic, a subject of our work.

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 completely prevent network congestion from flash crowds. For example, flash crowds could occur for traffic not carried by CDNs, or for traffic marked as uncacheable by the origin server, or for traffic that is not suitable for multicast distribution. Internet slowdowns could still be caused by an event or site that witnesses an unprecedented "success" for which neither it nor the related infrastructure is prepared.

Flow-based congestion control is related to ACC but operates at a different granularity. This includes per-flow scheduling mechanisms like Fair Queuing [6], and mechanisms that use preferential dropping to approximate per-flow scheduling [27] or to protect conformant flows from misbehaving flows [9, 18, 20]. However, flow-based congestion control is not a solution for aggregate-based congestion, since an aggregate could be composed of many flows that are low-bandwidth individually.

CBQ [11] is a class-based scheduling mechanism in which aggregates can be limited to a certain fraction of the link bandwidth during congestion. However, CBQ is discussed largely for fixed definitions of aggregates, and does not include mechanisms for detecting particular high-bandwidth aggregates during congestion.

The flow-based congestion control scheme that comes closest to pushback is credit-based flow control [17]. Both mechanisms send messages upstream specifying how much traffic of a certain category it should send. However, other aspects like which categories to limit and how those limits are computed are completely different.

8 Conclusions

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 DDoS mechanism itself. A technique to incrementally deploy pushback is presented in [19]; a prototype implementation can be found in [15].

For effective evaluation we need measurement-based answers to questions like "how frequently is sustained congestion caused by aggregates, and not by failures", and "what do attack traffic and topologies look like". For example, 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. Possible policies 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 the maximum throughput of an aggregate. These policies could be based on known traffic patterns at the router, on contractual relationships of the ISP, or on local policies regarding the response to DDoS attacks. For example, an ISP might have a contractual relationship with a content provider not to unduly restrict the bandwidth to or from that content provider during a flash crowd or DDoS attack.

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, Neil Spring, Ed Vielmetti, and anonymous reviewers for feedback on an earlier draft of this paper.

References

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

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

[3] CERT Web Pages: CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks. http://www.cert.org/advisories/CA-1996-21.html, September 1996.

[4] CERT Web Pages: CERT Advisory CA-98.01 "smurf" IP Denial-of-Service Attacks. http://www.cert.org/advisories/CA-98.01.smurf.html, January 1998.

[5] CERT. CERT Incident Note IN-2000-04, 2000. http://www.cert.org/incident_notes/IN-2000-04.html.

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

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

[8] S. Floyd, S. Bellovin, J. Ioannidis, K. Kompella, R. Mahajan, and V. Paxson. Pushback Messages for Controlling Aggregates in the Network. Work in progress. Internet-draft: draft-floyd-pushback-messages-00.txt, July 2001.

[9] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, August 1999.

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

[11] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, Vol. 3(4):pp. 365–386, August 1995.

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

[13] 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.

[14] A. S. Induruwa, P. F. Linington, and J. B. Slater. Quality of Service Measurements on SuperJANET - The UK Academic Information Highway. In *Proc INET'99*, June 1999.

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

[16] 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.

[17] 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 *ACM SIGCOMM*, August 1994.

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

[19] 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/, July 2001.

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

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

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

[23] V. Paxson. An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks. *CCR*, vol. 31(3), July 2001.

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

[25] D. Schnackenberg, K. Djahandari, and D. Sterne. Infrastructure for Intrusion Detection and Response. In *Proceedings of the DARPA Information Survivability Conference and Exposition 2000*, March 2000.

[26] 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 *ACM SIGCOMM*, August 2001.

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