Short-Circuiting the Congestion Signaling Path for AQM Algorithms using Reverse Flow Matching [⋆]

Mart Molle [a] and Zhong Xu [b]

[a]*Department of Computer Science & Engineering*

[b]*Department of Electrical Engineering*

University of California, Riverside, CA 92521, USA

Abstract

Recently, we introduced a new congestion signaling method called *ACK Spoofing*, which offers significant benefits over existing methods, such as packet dropping and Explicit Congestion Notification (ECN). Since ACK Spoofing requires the router to create a "short circuit" signaling path, by matching marked data packets in a congested buffer with ACK packets belonging to the same flow that are traveling in the opposite direction, the focus of this paper is evaluating the feasibility of reverse flow matching. First, we study the behavior of individual flows from real bi-directional Internet traces to show that ACK Spoofing has the potential to significantly reduce the signaling latency for Internet core routers. We then show that reverse flow matching can be implemented at reasonable cost, using essentially the same hardware as the packet filtering logic commonly employed in Layer 2 transparent bridges. Finally, we show that this architecture can be scaled to accommodate worst-case traffic patterns on multi-gigabit links that would render ordinary route caching algorithms completely ineffective.

Key words: ACK Spoofing, active queue management, route caching, signaling, packet marking, TCP congestion control, flash crowd

[⋆] This work was supported in part by Nortel Networks Inc, and by the Industry-University Research Program of the University of California under grant DiMI00-0061

Email addresses: `mart@cs.ucr.edu` (Mart Molle), `zhong@cs.ucr.edu` (Zhong Xu).

URLs: `www.cs.ucr.edu/~mart` (Mart Molle), `www.cs.ucr.edu/~zhong` (Zhong Xu).

1 Introduction

Active Queue Management (AQM) schemes for IP routers, in combination with congestion avoidance algorithms for TCP sources, play a fundamental role in improving QoS for network services. The key idea behind AQM is that the router must adopt a more proactive congestion control policy, in which it tries to gradually signal the onset of congestion before its queue has become completely full. In this way, TCP sources are forced to decrease their transmission rates before severe congestion occurs. A well-designed AQM algorithm could yield better fairness, much lower queueing delay and possibly higher throughput than Tail Drop[1]. Published AQM algorithms include RED[2], BLUE[3], REM[4], SRED[5], FRED[6], etc. In addition, there have been many efforts on combining existing AQM algorithms with fbw classifi cation, prioritizing and packet marking techniques, so as to provide some kind of QoS in the router.

However, most of the work on congestion control has focused on the signaling policy, rather than the signaling mechanism itself. Recently, we have found that the congestion signaling mechanism has a signifi cant impact on network Quality of Service, and that the effect of improving the signaling method can be as large as changing AQM algorithms. *Packet dropping* is widely used as an implicit congestion signaling method. However, packet dropping is *expensive*, in the sense that it wastes a signifi cant amount of network resources. Moreover, packet dropping may cause timeouts, which can drastically reduce the throughput of the targeted stream [7]. On the other hand, *Explicit Congestion Notification* (ECN) simply marks some ECN control bits in the header of the target packet and then allows it to continue through the network [8][9]. Thereafter, the ECN markings that reach the TCP receiver are returned to the TCP sender through the acknowledgement stream. ECN signaling was fi rst introduced in combination with RED, but it has subsequently been adopted by several other AQM algorithms, such as REM and BLUE. There is also ongoing research on how to mark ECN bits effi ciently and fairly [10][11][12]. If carefully designed, AQM algorithms with ECN signaling gain several benefi ts, including smaller queueing delays, less packet losses, and improved effective transmission throughput [8][3][4][13][14].

Unfortunately, ECN signaling suffers from a serious deployment problem because it is not supported by existing IP routers and TCP implementations. Incremental deployment of ECN would create a mixture of ECN-compatible traffi c (i.e., fbws with ECN-capable TCP implementations in both end hosts) and ECN-incompatible traffi c. Such heterogeneous systems can lead to severe fairness problems, even though we preferentially adopt ECN signaling for ECN-compatible fbws and packet dropping for ECN-incompatible fbws [15]. Therefore, we have recently introduced another new congestional signaling method called *ACK Spoofing* [16], which is compatible with existing TCP implementations. In the following sections we describe ACK Spoofi ng and demonstrate via simulation that it provides a signifi cant

QoS improvement over both ECN signaling and packet dropping. However, the creation of spoofing ACKs requires the router to capture state information about the target flow from the ACKs traveling in the reverse direction. Thus, the main focus of this paper is to investigate the feasibility of implementing reverse flow matching in Internet core routers.

The rest of this paper is organized as follows. Section 2 provides a brief introduction to ACK Spoofing, together with its associated on-demand state maintenance scheme, and signal cancellation enhancement mechanism. We also provide a few illustrative examples, obtained via simulation, to demonstrate its performance. Since ACK Spoofing's performance advantage comes from reducing the congestion signaling delay, in section 3 we study some Internet traces to estimate its potential benefits in the real world. We then turn our attention to evaluating the implementation complexity for ACK Spoofing. In section 4 we provide a brief introduction to IP routing, and focus on aggregate flows and/or flash crowds to model the worst-case traffic pattern for a congested router. Although these worst-case traffic patterns would render route caching completely ineffective for high speed Internet core routers, we show in section 5 that reverse flow matching can be done quite easily under the same conditions. Moreover, it can be implemented very efficiently using the same hardware components used for packet filtering in layer 2 switches. Finally, we give our conclusions in section 6.

2 ACK Spoofing and its Performance

Almost all current TCP implementations are based on the TCP Reno or later releases, which incorporate the fast retransmit and fast recovery mechanisms. These mechanisms cause the TCP sender to reduce its congestion window size by half after receiving multiple duplicate ACKs. In ACK Spoofing, this duplicate-ACK response is artificially triggered by the router as a congestion signaling method. Whenever the AQM algorithm targets a particular TCP flow to receive a congestion signal (or the router is forced to drop a packet due to buffer overflow), the router sends multiple artificially-generated duplicate ACK packets (called *spoofing ACKs*) to the corresponding TCP sender. Upon receiving the spoofing ACKs, the TCP sender will be tricked into immediately reducing its sending rate and retransmitting the "missing" packet. However, unless the packet was actually dropped, the retransmission is just a needless duplicate that can be discarded at the router.

Note that setting the ack number carried by the spoofing ACKs to the proper value is critical to the operation of ACK Spoofing. A value that is smaller[1] than in previously-seen ACK packets for the same flow might cause the TCP sender to

[1] Using modulo arithmetic to handle wrap around of ACK and sequence numbers, of course.

ignore the spoofi ng ACKs (delayed, out-of-order), while larger ack numbers would compromise reliability of the TCP session and possibly even deadlock. Therefore, to generate spoofi ng ACKs the router must include some state variables obtained from real ACK packets traveling over the reverse path, i.e., ack number (*th_ack*) and advertised window size (*th_win*). In the ideal case, the router would simply maintain per-fbw state information about every active fbw all the time — so it could instantly generate spoofi ng ACKs for any of those fbws. Clearly, this would generate considerable processing overhead and possibly reduce the router's throughput. However, we can drastically reduce this state-maintenance overhead (at the cost of increasing the signaling latency) by adopting an *On-demand State Maintenance* scheme, in which the router only tracks the state variables for a given fbw during a short time period after one of its packets has been targeted by the AQM algorithm.

We can use the signaling latency caused by on-demand state maintenance to our advantage in the following way. Consider the time delay between the *decision to target* a given fbw and the *opportunity to send the congestion signal*, when the router fi nds a matching ACK packet in the reverse fbw from which to extract the state variables for the spoofi ng ACKs. If the congestion problem at the router has cleared itself during the time, then this congestion signal was not really needed—and sending it now might even be harmful if it causes the bottleneck link to become underutilized. In this case, ACK Spoofi ng naturally[2] gives us an opportunity to reevaluate the packet-marking decision, and to *cancel the congestion signal* if later conditions indicate that it is not needed, i.e., if the buffer occupancy at the router drops below some threshold.

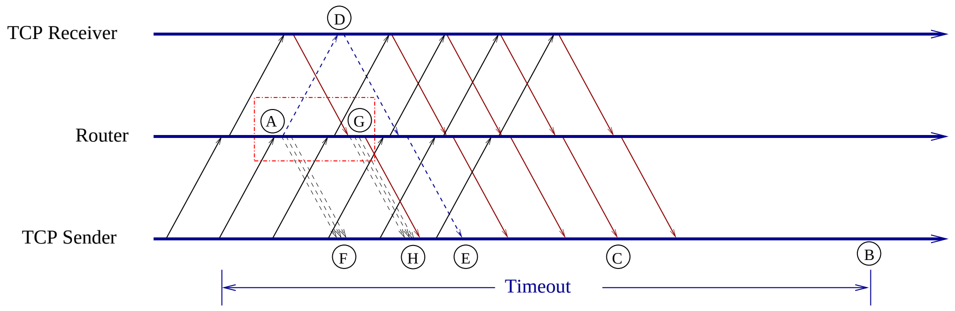


Fig. 1. Feedback latencies of different congestion signaling methods

Figure 1 provides an example to illustrate the operation of ACK Spoofi ng, and to compare the path lengths for different congestion signaling mechanisms. For example, suppose the router decides to drop a packet at point A. Then the TCP sender will discover the packet loss at either point B, if a timeout occurs, or at point C, if it can be detected using the Fast Retransmission algorithm. Conversely, if the router instead uses ECN to mark the packet at point A, then the congestion signal would pass through the destination at point D before reaching the sender at

[2] Although signal cancellation could be combined with ECN signaling, it would drastically increase its implementation complexity (because of the need for reverse-flow matching) without doing anything to improve ECN's intrinsic deployment problems.

point E. Finally, suppose the router uses ACK Spoofing as its congestion signaling mechanism. In the ideal case the router would send the spoofing ACKs at point A, as soon as its AQM algorithm decided to mark the packet, causing the sender to respond to the congestion signal at point F. However, a more practical implementation would rely on the on-demand state maintenance algorithm to reduce its processing overhead, in which case the router would just begin searching for the next matching ACK packet belonging to the target flow at point A. In this example, the matching ACK packet arrives at point G, which triggers the router to check the signal cancellation policy to make sure that the congestion signal is still needed. If so, it immediately generates the associated spoofing ACKs, then stops tracking this flow. Finally, the congestion signal reaches the sender at point H.

Clearly, since ACK Spoofing is just another signaling method (similar to packet dropping and ECN) that can be used by any AQM algorithm to rate-control responsive TCP flows, we must ask ourselves whether this choice of methods has a meaningful effect on performance. To address this question, we have conducted a series of careful simulation experiments to investigate the sensitivity of two well-known AQM algorithms to different signaling methods: Random Early Detection (RED) [2] and Random Exponential Marking (REM) [4] (Please refer to the original publications for algorithm details). For each AQM algorithm, we use the parameter values given in Table 1.

Table 1
Simulation parameters for buffer size 120

RED Parameter	min_{th}	max_{th}	max_p
RED	10	90	0.02
RED/ECN	10	90	0.10
RED/Spoofing	10	90	0.05

REM Parameter	ϕ	α	γ	b^*
Value	1.001	0.1	0.001	40

Due to space limitations, we can only show one experiment to illustrate our results; for more details, please see [16][15]. We used the commercial simulation package CSIM-18 [17] to construct our simulator, and our implementation of TCP Reno faithfully follows the model by Stevens et al. [18]. The simulation network used here is given in Fig.2, where all link speeds are 100Mbps and the propagation time (in milliseconds) is shown on each link. In this experiment, we set up four groups of 10 individual TCP flows, including 10 flows from H0 to H5, 10 flows from H1 to H4, 10 flows from H2 to H4, and 10 flows from H3 to H5 respectively. Here, the bottleneck is the link from R2 to R3, so only the simulation results related to the bottleneck link will be given.

Fig.3 shows the dynamics of queue sizes with different signaling methods. From the graphs, we can find that Tail Drop suffers from the full-queue problem, and the

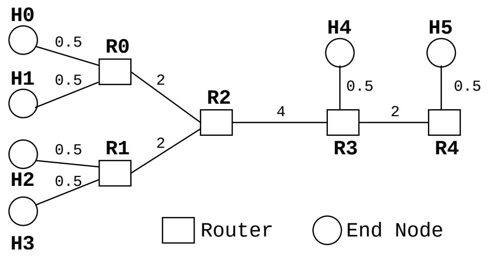
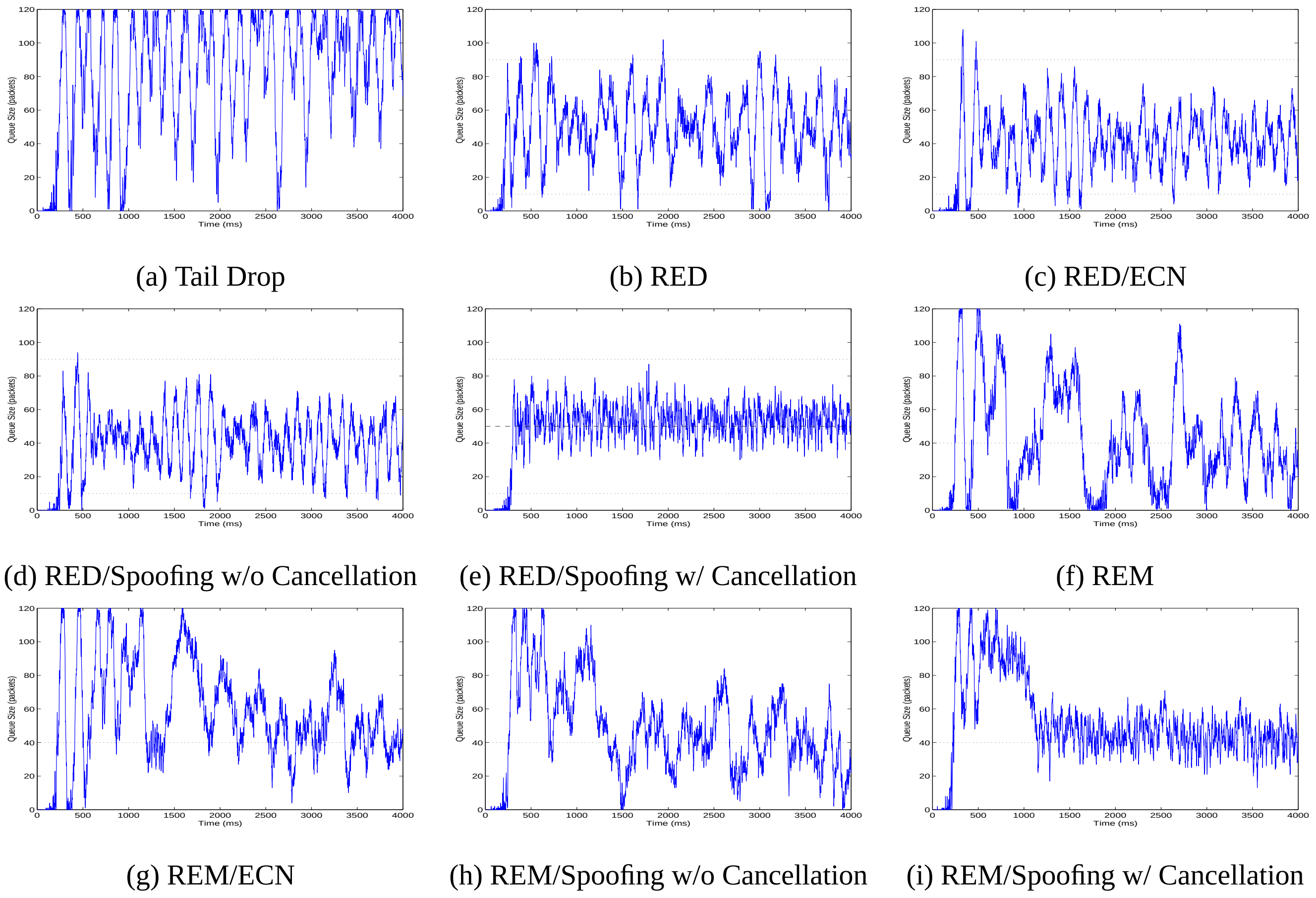


Fig. 2. The simulation network

AQM algorithms can solve this problem. We also fi nd that signaling methods have some impacts on the stability of queue sizes. ECN and ACK Spoofi ng (even without signal cancellation) yield noticeably better control of queue sizes, no matter what AQM algorithm it is associated with. Moreover, the congestion signal cancellation mechanism is very useful in maintaining a much more stable queue size dynamics. In the application of QoS, stable queue sizes are especially desired, because stable queue sizes mean stable queueing delays and thus could yield smooth packet delivery (e.g. less jitter in video/audio streaming service).



(a) Tail Drop (b) RED (c) RED/ECN

(d) RED/Spoofing w/o Cancellation (e) RED/Spoofing w/ Cancellation (f) REM

(g) REM/ECN (h) REM/Spoofing w/o Cancellation (i) REM/Spoofing w/ Cancellation

Fig. 3. Dynamics of queue sizes for different signaling methods and AQM algorithms

In other experiments not shown here, we also found that different signaling methods exhibit different fairness properties in terms of convergence speeds and stabilities of bandwidth allocation. While packet dropping has signifi cant oscillation in bandwidth and ECN has better performance but still exhibits slow convergence in some cases, ACK Spoofi ng (both with and without signal cancellation) yields much more consistent performance across all simulated cases, including superior convergence speed and stability properties.

In addition, we have studied the impact of lost congestion signals under ECN or ACK Spoofi ng. We found that even with about 25-35% of ACK packets losses, ACK Spoofi ng and ECN can still maintain very high goodput and reasonable fairness among fbws, while packet dropping begins to exhibit severe unfairness of bandwidth allocation at ACK packet loss rate of about 15-25%. The good performance of ACK Spoofi ng on resisting ACK packet losses is very important, since its congestion signal is carried on the (spoofi ng) ACKs and ACK packet losses is unavoidable due to congestion in the Internet.

3 Characterizing the "Short Circuit" Signaling Path using Internet Traces

In this section we address the following question. *For a busy Internet core router, how much reduction of the congestion feedback latency can we possibly gain by implementing ACK Spoofing?* By studying two very different bi-directional Internet traces, we will now show that signifi cant latency reductions, as described in fi g. 1, should be possible using ACK Spoofi ng in the real world.

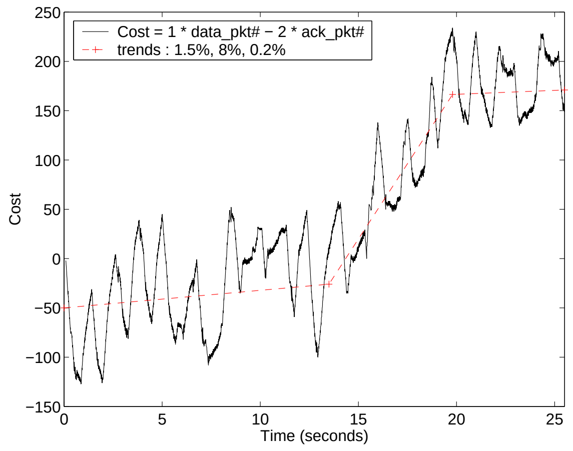


Fig. 4. Relationship between data and ack packet numbers

Fig.4 shows a trace of a single very large bi-directional TCP fbw recorded on an OC-48 link monitored by CAIDA. This fbw spanned the entire duration of a 25-second trace fi le, during which the fbw carried approximately 6,800 data packets and 3,800 ACK packets [19]. Although typical large TCP fbws are rarely able to sustain such high bandwidth across the Internet, it is an excellent illustration of the chaotic network conditions experienced by a TCP session in the absence of QoS support.

In this figure, we show an estimate as a function of time for the current number of unacknowledged TCP segments belonging to this flow. Most TCP implementations adopt a mechanism called *delayed ACK* [20], in which the TCP receiver sends one ACK after receiving each pair of data packets (unless a timeout expires, or the arriving packet is out of order). Thus, we *add one* to the estimate each time we see a data packet, and *subtract two* from the estimate each time we see an ACK packet. We can see trends in the data at two different scales. First, at the *local* time scale we see an alternating pattern of "peaks" and "valleys." For example, it takes approximately 0.5 seconds for the curve to rise from the leftmost "valley" to adjacent "peak", during which time we saw 258 data packets, but only 55 ACKs returned. However, during the next 0.5 second period, we saw only 91 data packets, while 88 ACKs returned. Since the final result is to (almost) return the estimate to its previous level, we conclude that there must large numbers of unacknowledged packets (perhaps as many as 150 in this case) between our measurement point and the TCP destination during those "peak" periods. On the other hand, if we follow the estimate over global time scales, there is a clear long-term increasing trend in the data. We attribute this long-term trend to packets that are dropped downstream from the measurement point, for which we never see an acknowledgement. Moreover, the slope of the trend line changes over different regions of the graph. In the best case (where each ACK signals the arrival of two additional data packets), these slopes correpsond to respective packet loss rates of 1.5%, 8% and 0.2%. Conversely, in the worst case (where each ACK signals the arrival of one out-of-order data packet) these slopes would correspond to packet loss rates of more than 50%. In any case, this flow is clearly experiencing a significant amount of congestion and/or packet loss somewhere downstream from the measurement point. Moreover, ACK Spoofing would provide a significant performance advantage for routers attempting to control this flow, by allowing their congestion signals to "jump ahead" by dozens of packets during its peak traffic periods.

Let us now look at the well-known LBL-TCP-3 trace, which consists of approximately 1.8 million TCP packets collected by V. Paxson[21] in 1994. This trace covers two hours of bi-directional traffic recorded at the gateway between Lawrence Berkeley Laboratory (LBL) and the global Internet. Although it is quite old, it is one of the few publicly available traces online that records bi-directional traffic traveling in enough detail to calculate the latency distributions and analyze their relationships.

The LBL gateway connects their local area network to the wide-area Internet. Thus, we must distinguish between two kinds of flows passing through this gateway. *Inward* flows correspond to remote TCP senders, which may be located anywhere throughout the global Internet, establishing connections to local TCP receivers located within LBL's local network. Conversely, *outward* flows correspond to local TCP senders within the LBL local network establishing connections to remote TCP receivers located somewhere else in the global Internet. From the viewpoint of the gateway, TCP packets coming from the hosts inside the LBL network should ex-

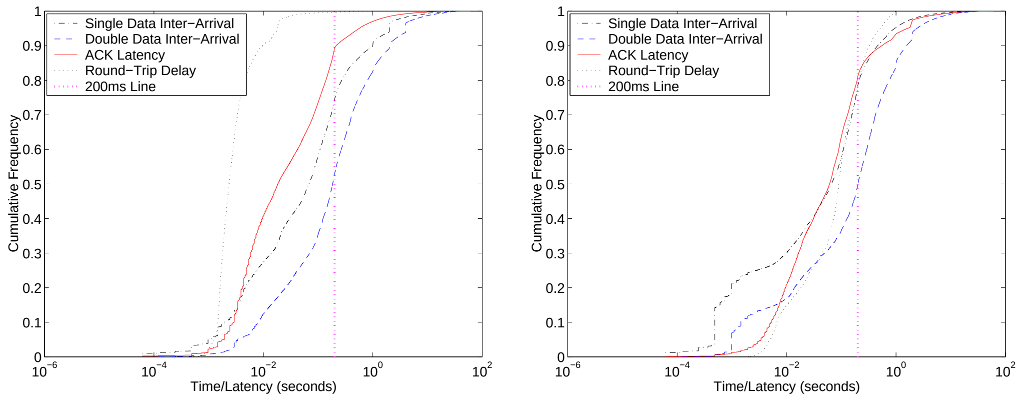


Fig. 5. Inter-arrival time and latency distributions of inward and outward flows

hibit similar delay characteristics because the distances spanned by the LBL local network are quite small and the available bandwidth each local link tends to be high. However, packets from outside nodes should exhibit significantly higher latencies, together with a much wider variance of delay.

Fig. 5 shows the cumulative distribution functions for the data packet inter-arrival time, ack latency, and round-trip delay for inward (left) and outward (right) TCP flows. In each graph, the *round-trip delay* between the gateway and TCP receiver represents the sum of the one-way delivery times from the gateway to the TCP receiver and from the TCP receiver back to the gateway. To avoid biasing the measurements due to processing delays at the TCP receiver, it is calculated during TCP's opening three-way handshake by measuring the elapsed time between the arrivals of the initial SYN packet and middle SYN/ACK packet. The *ack latency* is defined as the elapsed time between the arrival of a data packet in the forward direction and the arrival of the next ACK packet belonging to the same flow in the reverse direction. The *single* and *double* data packet interarrival times represent the elapsed time between either the kth and $k+1$st data packets, or the kth and $k+2$nd packets, respectively, belonging to the same flow.

From the graphs, we can find that two kinds of flows exhibit different distributions. Obviously, the network distance between the gateway and any host inside the LBL local network must be quite small, whereas the path between the gateway and an outside host may be much longer. Hence, inward flows have small and consistent round-trip delay, but the round-trip delays for outward flows are much less predictable and their mean is nearly two orders of magnitude greater. The difference of locations also affects the distributions of ACK latencies, which is more important in estimating the performance of our ACK Spoofing scheme. However, the distributions of the ACK latencies for both kinds of flows have a similar shape, but with different means.

Now we need to determine whether the distribution of the ack latency might be small enough to allow reverse flow matching to give us a significant reduction in congestion signaling delay. In the case of inward flows, this clearly is not the case, since the round-trip delay is an order of magnitude smaller than the ack latency, which is itself noticeably smaller than the single data packet interarrival time. What this tells us is that no significant reduction in congestion signaling is possible for the inward flows because we are flow-matching the ACK packet associated with our marked packet, which is the same ACK packet that would have carried the ECN bit. The data also shows that the LBL servers must be using the *delayed ack* mechanism [20]: notice how the ACK latency distribution seems to be interpolating between the distributions for the round-trip delay and single data packet interarrival time because half the time the TCP receiver waits for another data packet to arrive before generating an ACK packet. As expected, the ACK latency distribution shows a steep increase near the delayed-ACK timeout of 200 msec. (shown as a vertical line in the figure), corresponding to delayed ACKs sent for a single data packet after the timeout for the next data packet expired. In addition, the long tail forming the last 10% of the distribution for ACK latency, together with an even longer tail for the single packet interarrival time distribution (dominated by a pair of large jumps at approximately 1 and 2 seconds, which are likely the result of retransmission timeouts) provides strong evidence of significant packet losses within the LBL local network.

On the other hand, the ACK latency from outward flows is much smaller than the round-trip delay, so the first ACK packet we see is associated with an earlier data packet that must have passed through the router before our marked packet. In fact, the speedup in congestion signaling appears to be approximately 4–5 times faster than waiting for the same ACK to return within the second quartile of the distribution. Note, also, that the initial jumps in the single and double packet interarrival time distribution functions at approximately 500 μsec. and again at 1 msec. can also be used to estimate the average size of the TCP congestion window that applies to fig.5. As we already discussed previously, the relatively large initial jump in the single data packet interarrival time distribution at 500 μsec. is likely because of the delayed ack mechanism, which means that the TCP sender can transmit two back-to-back data packets after receiving a single ACK packet.[3] However, this mechanism does not explain the small jump in the double data packet interarrival time distribution at approximately 1 msec. Instead, that is caused by the additive increase phase of TCP's congestion avoidance algorithm, which opens the congestion window by one packet after completing the successful transmission of an entire window's worth of other data. Since the height of this initial jump is approximately 15%, we can conclude that the average window size is about 7 packets. Since almost all of the outstanding packets from a given TCP session's congestion window will be somewhere beyond the LBL gateway in the global Internet, we can conclude that the LBL gateway will be on the "wrong" side to make them targets

[3] At the time that this trace file was recorded, many TCP implementations used the Internet default Maximum Segment Size of 576 bytes, which corresponds to a packet transmission time of about 500 μsec. on a 10 Mbps Ethernet link.

for ACK spoofing. However, if the characteristics of the inward and outward flows were mirror images of each other, it would mean that the LBL gateway would be able to "skip ahead" by an average of about 3 ACK packets (i.e., because of delayed acks, we must use half the average congestion window size, in units of a data packet) when it attempts to use reverse flow matching to rate control TCP flows carrying data from an internal LBL sender to an outside receiver.

Based on the above analysis of packet interarrival time distributions, we can see that ACK Spoofing is most effective for allowing a router to control a nearby sender trying to send data to a distant receiver. Since an Internet core route is likely to be far away from most hosts, ACK Spoofing should be generally very effective in the Internet backbone. In the following sections, we will describe some extended techniques for ACK Spoofing, which could further reduce the overall feedback latency and thus gain more even when the destination is close to the router.

4 Packet Forwarding in IP Routers

Each time an IP router receives a packet through one of its input ports, it must execute a series of packet processing functions in order to determine how to forward that packet one step closer to its final destination [22]. Before the packet leaves the input port, the IP header fields are updated to reflect the reduction in its time-to-live by one "hop", and the *packet classifier* extracts the relevant fields from the packet header by which it determines the fate of this packet. In general, the classifier needs to identify (at least) the *destination IP address*, and possibly also such additional information as the *Protocol* (e.g., TCP, UDP, ICMP, or some other protocol), its *DiffServ/TOS tag*, its *Application type* (based on well-known port numbers), the *presence of certain flags* (SYN, ACK, etc), or its unique *flow ID* (i.e., the 4-tuple consisting of the source IP address and port number, plus the destination IP address and port number).[4] This packet header information is then used as the input to the IP routing lookup function (i.e., longest common prefix matching in the routing table) to select the appropriate output port, and to the access control and/or QoS classification policies (if any) to decide whether to block this flow or assign it to one of the available priority classes at the output port.

Once the packet classification is completed, the IP datagram will be transferred from the input port to the output port through the router's internal switching fabric (eg., crossbar, TDM bus, shared memory, banyan-type interconnection network, etc). A discussion of the many implementation choices available for creating this

[4] Although we could also consider some fields from the Layer 2 header during the packet classification process —such as the source and destination MAC addresses, VLAN tag — under normal circumstances this Layer 2 information is only of local significance to the single IP subnet/VLAN that is directly adjacent to the input port.

internal fast data path is well beyond the scope of this paper. The key point is simply that the transfer of responsibility for this packet to the output port triggers the associated AQM algorithm to execute one iteration of the *packet marking algorithm*.

As we discussed in section 1, the marking algorithm looks at the current state of the output buffer at each packet arrival event and decides whether to: (i) simply *accept* this packet and append it to the appropriate output queue, (ii) *discard* this packet, or some other randomly-chosen "victim" packet already in the queue (either because there is no free space available in the queue or simply to serve as a rather harsh congestion notification signal), or (iii) *mark* the arriving packet, or some other randomly-chosen "victim" packet already in the queue, causing the source for that *flow ID* to receive a (more gentle form of) congestion notification signal and hence to respond by reducing its transmission rate. Thus, after the marking algorithm completes its iteration, the packet simply waits in the assigned queue until it is either transmitted over the output link or subsequently chosen to be the "victim" packet during another iteration of the marking algorithm.

4.1 Route Caching is Not Practical for Internet Core Routers

Packet classification is a relatively complex operation, which requires different fields within the packet header to be evaluated according to multiple sets of rules. Some packet classifier implementations use *route caching* to reduce the workload associated with packet arrival event. In this case, the full classification algorithm is only executed once for each flow — or at least once per flow between routing updates. Thereafter results produced by executing the packet classification rules (i.e., output port number, priority class, access rights, or other policy decision) associated with a set of recently-seen destination IP addresses and/or *flow IDs* are saved in a cache. Thus, subsequent packets belonging to the same flow can be quickly classified, using a simple table lookup, without having to execute the complete packet classifier algorithm again. The existence of a route cache would also trivialize our reverse flow matching problem, since we could simply add a boolean "spoofing flag" to the existing set of packet classifier outputs that form the route cache data for this *flow ID*.

Route caching represents a tradeoff between reducing the cost of the packet classifier (since it doesn't need to execute at wire speed), versus the additional cost of having to store information about a large number of individual flows in fast memory. For routers designed to serve the users within a single building (i.e., an "edge router"), or even a single organization (i.e., an "enterprise router"), the size requirements for an effective route cache are easily met with current hardware. [5]

[5] For example the YAGO Systems/Cabletron/Enterasys SSR series routers, which have been part of authors' network since 1998, use a chassis-based architecture with separate "Layer 3/4" route caches (i.e., packet classification based on individual *flow IDs*) integrated

However, the situation for an Internet core router may be quite different because the number of individual flows being multiplexed over a single link must surely be an increasing function of the physical *distance spanned* by that link (making it a more attractive "short cut" along the paths between a greater number individual source-destination pairs) and its *data rate* (making it possible to support a greater number of individual source-destination flows before it reaches saturation). Thus, a single cross-country link between two core routers in the Internet backbone will surely carry a much greater number of simultaneous flows than any link connected to a typical edge or enterprise router. But how large is large, and will the number of simultaneous flows be so large that a route cache would be too large/slow/expensive to be practical?

We looked at several sources of Internet measurements to learn more about the kind of flow patterns we should expect to find on current Internet backbone links. One excellent source of detailed Internet traffic statistics is the traffic archive maintained by the *WIDE* Project [24], through its *Measurement and Analysis on the WIDE Internet* (MAWI) Working Group [25]. Their monitored links carry a mixture of general Internet traffic across the Pacific, so their workload should not be biased towards any particular class of traffic. In addition, their traffic archive provides easy access to a detailed statistical summary of the traffic characteristics for large numbers of trace files collected over several years. We found that a typical 15-minute trace file generated by their monitoring point on a 100 Mbps trans-Pacific link (which has an average utilization of approximately 15-20%) only contains about 250,000 unique "flows". [6] The flow sizes are highly skewed, such that the top 10 individual flows account for almost 20% of the total bytes even though the average number of packets per flow is consistently between 15 and 20 packets. Thus, since almost all the flows are so short lived, we expect the number of simultaneously active flows to decrease in direct proportion to the length of the sampling period while we reduce the measurement time by a factor of 100. Such a flow pattern seems easily within the flow caching capabilities of the author's existing enterprise router.

into each port module. Each single 16-port Fast Ethernet module has an internal route cache with a capacity to store 256,000 individual *flow ID* entries [23], and a fully-expanded 16-slot chassis has an aggregate route cache capacity to store 4 million individual *flow ID* entries, and to support wire-speed layer 2 switching and/or *flow ID*-based Layer 3/4 routing up to a maximum of 48 million packets/sec.

[6] They define a "flow" to be a unique IP source/destination address pair, without distinguishing between port numbers. Had they followed our definition for a unique *flow ID*, which also uses the source and destination port numbers to distinguish between flows, then there might have been a small increase in the total number of flows, together with an equivalent decrease in the number of packets sent per flow. However, since the average number of packets per flow is already very low, even without distinguishing on the basis of port numbers, we do not expect this discrepancy to change the statistical properties of the flow patterns significantly.

Fomenkov et al. [26] have recently analyzed the flow patterns in Internet traffic, as one facet of a longitudinal study about long term trends in the evolution of Internet traffic. Their analysis is based on a series of traffic measurements, obtained from 20 high performance sites,[7] between 1998 and 2001. According to their results, a typical 90-second trace file rarely contained more than about 10,000 distinct flows, which is surprisingly consistent with the MAWI data considering the differences between the workloads in these two environments. In addition, they found that the number of active flows increased very slowly as a function of link speed. Indeed, they concluded that maintaining per-flow state in routers seems to be coming easier over time:

> "While the packet rate scales almost linearly with the bit rate, the counts of flows and IP pairs grow considerably slower than the bit rate. This observation indicates a potential possibility of storing these parameters as part of a router's state: the memory necessary for storage should grow slower than the CPU power required to process traversing packets." [26]

Unfortunately, these measured flow patterns at best represent some approximation to the "average case" workload for the packet classifier in an Internet core router. The router must also be capable of surviving the "worst case" workload it is likely to face in the real world. This is particularly important for our work, since the whole reason for adding sophisticated active queue management policies to the router in the first place was to maintain QoS support for the high priority services *despite* excessive service demands from low priority services.

4.2 Modeling Worst-Case Traffic as Aggregate Flows

Networking researchers have recently come to realize that choosing the "worst case" traffic workload is a lot more complicated than just increasing the arrival rate beyond the capacity of the link. In other words, if the excessive traffic is generated by one mis-behaving source, we can solve the problem in the packet classifier, either by passing the mis-behaving traffic through a separate rate-limiter, or by applying a specific access control rule to the input port which blocks that flow. Conversely, if the link is simply incapable of supporting its normal workload, and there is no way to bypass the link through simple routing changes, then nothing can solve the problem short of redesigning the network.

Thus, we must consider a new type of "worst-case" traffic pattern, called an *aggre-*

[7] Note that "high performance site" in this context refers to an organization that hosts supercomputers and/or giant data repositories used for academic research and enjoys a direct high bandwidth connection to Internet 2. Such organizations experience a very different traffic mix than a major commercial Internet service used by the general public, such as eBay, the CNN home page, online gaming systems, etc.

gate flow, which can suddenly overwhelm all normal traffic flows on a particular router link [27]. A high-volume aggregate flow is characterized by a *large* number of *coordinated* low-volume flows that: (a) occur simultaneously, and (b) originate from distinct sources but share a common destination. For example, a flooding-style distributed denial of service (DDoS) attack against a particular network-accessible service (such as a particular web server, the Internet's root name server, etc) would create aggregate flows in router links adjacent to its target.

Clearly DDoS attacks represent yet another variation on the mis-behaving traffic source problem, and everyone would be very happy if we could quickly find a way to distinguish the DDoS traffic from normal traffic so it can be controlled through the selective application of rate limiting and/or access control rules [27]. However, there is also a second type of flow aggregate, known as a *flash crowd*, which consists of a sudden spike of *legitimate* traffic, so it cannot simply be thrown away like a DDoS attack, and hence represents a very interesting model for the "worst-case" traffic pattern for a packet classifier with a routing cache. Flash crowds occur when a global *trigger event* causes large numbers of legitimate users to try to access the same network-accessible service simultaneously. The trigger event may have been planned well in advance — except for (vastly) underestimating the magnitude of the response it generates. For example, after the release of Independent Counsel Ken Starr's report on President Clinton to the public in September 1998, a CNN poll showed that an estimated 20 million people attempted to download the document from a government website (which normally handles 200,000 hits per month) within 48 hours of its release [28], and at the same time CNN's own website experienced a peak rate of 340,000 hits/minute [29]. Similarly, the 1999 Victoria's Secret Webcast of a live video broadcast event attracted 1.5 million hits [30].

Thus, to determine whether it is feasible to incorporate a route cache into the packet classifier for an Internet core router, we now present the following naïve performance model of the performance requirements for a single port to survive a flash crowd. If we assume that the layer 2 framing is based on Ethernet-like packet sizes, then we can approximate the size of each "large" data packet as 10,000 bits (since a 1,500 byte maximum length Ethernet packet corresponds to 12,000 bits excluding framing overhead), and the size of each "small" ACK packet as 500 bits (since a 64 byte minimum length Ethernet packet corresponds to 512 bits excluding framing overhead). In this case, the capacity of a single link would be sufficient to carry approximately 1 million "large" data packets per second at 10 Gbps, or approximately 4 million data packets per second at OC-768. If we further assume that the minimum bandwidth requirement to support a single participant in the flash crowd is to provide him/her with a data rate of about 10 Kbps (i.e., an average of one "large" data packet per second — equivalent to a fairly poor quality dialup modem connection), then a single router link can support approximately 1–4 million members of the flash crowd at the same time.

Meanwhile, the reverse link will be carrying the ACK traffic associated with each of those simultaneously active flows, together with an assortment of TCP control packets as other members of the flash crowd attempt to establish new connections to the target service, the service tries to limit the load by sending RESETs, and so on. Thus, in the worst case the reverse link may become heavily loaded with "small" control packets, which would require up to 20 million route cache lookups per second at 10 Gbps, or approximately 80 million route cache lookups per second at OC-768. Even worse, since the total size of the flash crowd could be much larger than the capacity of either the target service or this link,[8] every one of those "small" control packets could represent an attempt to establish a new flow that is not already stored in the route cache, no matter how large we make it. Based on these worse-case estimates, the performance requirements for the route cache are not very encouraging:

- The route cache must be fast enough to support tens of millions of lookup operations per second.
- Since the majority of the arriving packets represent doomed attempts to establish a new flow, the packet classifier must be fast enough to route every packet without any help from the route cache.
- To be effective, it must be large enough to hold millions of useful *flow ID* entries, representing the set of active flows, along with many more useless entries.
- A useful cache entry may only have one "hit" per sec.

Thus, if an Internet core router needs to be robust enough to survive the disruptive effects of a flash crowd, while continuing to offer suitable QoS levels to high priority applications, then the packet classifier should not rely on route caching to reduce its workload.

5 Comparison of Reverse Flow Matching and IP Routing

Despite the fundamental limitations of route caching, which work against its effectiveness in Internet core routers, we will now explain why reverse flow matching is a much simpler task than route caching. Consequently, we will show that these differences make reverse flow matching feasible in today's fastest Internet core routers. Thus, let us now focus our attention on two specific events in the packet forwarding process that form the key steps in the reverse flow matching algorithm at port i:

(1) The execution of one iteration of the AQM packet marking algorithm at port i, which is triggered by the arrival of an outgoing packet from some other router port, j say, across the internal interconnection fabric. Depending on its current

[8] Think of the Starr report, where the demand was so high that it took two days for the flash crowd to dissipate.

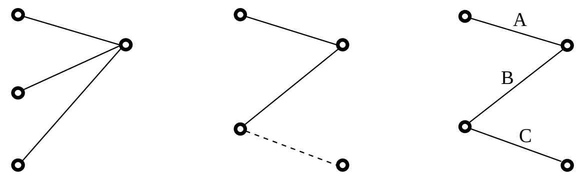
estimate of output buffer congestion at port i, the marking algorithm may decide to turn on the "spoofing flag" for one *flow ID* that currently has a packet waiting in that output buffer. Notice that this event happens *asynchronously* from (and can be handled in parallel to) any external packet arrival events experienced by port i. In addition, since the packet has just gone through the packet classification process at port j, we can assume that it carries with it all the relevant attributes from its route cache entry (if such a thing existed).

(2) The arrival of an incoming packet to port i through its interface to the external link. In addition to the normal steps in the packet classification process, we now add a simple reverse flow matching test, to see whether the incoming packets represents an ACK for any *flow ID* that currently has its "spoofing flag" turned on at this port. Since an ACK packet merely has a particular flag bit set in its TCP header, it is very easy to identify all the incoming ACK packets as part of the packet classification process. Determining whether an incoming ACK packet is also a target for ACK spoofing is equivalent to testing for an exact match between a single *flow ID* "key" (derived from the incoming ACK) and any member of target list of *flow IDs* (representing the set of flows that currently have their "spoofing flag" turned on).

Although the *flow ID* matching in step (2) make look remarkably similar to a ordinary route cache lookup, the effect is really quite different when you look more closely at the details. First, as we will see below, *lookup speed* is not a significant problem for such structures as long as we can control the *size* of the target list. This was not possible for route caching (at least, not under the "worst case" traffic conditions of a flash crowd), because the route cache loses its effectiveness as the cache miss rate increases. On the other hand, the target list for reverse flow matching is naturally restricted in size because the AQM algorithms normally select only a small fraction of the traffic passing through the output buffer to receive a congestion signal. Thus, the number of "spoofing flag" turn on events per second generated by the AQM algorithm that is executing at port i should be at least an order of magnitude smaller than the number of outgoing packets transmitted per second through its external interface. Second, even the implication of "failing to find a match" in the lookup table is completely reversed, since finding that the "spoofing flag" is turned off for this *flow ID* is a good thing because it allows us to do nothing, other than allowing this packet to follow the fast path through the router as usual. Third, even if the reverse flow matching does succeed, we still don't need to disturb the fast path through the router: the matched ACK packet can be redirected through a "detour", where an asynchronous process (perhaps even external to router) can use it for a template for creating a set of spoofing ACKs, before it returns to the normal data stream. Alternatively, if reverse flow matching is combined with ECN to short circuit the congestion signaling delay, then the "detour" would merely set the ECN flag and return the matched packet to the normal data stream.

Having now explained why reverse fbw matching is a different, and simpler problem than route caching, it remains to show how it can be implemented inexpensively. In particular, we believe that reverse fbw matching can be carried out using the same commodity hardware that is widely used to implement the transparent bridging algorithm that forms the basis for high performance Layer 2 Ethernet switches [31]. Recall that the basic operation of a Layer 2 switch consists of two things. First, the switch must build a *filtering database*, which contains every active 48-bit MAC address that is observable from the switch, by passively listening to the source addresses from every packet it can hear on the network. These source addresses are used to create and/or update its database of port numbers through which each of those MAC addresses can be reached. In parallel with its database maintenance, the switch also executes a simple *packet filtering* algorithm each time it receives another incoming packet, to determine whether or not it should discard that packet or relay it to one or more other port(s). Thus, each iteration of the packet filtering algorithm is nothing more than extracting the 48-bit destination address field from each incoming packet, and then attempting to use it as the "key" for finding an exact match among the list of other 48-bit addresses stored in the filtering database.

If we compare a single application of the Layer 2 packet filtering algorithm to a single application of the reverse fbw matching algorithm, the only major difference is that the "key" for reverse fbw matching is a 96-bit *flow ID* instead of a 48-bit MAC address! However, it is very easy to partition the single 96-bit lookup for solving the reverse fbw matching problem into two parallel lookups using a pair of disjoint 48-bit "keys", representing the source IP address and its associated port number as one "key" and the destination IP address and its associated port number as the other "key" respectively. In theory, by splitting the reverse fbw matching problem into two independent parts we have introduced the possibility of creating "false positives", where both 48-bit halves of the *flow ID* are included in the reverse fbw matching lookup, but they were not paired with each other.



(a) CASE 1 **(b) CASE 2** **(c) CASE 3**

Fig. 6. Three cases of flow overlapping

If the probability of "false positives" goes too large, then partitioning 96-bits matching into two 48-bits matchings would create some problems. Fortunately, after we analyzed several traces from the MAWI repository [25], we are convinced that such a partitioning will not cause any serious problems. Each MAWI trace file we have studied ran for 15 minutes some time in 2002 or 2003, and recorded about 2 million packets at point B. Typically, about 2/3 of recorded packets are TCP packets, and there are about 10,000 TCP flows identified from each trace. We drew a flow-matching graph for each trace, where we regarded each distinct (IP, port) pair as one vertex in the graph and drew one line to connect two pairs if both appear in one packet. In each graph, we found that about half of the flows overlap with other flows. Three patterns of flow overlapping are possible, as illustrated in fig.6.

In the first case, many clients connect to one server, which cannot cause any "false positives". The second case has multiple conflicting pairs forming a connected subgraph, but the lifetime of overlapping flows are disjoint. Thus, if we add time as a third dimension in the flow-matching graph, this case cannot cause any "false positives" either. Finally, in the third case, we consider overlapping flows in which the conflicting pairs are also overlapping in time. Although case 3 overlapping clearly has the potential to create "false positives", these situations occurred very rarely in the MAWI traces. Furthermore, the conditions for generating a "false positive" on flow B are very unlikely: both of its conflicting endpoints (represented by the two keys x and y, say) must be loaded in the cache at the same time because of other overlapping marked flows flows A and C say (which are represented the keys w and x, and the keys y and z, respectively). Moreover, even though a "false positive" will mean that ACK Spoofing sends a congestion signal to a different source than the one selected by the AQM algorithm, this mistaken identity does not cause a serious problem because the congestion signal can only be sent if the router is congested and the accidental target is actively using this link.

It is normally expected that a good Layer 2 switch can update its filtering database and/or carry out packet filtering in any combination at wire speed across all ports simultaneously, which is equivalent to a processing rate of 148,810 packets per second per 100 Mbps port or almost 1.5 million packets per second per Gigabit port. The key to achieving such high performance is to use special-purpose hardware to speed up the packet filtering algorithm. This hardware, known as a Content Addressable Memory (CAM) or associative memory, is a storage device that can be addressed through its own contents. Each bit of CAM storage comes equipped with its own comparison logic [32]. Thus, whenever you present some data as an input "key" to the CAM, its value is simultaneously compared with all the data currently stored within the CAM. If a match is found, then the address of the matching data is returned as a result. Of course, since that addresses and data are treated interchangably, in the event of a match the CAM gives us back another copy of our original input "key"; otherwise we get nothing. The ternary CAM is a generalization of this basic concept in which some parts of the input "key" can be encoded as "don't care" values, allowing the CAM to return new information that was not part of our original key, such as the port number, priority class, or other information associated with a given address.

Today, high performance CAMs that are optimized for Ethernet switching applications are widely available from various semiconductor vendors. For example, SwitchCore offers a high performance CAM 4 Mbit device [33], which can store up to 32,000 entries on a single chip (each up to 80 bits wide) or be chained together to form a 3-level hierarchy that can store up to 224,000 entries, while offering a sustained processing rate of 75 million lookups per second. It is interesting to note that this existing SwitchCore CAM can already be scaled to meet the demands of the reverse path matching algorithm under the naive worst-case traffic model for a 10 Gbps link that we described in section 4.1, i.e., up to 20 million lookups per second (assuming the link is saturated with "short" packets), and 100,000 entries (assuming 10% of the 1 million flows have been marked by the AQM algorithm).

6 Conclusions

ACK Spoofing, particularly in combination with signal cancellation, represents a very attractive means for congestion feedback signaling in IP routers. The basic ACK Spoofing algorithm offers significant performance advantages over other congestion signaling methods, such as packet dropping and ECN. It delivers congestion signals more quickly than other signaling methods because of its "short circuited' signaling path. It is compatible with the installed base of TCP implementations. And, it avoids the negative side effects caused by needlessly dropping packets, such as stalled connections because of timeouts and additional retransmissions. Moreover, once we adopt the basic ACK Spoofing algorithm, we can easily add several performance-enhancing features which add almost no extra complexity to the method, such as signal cancellation and latency reduction by applying "overbooking" to the packet marking process while limiting the signals to the quickest matches. We could even apply the all of the same methods to ECN signaling, simply by setting its ECN flag instead of using it as the template for generating a set of spoofing ACKS when we identify a suitable reverse ACK using reverse flow matching,

However, the practicality of ACK Spoofing, and its associated enhancements, depends critically on our ability to carry out the associated reverse flow matching problem quickly, across high volumes of network traffic, and using only a modest amount of additional hardware support. Our results show that reverse flow matching can be implemented at reasonable cost, using essentially the same hardware as the packet filtering logic commonly employed in Layer 2 transparent bridges. Moreover, it can accommodate worst-case traffic patterns, including flow aggregates such as flash crowds and distributed denial-of-service attacks, that would render ordinary route caching algorithms completely ineffective.

We also examined a variety of Internet trace files to obtain realistic estimates for the total size of the lookup table that would be required for reverse flow matching, the false hit probability if we implemented the lookup table as a pair of 48-bit lookups into a standard Ethernet CAM instead of a single 96-bit lookup into a purpose-built *flow ID* table, and the latency reduction from adopting the "short circuited" congestion signaling path.

Finally, we note that the list of *flow IDs* that currently have their spoofing flags turned on — which allows us to carry out reverse flow matching in support of ACK spoofing — is essentially equivalent to the identification data for the aggregate-based congestion control (ACC) algorithm proposed by Mahajan et al. [27], and that the same techniques we use to carry out reverse flow matching could also be used to handle the local ACC problem. In their algorithm, they first construct a profile for the *flow IDs* associated with a particular high-bandwidth flow aggregrate by using the packet marking algorithm to obtain a random sampling of the output queue. The individual samples are then combined to create a smaller set of more general rules by prefix matching. But replacing a pair of adjacent 24-bit IP address prefixes by a single 23-bit IP address prefix is equivalent to combining the two entries in a ternary CAM by setting bit 24 to the "don't care" state. Thus, by adding some additional bits to the result field, we can distinguish between a simple request for ACK spoofing versus a standing order to divert all traffic associated with the given flow aggregate into some sort of rate limiter. Hence, the only extra features we would need to add to our implementation of ACK Spoofing so that it can also handle local ACC problem would be certain "higher level" policy decisions that occur outside the main packet forwarding path, such as determining whether the router is currently under attack by some flow aggregate, and building an appropriate set of aggregate signatures through prefix matching (i.e., [27] section 3.1).

References

[1] B. Braden, et al., Recommendations on queue management and congestion avoidance in the Internet, http://www.ietf.org/rfc/rfc2309 (April 1998).

[2] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1 (4) (1993) 397–413.

[3] W.-C. Feng, D. D. Kandlur, D. Saha, K. G. Shin, BLUE: A new class of active queue management algorithms, Technical Report, CSE-TR-387-99, University of Michigan.

[4] S. Athuraliya, S. H. Low, Optimization flow control, II: Random exponential marking, preprint, http://netlab.caltech.edu, May 2000.

[5] T. J. Ott, T. V. Lakshman, L. Wong, SRED: Stablized RED, in: Proc. INFOCOM '99, New York, 1999, pp. 1346–1355.

[6] D. Lin, R. Morris, Dynamics of random early detection, in: Proc. SIGCOMM '97, Nice, France, 1997, pp. 127–137.

[7] J. S. Ahn, P. Danzig, Z. Liu, L. Yan, Evaluation of TCP Vegas: Emulation and experiment, in: Proc. SIGCOMM ’95, 1995, pp. 185–205.

[8] S. Floyd, TCP and explicit congestion notification, ACM Computer Communication Review 24 (5) (1994) 10–23.

[9] K. Ramakrishnan, S. Floyd, D. Black, The addition of explicit congestion notification (ECN) to IP, RFC 3168.

[10] D. Wischik, How to mark fairly, in: Workshop on Internet Service Quality Economics, MIT, 1999.

[11] S. Kunniyur, R. Srikant, A time scale decomposition approach to adaptive ECN marking, in: Proc. INFOCOM ’01, Anchorage, Alaska, 2001, pp. 1330–1339.

[12] I. Yeom, A. Reddy, Marking for QoS improvement, Computer Communications 24 (1) (2001) 35–50.

[13] W.-C. Feng, D. D. Kandlur, D. Saha, K. G. Shin, A self-configuring RED gateway, in: Proc. INFOCOM ’99, New York, 1999, pp. 1320–1328.

[14] S. Athuraliya, V. H. Li, S. H. Low, Q. Yin, REM: Active queue management, IEEE Network 15 (3) (2001) 48–53.

[15] Z. Xu, M. Molle, TCP congestion control via ack spoofing, Technical Report, UCR Computer Science Dept.

[16] Z. Xu, M. Molle, Red with ack spoofing, in: Proc. Allerton Conference on Communication, Control, and Computing, 2003, pp. 120–129.

[17] Mesquite Software, CSIM18 documentation: User guides, http://www.mesquite.com/htmls/guides.htm (2001).

[18] G. R. Wright, W. R. Stevens, TCP/IP Illustrated, Volume 2: The Implementation, Addison-Wesley, 1995.

[19] T. Karagian, (private communication), UCR Department of Computer Science and Engineering, 2003.

[20] W. R. Stevens, TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994.

[21] V. Paxson, S. Floyd, Wide-area traffic: The failure of poisson modeling, IEEE/ACM Transactions on Networking 3 (3) (1995) 226–244.

[22] D. E. Comer, Network Systems Design using Network Processors, Peason Prentice Hall, 2004.

[23] Enterasys Networks, Ssr-htx32-16 fast ethernet t-series module data sheet, http://www.enterasys.com/products/routing/SSR-HTX32-16/ (2001).

[24] K. M. K. Cho, A. Kato, Traffic data repository at the WIDE project, in: Proc. Freenix ’00, San Diego CA, Usenix, 2000, pp. 263–270.

[25] MAWI working group traffic archive, http://tracer.csl.sony.co.jp/mawi/.

[26] D. M. M. Fomenkov, K. Keys, k claffy, Longitudinal study of Internet traffic from 1998-2001: a view from 20 high performance sites, http://www.caida.org/outreach/papers/2003/nlanr/index.xml (April 2003).

[27] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, S. Shenker, Controlling high bandwidth aggregates in the network, ACM Computer Communications Review 32 (3) (2002) 62–73.

[28] CNN.com, 20 million Americans see Starr's report on Internet, http:www.cnn.com/TECH/computing/9809/13/internet.starr/ (September 13 1998).

[29] CNN.com, Starr report causes Internet slowdown, but no meltdown, http://www.cnn.com/TECH/computing/9809/11/internet.congestion/ (September 11 1998).

[30] F. Douglis, M. F. Kaashoek, Scalable Internet services, IEEE Internet Computing 5 (4) (2001) 36–37.

[31] IEEE Computer Society, Media Access Control (MAC) Bridges, Vol. ANSI/IEEE Std 802.1D, IEEE, 1998.

[32] M. Defossez, Content addressable memory (CAM) in ATM applications, http://www.xilinx.com/xapp/xapp202.pdf (January 2001).

[33] SwitchCore AB, CXE-5000 32k entries multi-protocol content addressable memory, http://www.switchcore.com/products/cxe-5000/ (2003).