Robust Congestion Signaling

David Ely[†], Neil Spring[†], David Wetherall[†], Stefan Savage[‡], and Tom Anderson[†]

[†]Dept of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350

[‡]Dept of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

Abstract

We present an improved Explicit Congestion Notification (ECN) mechanism that enables a router to signal congestion to the sender without trusting the receiver or other network devices along the signaling path. Without our mechanism, ECN-based transports can be manipulated to undermine congestion control. Web clients seeking faster downloads, for example, can trivially conceal congestion signals from Web servers. A misbehaving connection would exceed its fair bandwidth share at the expense of competing traffic by as much as an order of magnitude in our simulations. Our improved mechanism is robust *because it does not depend on correct implementation at locations other than the sender and marking router, and it is* practical *because it admits an efficient implementation that is backwards-compatible with prior ECN and TCP/IP mechanisms.*

1. Introduction

Explicit Congestion Notification (ECN) [21], with active queue management in the form of RED gateways [10], has been proposed as a standard mechanism to improve congestion control in the Internet. With ECN, routers are able to mark packets to signal incipient congestion, as well as simply drop them during congestion. This avoids loss and improves performance [24].

Unfortunately, existing transport protocols that use ECN are more vulnerable to faults than when congestion is signaled only by packet drops. ECN-based transports depend on receivers and network devices to return congestion signals to senders, opening the door to misbehaviors that conceal congestion indications and cause well-implemented senders to transmit too aggressively. In contrast, packet drops cannot be concealed without sacrificing reliable data transfer.

Misbehavior may be the result of a deliberate attempt to exploit ECN. There is an opportunity for Web clients to reduce their download times, regardless of the impact on others, by concealing congestion signals from servers. A deceitful client is *trivial* to implement, requiring only a single line change to the TCP source code. Our simulations show that such deceit is both *effective*, gaining up to ten times their fair share of bandwidth, and *harmful*, slowing competing flows to as little as one tenth their fair share.

Misbehavior may also be the accidental result of incompatible designs or buggy implementations. For example, ECN is not recommended for use with older IPSEC tunnels because congestion signals may be inadvertently lost at the tunnel endpoint when one layer of packet headers is removed [22]. Deployment of ECN depends on the reliable delivery of congestion signals, and even unintentional loss of congestion signals results in an unfair distribution of bandwidth.

One approach to the problem of misbehavior is to rely on network support such as Fair Queuing [3] to enforce fair bandwidth allocations. This addresses intentionally aggressive senders as well as misbehaviors that indirectly affect the sending rate. However, there are drawbacks to depending solely on network enforcement. While ECN is poised to enter mainstream deployment, efficient bandwidth allocation mechanisms are still the subject of research [4, 15, 29]. Even if fair bandwidth allocation is deployed, robust congestion control is still needed because it allows senders to make efficient use of allocated bandwidth. Without congestion control, a flow cannot determine the current allocated bandwidth, which changes dynamically with the number of active flows.

We focus on what can be accomplished with an end-to-end approach that requires no network support beyond ECN packet marking. We make three contributions in this paper. First, we demonstrate that incorrect ECN behavior can cause an unfair disparity in bandwidth allocation. Second, we present the design of a congestion signaling mechanism that enables the sender to detect misbehavior without the cooperation of any network device other than the marking

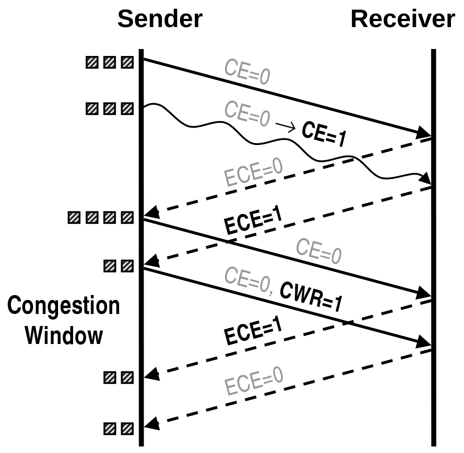

Figure 1. ECN avoids dropped packets by allowing routers to signal congestion by setting the Congestion Experienced (CE) bit in the IP header. In TCP, the receiver returns congestion signals by setting the ECN-Echo (ECE) flag. The sender sets the Congestion Window Reduced (CWR) flag to inform the receiver that it has reacted to congestion.

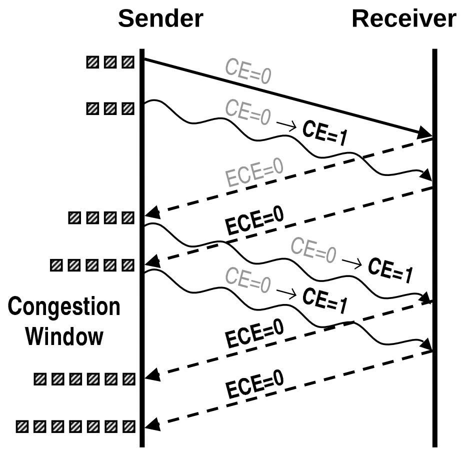

Figure 2. By hiding congestion signals, a misbehaving TCP receiver can cajole the sender into increasing the congestion window. Because the data packets are ECN-capable, they will not be dropped by the router until the link becomes congested.

router. Third, we show how to implement our design for deployment in TCP. We base our design on TCP, because it is the only mainstream transport protocol for which ECN is currently defined. However, our design can readily be applied to other transports, such as SCTP [28] and TFRC [9].

Our design includes a random nonce in the IP header of each packet, which is erased by routers to signal congestion. The transport receiver returns nonce information in acknowledgements to the transport sender to demonstrate that received packets did not experience congestion. Our design can be implemented efficiently because even a *single* bit of nonce information is sufficient to detect misbehavior. To carry this nonce, our design uses a newly allocated codepoint in the IP header [22] and one bit in the TCP header.

The remainder of this paper is organized as follows. In the next section, we describe how ECN can be undermined by misbehavior. In Section 3, we present the design of a robust congestion signaling protocol that does not depend on trust. This is followed by implementation details in Section 4, showing how our protocol can be combined with TCP/IP. In Section 5, we present an evaluation and simulation results that show our protocol is effective at preventing misbehavior. We present related work in Section 6 before we conclude in Section 7.

2. Motivation

Signaling congestion via packet drops has proven to be a simple and robust mechanism. It demands little of congested routers, and once a packet is dropped, subsequent routers cannot interfere with the congestion signal. To implement reliable transfer, receivers must correctly acknowledge lost packets, thereby returning the congestion signal to the sender.

Explicit Congestion Notification (ECN) [21] changes the character of congestion signaling to improve performance. It allows routers to signal congestion to end hosts explicitly, rather than implicitly via packet drops. Routers mark packets along congested links, and the receiver returns these congestion marks to the sender in a transport-specific manner. The basic ECN protocol for use with TCP is shown in Figure 1. To signal congestion, routers set the Congestion Experienced (CE) state in the IP header of ECN-capable packets. The receiver returns this signal to the sender by setting the ECN-Echo (ECE) flag in the TCP header of subsequent acknowledgements. To ensure reliable delivery of this signal, the receiver continues to set the ECE flag in acknowledgements until a Congestion Window Reduced (CWR) flag is received, implying the sender has reacted to the congestion.

Unfortunately, the design of ECN requires routers and receivers to explicitly and correctly participate in the congestion control loop, but has no means to check or enforce this cooperation. As noted in the ECN specification [22], this raises the possibility of misbehavior. The following behaviors are dangerous because they subvert congestion control:

- A receiver may receive marked packets but neglect to inform the sender, as shown for TCP in Figure 2.

- A router may clear congestion signals received from upstream.
- A router on the reverse path may clear the congestion echo signals being returned to the sender.

Misconfigured or incompatible devices, such as proxies, firewalls, tunnels, and NATs, could exhibit these behaviors. They are not hypothetical. ECN deployment has revealed incompatibilities with some IPsec tunnels [22] and interference in ECN negotiation by some firewalls [6, 11].

Intentional misbehavior by receivers would allow Web clients to improve their download performance at the expense of competing transfers as shown in Section 2.2. Unscrupulous ISPs could even provide an accelerator "service" that unmarks packets to improve performance for their customers. The ECN specification argues that network and receiver misbehavior is no worse than the problem of aggressive senders [22]. While this is true, receivers and senders often have conflicting incentives with respect to congestion control: Web servers aim to share bandwidth between all clients, while Web clients aim to improve their download times. We argue that receiver misbehavior is an important case in its own right.

These misbehaviors are not possible when packet drops are the only signal of congestion because receivers cannot conceal packet loss without sacrificing reliability, and network agents cannot "undrop" a packet as they can unmark ECN congestion signals. The intent of our work is to ease the deployment of ECN by making it as robust as packet drops to such misbehaviors, ensuring that ECN can only be used as intended and not in a way that undermines congestion control.

2.1. Simulation Setup

To test the impact of abusing the ECN protocol, we simulate, using *ns* [19], a misbehaving receiver that ignores congestion signals by never setting the ECN-Echo flag. Our simulated network extends the ECN evaluation network included in *ns* to incorporate faster links and recent experience in tuning RED. We do not expect that simulations on small networks will accurately predict a misbehaving receiver's impact in the heterogeneous Internet, but they do highlight its potential.

Our network topology includes a 10 Mbit/s bottleneck link as shown in Figure 3. We configure the RED queue using the best practices described in [5, 7]: *minth* = 25 packets, *maxth* = 75 packets, the maximum instantaneous queue size is 150 packets, the packet size is 1,000 bytes, and the recent "gentle" option is enabled. Decreasing the queue capacity decreases the performance imbalance caused by misbehavior, but also reduces link utilization.

To reduce harmful synchronization effects, flows begin independently, the bottleneck carries reverse traffic, and the

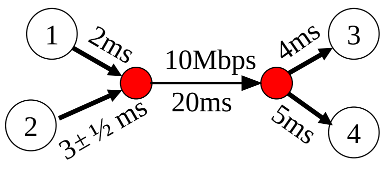

Figure 3. The simulated network topology is shown. Arrows represent the direction of traffic flow. Links other than the bottleneck are 100Mbit/s. The link to Node 2 has a one way latency randomly chosen uniformly between 2.5 and 3.5ms. Several traffic sources or sinks were placed on each node to vary congestion at the bottleneck.

results presented are averages from different simulations using different random number seeds. The seed of the random number generator affects the behavior of the RED gateway, the start time of connections, and the latency to node 2. All flows support selective acknowledgements [17] and are not limited by flow control. These attributes represent modern TCP implementations. [13, 26]

2.2. Effects of a Misbehaving ECN Receiver

In this section, we only address the effects of misbehavior on long flows. Figure 4 shows the performance of the ECN misbehaver competing with ECN-enabled flows. We show the bandwidth obtained by all flows relative to their "fair share" of the bottleneck as the number of competing flows varies. The "fair share" is easy to interpret, but underestimates the impact of misbehavior because not only does the misbehaver receive up to six times its fair share, the behavers receive as little as one tenth their fair share. The dotted line at $y = 1$ represents an equal distribution of the capacity of the link.

Figure 4 shows that misbehavior gains a significant performance advantage over compliant flows, does not harm its own performance in the absence of contention, and greatly reduces the bandwidth available to compliant flows. When the misbehaver competes with ECN-enabled TCP connections, it forces the sender's congestion window to increase until the router drops packets. When the number of flows is sufficient to saturate the router, it no longer marks packets to alleviate congestion and instead drops packets from *all* flows in proportion to their queue consumption, decreasing the misbehaver's effectiveness. This explains the decrease in the misbehaver's share of the bandwidth when the link has more than twenty connections.

The performance advantages for a misbehaving receiver are even more significant when it contends with conformant flows that are not ECN-enabled. In Figure 5, the misbehaver takes up to ten times its fair share. While the router drops

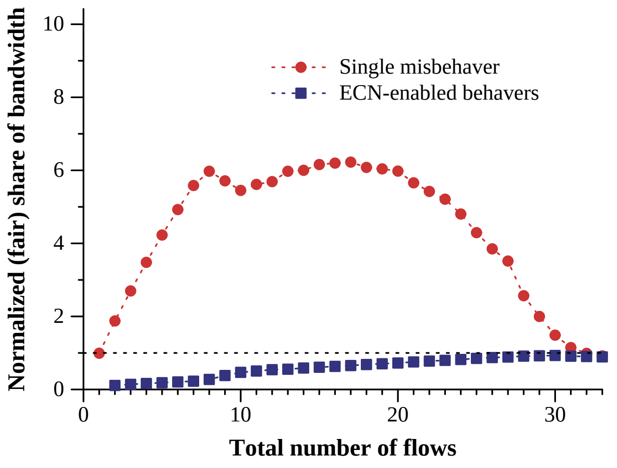

Figure 4. The ECN misbehaver contends with behaving ECN flows through a 10Mbit/s bottleneck. The misbehaver (upper line) is able to achieve significant performance gains at the expense of conforming ECN flows (lower line). As the number of flows increases, all flows suffer losses, and the misbehaver loses its advantage.

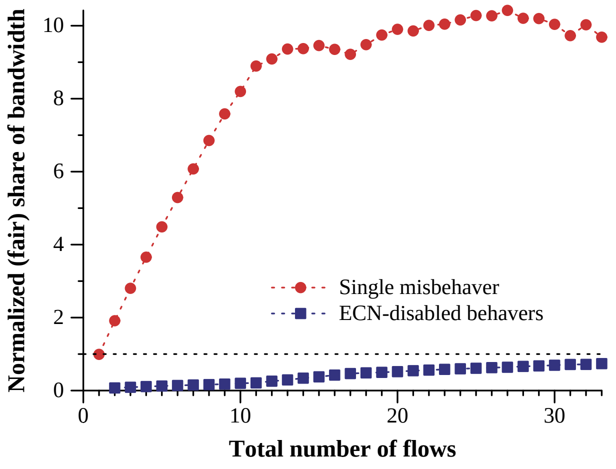

Figure 5. The ECN misbehaver contends with non-ECN flows through a 10Mbit/s bottleneck. The misbehaver's advantage (upper line) is even more pronounced, as the router drops packets from the non-ECN flows (lower line). The misbehaver suffers fewer drops than non-ECN flows, who often have to recover from those losses by timeout.

packets from other flows, it incorrectly assumes the misbehaver will respond to the explicit signals and back off. This gives the misbehaver even more space in the queue to gain additional bandwidth. In this graph, the misbehaver's performance does not decrease as the number of flows increases above twenty because many flows suffer timeouts to recover from losses, yielding their bandwidth.

3. A Robust ECN Design

When a router drops a packet to signal congestion, this signal is permanent: a downstream router cannot "undrop" a packet. By enabling the ECN sender to detect when marked packets are unmarked, we make ECN as robust a congestion signal as packet drops. Unlike earlier designs, our design uses multiple unmarked packet states, so that the original unmarked state cannot be recovered from a marked packet. To prove the absence of congestion, the receiver returns the original unmarked states in acknowledgements. We refer to the original unmarked state as a *nonce*, because this exchange of random numbers is analogous to the use of nonces in security protocols.

Our solution, ECN with nonces, has the following properties:

- It does not remove any of the benefits of ECN for behaving participants.
- It detects misbehaviors with a high probability.
- It never falsely implicates behaving receivers and networks.
- It is efficient in terms of packet overhead, per-packet processing, and state required at the end points.

Consider an implementation that uses *billions* of unmarked states, i.e. nonces, which are cleared by routers to signal congestion, and furthermore, the receiver is required to return the correct nonce for *every* packet. A sender can detect when packet marks are concealed, but this solution is inefficient: header bits are sparse, and most transport protocols do not send per-packet acknowledgments, nor deliver them reliably. We address both of these shortcomings. We first show that as few as two nonce values can detect misbehavior. Then we demonstrate that just as transport protocols use *cumulative* acknowledgements, cumulative nonces eliminate the need for per-packet reliable acknowledgements.

3.1. One-bit Nonces

Large nonces of 16 or 32 bits would be effective at identifying concealed ECN congestion signals. However, the packet's nonce must be stored in protocol headers, which have few available bits. An insight enabling a less expensive implementation is that congestion control applies to a sequence of packets. Even a *one-bit* random nonce per packet is enough to detect misbehavior in the congestion control loop, since each mark of congestion is a separate trial. Al-

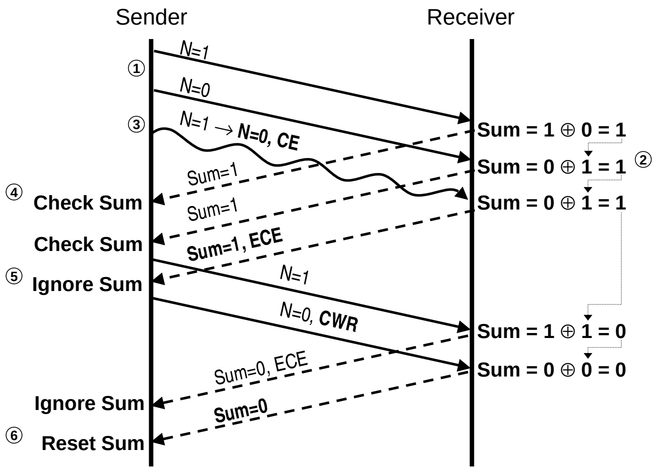

Figure 6. This diagram shows the ECN-nonce protocol as applied to TCP and one-bit nonces. The sender attaches a random nonce (N=0 or N=1) to each packet ①. The receiver always returns the one-bit cumulative nonce, which is the one-bit sum (i.e. parity) of these nonces ②. When a router marks the congestion experienced bit (CE), the nonce is cleared ③. The sender can verify that no congestion signals are concealed by checking that the sum reported by the receiver is correct ④. However when congestion is experienced, the receiver's nonce sum will be incorrect and is ignored ⑤, and the sender must resynchronize with the receiver's sum by resetting its own sum to the receiver's sum ⑥.

though a misbehaving node might occasionally avoid detection, this is acceptable, as long as the sender responds to misbehavior sufficiently when detected.

3.2. Cumulative Nonce Protocol

Cumulative nonces allow the receiver to prove receipt of unmarked packets without returning every original nonce. The sender places a random nonce in each packet, which is cleared by a router to signal congestion. The receiver maintains a cumulative nonce, which is the sum of the nonces received for all in-order packets, and includes it in every acknowledgement to be verified by the sender. Because every nonce is needed to calculate the correct cumulative nonce, it depends on the receipt of only unmarked packets. Figure 6 gives an example of this protocol, as adapted for TCP and one-bit nonces.

Resynchronization of the sender and receiver cumulative nonces is needed after congestion events as shown in Figure 6. When packets are marked, the nonce is cleared, and the cumulative nonce at the receiver will no longer match the sender's. Once a nonce has been lost, the difference between sender and receiver cumulative nonces will not change until there is further loss. After reacting to the congestion event, the sender resynchronizes with the receiver by adopting its view of the cumulative nonce. Resynchronization depends on the details of the transport protocol, but is not required to happen immediately because congestion indications are not conveyed more frequently than once per round trip [22]. In the case of TCP, we suspend checking of the cumulative nonce while the Congestion Window Reduced (CWR) signal is delivered to the receiver. We reset the sender's cumulative nonce to the receiver's when the packet containing CWR is acknowledged. This scheme is simple and has the benefit that the receiver is not explicitly involved in the resynchronization process. The receiver could delay the resynchronization process in two ways: 1) never acknowledge the packet containing CWR or 2) return ECN-Echo (ECE) in the acknowledgment of the CWR packet. However, both of these actions are interpreted as new congestion signals, which will discourage the receiver from interfering with resynchronization.

3.3. Detected Misbehaviors

ECN with nonces protects ECN from various abuses and incompatibilities. ECN-nonce senders are able to detect the dangerous misbehaviors listed in Section 2 and the potential misbehavior of network devices removing ECN capability from packets.

The ECN-nonce can also be used to protect other congestion related protocols from misbehavior. For example, Eifel [16] is a recently proposed mechanism for improving TCP performance by disambiguating retransmissions. When an Eifel receiver claims to have received the original transmission of a retransmitted packet, the sender "undoes" the congestion action to maintain its sending rate. However, a misbehaving receiver might claim all acknowledgements are of original transmissions, subverting congestion control. Because retransmissions are ECN-incapable [22], the correct cumulative nonce is evidence of having received original transmissions.

The ECN-nonce also prevents the optimistic acknowledgement vulnerability that we identified previously [25]. By optimistically acknowledging packets, a receiver can simulate a much smaller round-trip time and gain an unfair share of the bandwidth. ECN with nonces prevents this because placing the correct cumulative nonce in an acknowledgment depends on the receipt of each packet.

The ECN-nonce provides a mechanism to detect misbehavior, but leaves unspecified the sender-specific policy to address it. Different policies are possible: for example, a sender could send more slowly or no longer send ECN-capable packets. Whatever the policy, the incentive for violating the congestion control specification should be eliminated so that misbehavior is not effective.

3.4. Extensions for Other Transports

The Stream Control Transmission Protocol (SCTP) [28] is a new, reliable transport protocol being developed by the IETF. It includes modern features such as multi-homing, framing, and multiple concurrent streams per connection. SCTP uses selective acknowledgements and supports ECN. The cumulative nonce can be used in SCTP because like TCP, it uses cumulative acknowledgments. The cumulative nonce can be carried in a new SCTP option, known as a chunk.

The ECN-nonce can also be applied to unreliable transports by taking advantage of the transport specific acknowledgment mechanism. The Rate Adaptation Protocol (RAP) [23] is a TCP-friendly, unreliable transport protocol designed for unicast transfer of realtime streams. RAP receivers acknowledge consecutive ranges of packets. Augmenting RAP with the ECN-nonce consists of returning the cumulative nonce for the range of consecutive packets in each acknowledgement. TCP-Friendly Rate Control (TFRC) [9] is an unreliable transport protocol that uses a model of TCP performance to calculate a smooth sending rate based on the loss event rate and round trip time. TFRC receivers calculate the loss event rate from a weighted average of the length of recent loss intervals. TFRC can be adapted to use the ECN-nonce by requiring the receiver to provide the cumulative nonce for each interval between loss events.

4. Implementation

In this section, we show that ECN with nonces can be implemented efficiently in terms of both protocol header space and packet processing overheads. Specifically, we describe the high-level features of our Linux implementation of ECN-nonces.

4.1. Protocol in Depth

ECN with nonces adds four new aspects to the ECN protocol: 1) generating and sending the nonces, 2) clearing the nonces when congestion is experienced, 3) computing and returning the cumulative nonce, and 4) checking the cumulative nonce in acknowledgements.

Generating and Sending Nonces. The sender must generate random nonces that cannot be guessed by the receiver or another network agent. This is a lightweight operation because generating a single 32-bit pseudo-random integer, which requires fewer than twenty instructions, yields enough one-bit nonces for 32 packets. The sender transmits a one-bit nonce with each ECN-capable packet. For each unacknowledged packet, the sender stores the cumulative nonce expected in its acknowledgement.

The ECN-nonce is carried in the IP header so that it can be cleanly manipulated by routers. The ECN Proposed Standard [22] includes our encoding of the one-bit nonce, but does not specify its use. Two code-points, ECT(0) and ECT(1), are allocated for the ECN-Capable Transport flag as described in Table 1. In each ECN-capable packet, a transport sender can send either ECT(0) or ECT(1), allowing it to encode a nonce of either 0 or 1.

	RFC2481 Meaning	Proposed Meaning [22]
00	ECN-incapable	ECN-incapable
10	No congestion	No congestion, Nonce=0
01	*undefined*	No congestion, Nonce=1
11	Congestion	Congestion, Nonce cleared

Table 1. The Experimental Standard ECN (RFC2481) encoding is compared with the Proposed Standard ECN encoding, which supports the ECN-nonce. The ECN-Capable Transport (ECT) codepoints ’10’ and ’01’ are called ECT(0) and ECT(1) respectively.

Clearing Nonces. Routers identify ECN capable packets and mark them if indicated by active queue management. To mark packets, routers change either of the unmarked states to the single marked state. This process automatically erases the original nonce carried with the packet, which is key to our scheme. Neither the receiver nor any other device can clear the packet’s congestion mark without guessing the value of the original nonce.

Computing and Returning Cumulative Nonces. In addition to distinguishing marked packets and setting the ECN-Echo flag as before, receivers maintain a cumulative nonce as packets arrive and return the appropriate cumulative nonce with each acknowledgement. The receiver stores the nonce value for each out-of-order packet and a single cumulative nonce for all in-order packets. A cumulative nonce is computed by taking the exclusive-or of the previous cumulative nonce and the most recent nonce, as shown in Figure 6. In the case of marked packets, the nonce values will be unknown to the receiver. The missing nonce values are ignored when calculating the cumulative nonce, and ECN-Echo is set to signal congestion to the sender. The cumulative nonce is returned to the sender in a TCP header bit that is currently unallocated. This is the only additional bit that ECN with nonces requires in either the TCP or IP header.

Checking Nonces and Synchronization. Checking the cumulative nonce is straightforward and is performed every time an acknowledgement is received, except during congestion recovery. Minimal state is required at the sender to check cumulative nonces, as the sender maintains only a one-bit cumulative nonce for each unacknowledged packet. Checking consists of comparing the cumulative nonce in each acknowledgement to the correct value.

If ECN-Echo is set, the receiver is sending a congestion signal, and the sender ignores the cumulative nonce. As specified by ECN, the sender responds to congestion in the standard manner by halving the congestion window, and then sends CWR in the next *new* packet to inform the receiver that it has reacted to congestion. The receiver clears the ECN-Echo flag once the CWR signal is received.

During this recovery process, the cumulative nonce may be incorrect because one or more nonces were cleared. This does not matter, because TCP invokes congestion mechanisms at most once per round trip time, even if there were multiple congestion signals during that period. However, after recovery, the sender and receiver resynchronize cumulative nonces so that further acknowledgements can be checked. We use a simple resynchronization mechanism, which does not explicitly involve the receiver: when the sender receives an acknowledgement for the packet carrying the CWR flag, the sender resets its cumulative nonce to that of the receiver. In most instances, this will be the first acknowledgement without the ECN-Echo flag set.

4.2. Corner Cases

An issue in our design is whether nonces should be on a byte or packet granularity. Nonces are sent per packet, but TCP acknowledgements cover byte ranges. A receiver is not required to send acknowledgements on packet boundaries, and TCP retransmissions in rare circumstances are segmented on different byte boundaries than the original transmission. It is important for our implementation to behave correctly even in these rare cases; a receiver must never be labeled as misbehaving unless this has been proven.

Partial Acknowledgements. If a TCP receiver does not acknowledge the entire packet contents, there is some question as to what cumulative nonce should be returned. We assume that the nonce sent with a packet covers each byte of the packet. Therefore, if any byte of the packet is acknowledged, that packet's nonce is included in the cumulative nonce.

Retransmitted packets. An additional concern is what nonce values to place in retransmitted packets, especially when the retransmitted packets span different byte boundaries than the original packets. According to [22], retransmitted packets are not marked as ECN-capable, preventing them from carrying a nonce. Although the original nonce cannot be sent, the nonces associated with retransmitted packets are unimportant because packets are only retransmitted due to congestion signals. The sender ignores all cumulative nonces until all lost packets have been acknowledged, at which point, the sender's cumulative nonce is reset to match the receivers.

Interactions with Selective Acknowledgement. The TCP Selective Acknowledgement (SACK) [17] option loosens the restrictions of cumulative acknowledgements by allowing a receiver to acknowledge out-of-order packets. This improves performance with small windows and when multiple packets are lost in a single window of data. Although the ECN-nonce could be extended to include the out-of-order packets, this extension is unnecessary, as selective acknowledgements cannot be used to artificially increase the sending rate.

5. Evaluation

In this section, we show that misbehavior can be detected by a one bit nonce. We then describe how to make the single bit nonce effective by responding aggressively when misbehavior is detected. Finally, we demonstrate the effectiveness of the ECN-nonce for short flows.

5.1. Accuracy of One Bit Nonces

While multi-bit nonces improve the accuracy of our cumulative nonce design, we show that it suffices to use only a single bit. Challenging the receiver with a 32-bit nonce would leave so little opportunity for concealing congestion signals that an incorrect 32-bit cumulative nonce could replace the ECN-echo flag. Figure 7 shows the effectiveness of the misbehaver when detected by 1, 2, 4, and 10-bit nonces. The sender treats an incorrect cumulative nonce as though it were a normal signal of congestion. Both 4 and 10-bit nonces converge near the expected, fair share of bandwidth. Although the one bit nonce reduces the performance benefit of misbehavior from 6X in Figure 4 to 1.5X in Figure 7, a more aggressive congestion response is required to eliminate it completely.

5.2. Compensating for One Bit Nonces

A one bit nonce alone would allow a misbehaving receiver to gain 1.5 times its fair share bandwidth: perhaps sufficient for fault detection, but not sufficient to discourage misbehavior. An alternative is to use a more aggressive congestion response when misbehavior is detected. One such response, reducing the congestion window to one packet, is evaluated in Figure 8. Using this aggressive response, the misbehaver's performance is brought in line with the fair share bandwidth, and behaving connections are protected.

The sender can also choose to disable ECN for the misbehaving connection so that additional congestion signals are not concealed. The combination of disabling ECN with

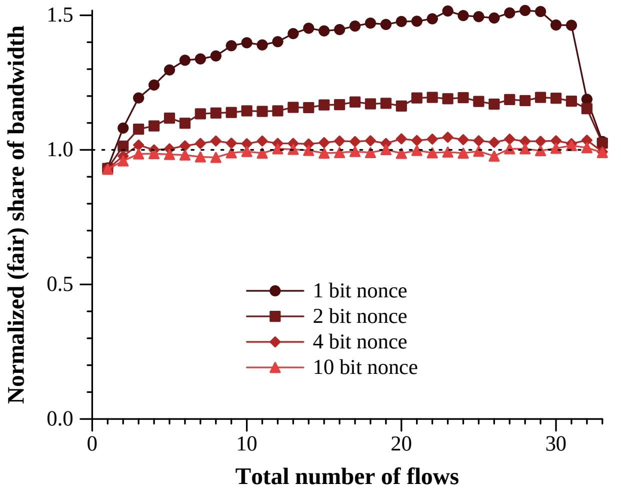

Figure 7. Misbehavior is detected by 1, 2, 4 and 10-bit nonces. When misbehavior is detected, the congestion window is reduced by one-half. A 10-bit nonce closely approximates a true ECN signal, while 4, 2, and 1-bit nonces allow the misbehaver to gain progressively more bandwidth.

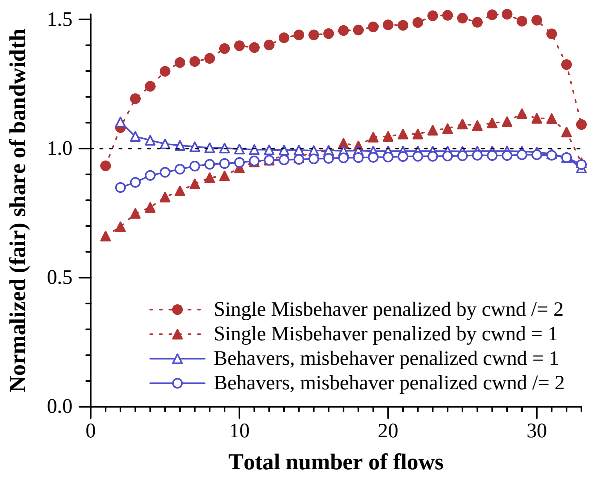

Figure 8. If the sender reduces the congestion window (cwnd) and slow start threshold to one whenever misbehavior is detected, the misbehaving receiver gets between 70-110% of the expected bandwidth. If the sender uses the ordinary congestion response of reducing cwnd by one half, the misbehaver is allowed a significant improvement in bandwidth.

the congestion response in Figure 8 makes the one-bit nonce sufficient to protect against misbehavior. We balance the need to protect against intentional misbehavior with the need to gracefully support non-compliant ECN implementations. We should remove any incentive for intentional misbehavior without needlessly penalizing buggy systems.

5.3. Short Flow Behavior

While long flows that misbehave have the greatest impact on the network as shown in Section 2, we demonstrate that our technique is also valuable for short, less congestion-sensitive flows. We evaluate the performance imbalance of misbehavior as a function of transfer size and show the effectiveness of the one-bit nonce in Figure 9.

Nine long-running ECN flows provide background traffic through the bottleneck link. After these flows saturate the link, a tenth flow begins sending with one of four congestion behaviors:

1. Misbehaver without nonces: The receiver never returns congestion signals.
2. ECN behaver: The receiver correctly returns congestion signals.
3. Misbehaver with nonces: The receiver never returns congestion signals, but is detected by the ECN-nonce. After detecting a concealed mark, the sender reduces the congestion window to one packet and disables ECN for the remainder of the flow.
4. ECN-disabled behaver: ECN is not used for this connection.

We measure transfer rate as the number of acknowledged bytes divided by the time since the connection started. To reduce the number of required simulations, we use a single long flow and estimate the rate at which all smaller transfers would complete by recording when every packet of the flow is acknowledged. To reduce the effects of occasional timeouts and transient behaviors, each rate measurement shown is the average of fifty simulations.

Misbehavior is effective for flows that are long enough to suffer and react to congestion. For flows longer than 50 kilobytes, performance diverges: misbehavior increases performance, correct ECN behavior converges to a fair rate with other ECN flows, and detected misbehavior is reduced to ECN-disabled performance. However, misbehavior can drive the bottleneck queue into severe congestion during slow start, forcing it to drop many packets from all flows. This explains the loss in performance for misbehaving flows between 200 and 400 kilobytes in this simulation. Correct ECN behavior exhibits a smooth transition from slow start into congestion avoidance, converging to its fair share, one megabit per second. Detected misbehavior is recognized quickly, since it is difficult to conceal repeated congestion signals, and performance drops because ECN is disabled after detecting misbehavior. This demonstrates the effectiveness of the one-bit nonce for a range of flow lengths.

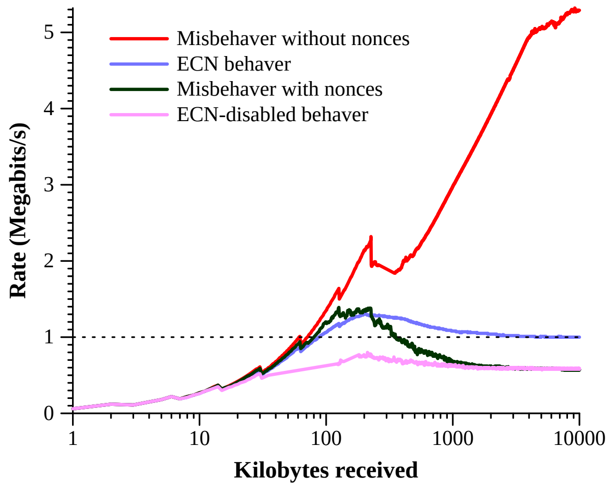

Figure 9. The transfer rate of flows with varying length and different congestion behaviors is shown. Flows shorter than 20KB see few or no congestion signals and thus have similar performance. At 50KB, performance diverges: misbehavior increases the transfer rate in slow start until the bottleneck queue is so congested that several packets are lost and performance suffers, correct ECN behavior smoothly reaches a fair rate, and detected misbehavior may see initial benefits, but is slowed to ECN-disabled performance after an incorrect nonce sum.

6. Related Work

ECN with nonces prevents receivers and network devices from accidentally or intentionally concealing congestion signals from senders and causing them to transmit at an excessive rate. The design builds on our earlier work on ACK verification [25] by extending it to prevent concealment of packet marks, as well as drops, and demonstrating how nonce-based designs can be implemented efficiently.

A different approach is to depend on network level support, enforcing fairness at routers with mechanisms such as Fair Queuing [3], or detecting and punishing flows that are not TCP-friendly [8]. The ECN specification [21] notes that this would limit the impact of both sender and receiver misbehavior. The obvious downside of this approach is that practical, lightweight fairness mechanisms such as those based on the “penalty box” strategy are still a research issue, and unlike ECN, their widespread deployment cannot be depended on in the near future.

However, we see this approach as complementary to our own. Even with network-level support for fair bandwidth allocation, end-to-end congestion control is still needed to adapt to the available bandwidth. Our mechanism is valuable as a check that end-to-end congestion control is being implemented properly. Further, lightweight fairness mechanisms are typically approximate because they make simplifying assumptions to avoid maintaining per flow state. An end-to-end approach, such as ours, enables precise checking of congestion signaling because per flow state is already maintained.

Our ECN-nonce mechanism is loosely based on the ad hoc authentication techniques developed by researchers in response to problems with individual Internet protocols. For example, the Domain Name System (DNS) matches DNS replies to requests by using a randomized transaction identifier. With a good implementation, this prevents an attacker from poisoning the DNS cache with spoofed replies [2]. Similarly, the three-way TCP handshake used for connection establishment [27] checks that the host accepting a connection has successfully communicated with the host initiating the connection. While the original motivation for this design was reliability of connection establishment, exchanging random initial sequence numbers is an effective mechanism for preventing the protocol from being misused to establish connections to spoofed addresses [1]. As another example, Photuris uses an unencrypted random nonce as an inexpensive way to filter certain kinds of denial-of-service attacks [14]. These mechanisms are both efficient and effective, and we have adapted them to the task of congestion signaling over an unreliable channel.

Finally, intrusion detection techniques [20] demonstrate a different approach to the problem of recognizing misbehavior. They aim to identify an attack at an early stage, when it is still possible to deter the attack or otherwise limit its damage. However, this task is complicated by its passive approach of observing activity from select vantage points. For this approach to provide a comparable level of protection to the ECN-nonce, two things are required. First, the passive monitor has to be near the receiver to see all congestion marks. Second, it either has to have a mechanism to deliver notifications of misbehavior to the distant sender, or drop packets to negate the effects of misbehavior. We extended the ECN protocol so that it is straightforward to check behavior and limit the impact of misbehaving parties, making it a failsafe tool that cannot be turned against the sender.

7. Conclusions

It is reasonable to assume that as the Internet continues to grow and mature, problems of misbehavior will be of continued or even greater importance. In this paper, we have shown two results that shed light on misbehavior in the context of congestion control.

First, we have explored Explicit Congestion Notification (ECN) [21] with an eye towards receiver or network misbehavior. ECN is a congestion signaling mechanism in which packets can be marked as well as dropped. When packets are marked, an ECN-capable transport such as TCP must return a congestion signal from the receiver to the sender. We observe that it is easy to cheat in this model by concealing marked packets from the sender; ECN was not designed to prevent such misbehavior. We then showed that cheating is effective at capturing an unfair share of the bandwidth when there is competition with other flows, and does not reduce performance when there is no competition. It is prudent to understand this potential abuse of ECN because it is widely considered to be the next improvement to congestion control to be deployed, and congestion control is central to the robustness of the Internet.

Second, and more importantly, we have addressed the question of whether it is possible to design robust congestion signaling protocols that do not admit the above cheating behavior. The solution we present is both simple and effective. It carries one-bit nonces on unmarked IP packets from the sender to the receiver, and returns one-bit cumulative nonces from the transport receiver to the transport sender. This allows the sender to probabilistically check that congestion signals are not being concealed without trusting any party other than the marking router. We further demonstrate an efficient implementation that is compatible with TCP/IP header space limitations and Internet deployment considerations. While we focus on the combination of ECN and TCP for concreteness, the ideas we present are applicable to other transports such as SCTP [28].

References

[1] S. M. Bellovin. Security Problems in the TCP/IP Protocol Suite. *Computer Communications Review*, 19(2):32–48, Apr. 1989.

[2] Computer Emergency Response Team. CERT Advisory CA-97.22 Topic: BIND - the Berkeley Internet Name Daemon. http://www.cert.org/advisories/CA-97.22.bind.html, Aug. 1997.

[3] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM '89*, Sept. 1989.

[4] W. Feng, D. Kandlur, D. Saha, and K. Shin. Stochastic fair blue: A queue management algorithm for enforcing fairness. In *INFOCOM 2001*, Apr. 2001.

[5] S. Floyd. RED: Discussions of setting parameters. http://www.aciri.org/floyd/REDparameters.txt, November 1997.

[6] S. Floyd. Negotiating ECN capability in a TCP connection, Oct. 2000. http://www.aciri.org/floyd/papers/ECN.Oct2000.txt/.

[7] S. Floyd. Recommendation on using the "gentle_" variant of RED. http://www.aciri.org/floyd/red/gentle.html, March 2000.

[8] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, Aug. 1999.

[9] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM 2000*, Aug. 2000.

[10] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), Aug. 1993.

[11] J. Gettys. Incoming mail isn't happening properly... (firewall's handling of ECN broken). mail to the end2end-interest list, Apr. 2001. http://www.postel.org/pipermail/end2end-interest/2001-April/000730.html.

[12] A. Heffernan. Protection of BGP sessions via the TCP MD5 signature option: RFC 2385, Aug. 1998.

[13] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. RFC 1323, May 1992.

[14] P. Karn and W. Simpson. Photuris: Session-key management protocol: RFC 2205, Mar. 1999.

[15] D. Lin and R. Morris. Dynamics of random early detection. In *SIGCOMM '97*, pages 127–138, 1997.

[16] R. Ludwig and R. Katz. The Eifel algorithm: Making TCP robust against spurious retransmissions. *Computer Communications Review*, 30(1), Jan. 2000.

[17] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options: RFC 2018, Oct. 1996.

[18] J. Nagle. Congestion control in IP/TCP internetworks RFC 896, Jan. 1984.

[19] UCB/LBNL/VINT network simulator - ns (version 2), 2000.

[20] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, Jan. 1998.

[21] K. Ramakrishnan and S. Floyd. A proposal to add Explicit Congestion Notification (ECN) to IP: RFC 2481, Jan. 1999.

[22] K. Ramakrishnan, S. Floyd, and D. Black. The addition of Explicit Congestion Notification (ECN) to IP. Internet Draft, June 2001. draft-ietf-tsvwg-ecn-04.txt, work in progress, approved as Proposed Standard.

[23] R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In *INFOCOM '99*, Mar. 1999.

[24] J. H. Salim and U. Ahmed. Performance evaluation of explicit congestion notification (ECN) in IP networks. RFC 2884, July 2000.

[25] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver, Oct. 1999.

[26] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. *Computer Communications Review*, 28(4), Oct. 1998.

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

[28] R. R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. J. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream control transmission protocol. RFC 2960, Oct. 2000.

[29] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high speed networks. In *SIGCOMM '98*, Sept. 1998.