Implementing Explicit Congestion Notification in ns-3

Brian Paul Swenson and George F. Riley
School of Electrical and Computing Engineering
Georgia Institute of Technology, Atlanta, GA, USA
{bpswenson, riley}@gatech.edu

ABSTRACT

To detect network congestion, TCP typically relies on detecting packet loss. While this is an effective approach for maintaining high throughput for bulk data transfers, a better approach for interactive, time-sensitive, or loss-sensitive traffic would be to detect congestion prior to packet loss. Explicit Congestion Notification (ECN) is a congestion avoidance strategy that makes use of Active Queue Management to allow TCP endpoints to detect congestion without a corresponding packet drop. This congestion detection strategy is particularly useful for delay-sensitive traffic where packet retransmissions can lead to noticeable delays for the user. In this paper, we present an implementation of ECN as an addition to the TCP protocols in ns-3. We have modified all TCP variants currently in ns-3 to work with this new addition. To validate our work we tested all TCP variants and compared our implementation's behavior to previous work.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; I.6 [**Simulation and Modeling**]: General, Model Development, Model Validation and Analysis

General Terms

Implementation, Simulation, Design, Testing, Verification

Keywords

ns-3, TCP, Explicit Congestion Notification, Active Queue Management, RED, Tahoe, Reno, New Reno

1. INTRODUCTION

To control data transfer speeds and to detect network congestion, TCP relies on packet drops. TCP hosts gradually increase the rate at which they send data until a loss is detected at which point they reduce their transfer rate. To minimize the impact of a packet loss from a throughput perspective, TCP congestion management algorithms contain strategies such a Fast Retransmit and Fast Recovery. The process is continuously repeated resulting in the well known TCP Sawtooth pattern. However this approach is not always effective for interactive, time-sensitive or loss-sensitive traffic where a better solution would be to detect congestion and reduce transmission rates without packet loss. Explicit Congestion Notification (ECN) is a congestion avoidance strategy where routers performing active queue management (AQM) mark packets instead of dropping them. The receiver of the marked packet then echoes the congestion notification back to the sender. The sender is then notified of congestion, without packet drops, and can reduce the rate at which it is sending.

This paper presents the work done to incorporate ECN into the TCP models in ns-3. Our ECN implementation is based on RFC 3168, The Addition of ECN to IP[11]. Along with presenting our work, validation is performed by comparing our results with previous ECN simulations[6] performed in ns-2[2]. This work is novel in that it is, to the best of the authors' knowledge, the only ECN implementation for ns-3 that does not require ns-3's packet metadata feature to be enabled.

The remainder of this paper is organized as follows. In section 2, we provide background on ECN and discuss related work. Section 3 describes our ECN implementation and how it was integrated into the existing TCP models in ns-3. Section 4 describes the experiments we performed with our models and results are presented. Section 5 outlines next steps and future work. In section 6 we conclude our work.

WNS3, May 07 2014, Atlanta, GA, USA
Copyright ACM 978-1-4503-3003-9/14/05 ...$15.00.
http://dx.doi.org/10.1145/2630777.2630782.

2. BACKGROUND AND RELATED WORK

In this section we start off by giving background information on how TCP with ECN operates from connection setup through data transfer. We then discuss its current status and research in the area.

Before ECN can be used within a TCP connection, both sides of the connection must first agree to it. This agreement is performed during the initial TCP three-way handshake. When sending the initial SYN packet, an initiating TCP host desiring to use ECN will also mark the ECE and CWR flags, see Figure 1. Both flags need to be marked for the receiving host to interpret the SYN packet as a valid ECN request. For the SYN sender, marking these flags indicate that it is willing to participate in an ECN connection as both a sender and a receiver. As a sender, it will make the necessary markings in outgoing packets so ECN capable routers know to mark

Bit #	Flag
0	CWR
1	ECE
2	URG
3	ACK
4	PSH
5	RST
6	SYN
7	FIN

Figure 1: TCP header flags contained in byte 14 of the TCP header. Commonly used in ECN are CWR and ECE which stand for Congestion Window Reduced and ECN-Echo respectively.

0	0	Not-ECT
0	1	ECT(1)
1	0	ECT(0)
1	1	CE

Figure 2: Values for the ECN Field in IP header

packets if congestion occurs and respond to ECN congestion notifications by reducing its window size. As a receiver, the host will correctly echo congestion notifications back to the data sender.

On reception of a SYN packet marked with the CWR and ECE flags, the receiving host can indicate that it wishes to use ECN by marking the ECE flag in the return SYN-ACK packet. If both hosts agree to the use of ECN, it may be used by both TCP hosts during the TCP connection. If either host does not agree to use ECN, it cannot be used, and any ECN indications during the transfer will be ignored.

Once an ECN connection has been established, hosts need a way to notify queues, along the path between the hosts, that they are ECN-capable and will respond to ECN notifications. TCP hosts do this my marking a ECN codepoint in the IP header of the outgoing packet. For IPV4, the codepoint is placed into bits 6 and 7 of the TOS octet, bits 14 and 15 of the header. For IPV6, the codepoint is placed in the last two bits of the traffic class field, bits 10 and 11 of the header. Figure 2 shows the meanings of the possible values for this field. RFC 3168 recommends TCP hosts use ECT(0) to indicate that they are ECN-capable. ECT(1) is not fully defined in RFC 3168 but it does outline possible uses for it.

Routers using AQM algorithms, such as Random Early Detection (RED)[8], probabilistically drop packets based on average queue length instead of just waiting for the buffer to overflow. Packets marked with a ECT codepoint are eligible to be marked with a Congestion Encountered (CE) codepoint instead of being dropped by ECN-enabled routers. This allows detection of congestion without the typical loss of packets and delays due to retransmission.

When a TCP host receives a packet that is marked with a CE codepoint, it echoes the notification back the sender using the ECE flag in the TCP header. All ACKs the receiver sends back to the sender from this point on with be marked with the ECE flag until the receiving host receives notification that the sender has reduced its congestion window. In response to the congestion notification, the sender cuts its congestion window in half and reduces its slow start threshold similar to the response of three duplicate ACKs. The sender notifies the receiver of the congestion window reduction by marking the CWR flag in the TCP header on the subsequent data packet.

Along with the basic implementation described above, RFC 3168 makes a few further modifications to how the TCP endpoints operate. First, the TCP sender should not increase its congestion window in response to a ACK packet with the ECE flag marked. Second, the TCP sender should only respond to congestion indications once per window of data. Therefore if the sender receives multiple packets within a window of data containing the ECE flag, it should only respond once. Also if the sender has reduced its congestion window in half in response to three duplicate ACKs it should not respond to an ECN notification in the same window and vice-versa. On a timeout of a retransmit timer, TCP should follow its normal procedure however it should not reduce the slow start threshold if it has been decreased within the last roundtrip time.

ECN is available in a wide array of operating systems including: Windows, Mac OS X, Linux, and FreeBSD, and is also supported by Cisco[1]. There have been numerous evaluations of it. Floyd found in simulations that ECN gave a clear, if modest, benefit to TCP/IP networks[5]. Salim and Ahmed reported ECN TCP substantially improved transactional transfers over non-ECN TCP and offered improvement for even bulk transfers[12]. Kadhum and Hassan reported improvements in throughput using ECN for both long[10] and short[9] TCP connections.

3. ECN INTEGRATION WITH NS-3

The first thing that needed to be decided when incorporating ECN into ns-3 was how to mark the packet's IP header with a CE codepoint when the queue detected congestion. Packets with CE marked in the IP header are henceforth referred to as CE packets. Previously Adrian Tam made a RED queue implementation that allowed CE marking however it only worked on IPV4 and it was dependent on packet metadata being enabled[14]. By default packet metadata is disabled in ns-3 for performance reasons[3]. While this approach works and can be easily extended to work with IPV6 traffic, we wanted to create an implementation that was not entirely reliant upon packet metadata being enabled.

In ns-3 the content of a packet is stored in a byte buffer. The buffer contains the serialized version of the packet's headers as well as the packet's data. Therefore the content of the buffer matches what would be found in a real network packet. The issue with this approach is that it is difficult to parse a packet's headers without context. It would require attempting to parse the individual serialized header bytes. Packet metadata alleviates this problem by providing information about the contents of the byte buffer.

In order to offer an implementation that was not solely dependent on packet metadata, we needed to provide the queue with some knowledge of the expected header structure within the packet being enqueued. RED queues in ns-3 are only used as output queues within a net device. All traffic coming into the queue has gone through the IP layer on the node where the queue resides. Packets either came from the application layer on the node that contains the queue, or came into another net device, went up to IP to be routed, and came down to the queue. Therefore, the first header will be the link layer header for the link layer protocol used by the net device, followed by the IP header. However the byte offset to the IP header depends on which link layer protocol is used. To handle this we have added an additional attribute to ns-3's RED queue class called *LinkLayerType*. The field is an enumeration type with available values of `LTYPE_PPP` and `LTYPE_ETHERNET`. Currently RED queues are only used in net devices that use these two link layer protocols. If that were to change in the future, additional header decoding logic would need to be provided to the RED queue class for this to work correctly.

We also allow the user to specify the strategy used by the to find the IP header by adding a new attribute to the RED queue class called *MarkingType*. The field is an enumeration type with available values of `MARK_METADATA` and `MARK_HEADER`. If `MARK_HEADER` is selected, the queue will use the *LinkLayerType* attribute to determine the first header in the packet. After the link layer header is removed, it will then examine the next byte to determine if the IP header is IPV4 or IPV6 and remove the header accordingly. The IP header can then marked appropriately and it and the link layer header can be readded. If `MARK_METADATA` is selected, the queue will use the packet's metadata to find the IP header. This mode would still be useful if the queue is placed into a net device that is using a link layer protocol that is currently not being handled by the RED queue class.

The remaining changes we have made involve the classes shown in figure 3. To enable a TCP socket with ECN the user sets the *EcnEnabled* attribute in *TcpSocketBase* to true. As mentioned previously, this must be done for both endpoints of the TCP connection before either will use ECN. Logic has been added to the connection setup phase to correctly negotiate ECN usage. In our implementation, following the initial connection setup phase, the sender always marks the TOS octect in outgoing packets with ECT(0) as recommended by RFC 3168. The ECT(1) designation is currently not used. We have added code to the TCP models to correctly mark and respond to ECN-Echo ACKS and to also mark the CWR flag following congestion window reductions. Since the congestion window and slow start threshold variables are defined in the TCP variant subclasses, a purely virtual method called *ECNCongestionControl* has been added to *TcpSocketBase* and is implemented in each variant. One benefit of this approach is that it allows each variant type to specify its specific ECN response.

4. EXPERIMENTS

For all of the following experiments a simple dumbbell topology was used with one leaf on each side, see Figure 4. A TCP bulk sender application was configured on node A and was set to send data to node D. A TCP sink was setup on node D to receive the data. Node B was given an ECN-capable RED queue. The minimum and maximum thresholds for the RED queue were set to 3.0 and 9.0 respectively. The reason these were lowered from their default values was to speed up the marking behavior since the parameters are based on average size. The other nodes were given the default drop-tail queues. The TCP segment size was set to 536 bytes and the slow start threshold for node A was set to 18760 bytes, or 35 packets. Finally TCP's delayed ACK was disabled, i.e. set to one segment.

The graphs for the results show the packets transmitted during each run. The Y-axis shows the packet's number modulo 90. The packet number is the packet's sequence number divided by the packet's size. The X-axis represents time. For each packet there is a red mark at the time it

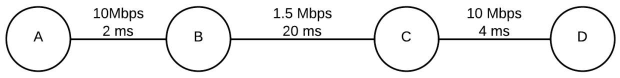

Figure 4: The network topology used for the verification tests

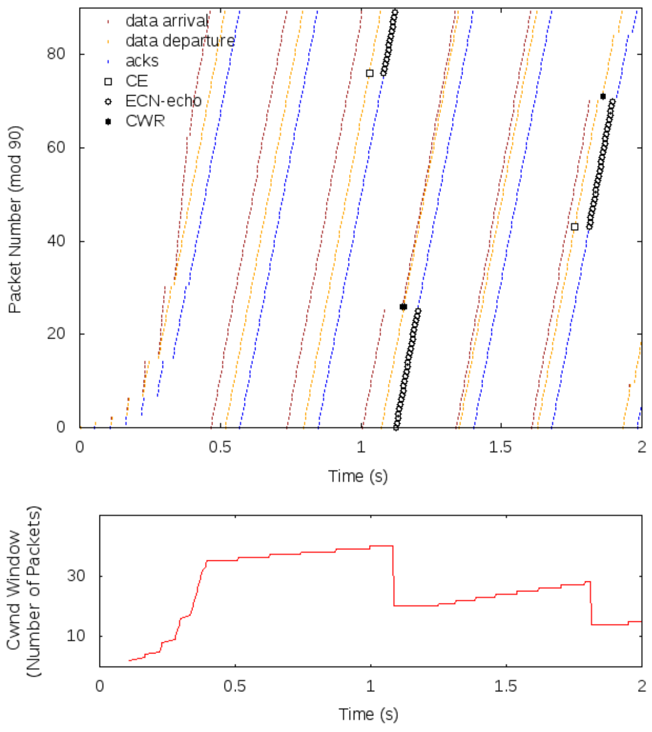

Figure 5: This run shows ECN operating with no dropped packets.

arrives into the queue on node B, see figure 4. If the mark has been replaced with a black square, the packet has the CWR flag marked in its TCP header. Next, there is a yellow mark at the time the packet leaves the queue on B. If this mark has been replaced with a white square, the packet has been marked by the congestion detection algorithm running in the RED queue. The queueing delay the packet experiences at node B can be determined by the distance between the packet's first and second mark. Finally, for each packet, there is a third mark made at the time when its corresponding ACK, sent by the packet sink on node D, arrives back at node B. If the third mark has been replaced with a white circle, the ACK has the ECE flag marked in its TCP header.

Figure 5 shows the data and ACK packets for a TCP connection using TCP Tahoe; however, the results would be the same for Reno and New Reno. Following slow start, the congestion window continues to grow until a CE is received by node D. Following this, node D sets the ECE flag on all ACKs until it receives notification from node A that it has reduced its congestion window. Note that following the CWR packet there is a substantial reduction in the queueing delay.

Figure 6 shows a zoomed view of Figure 5 during a con-

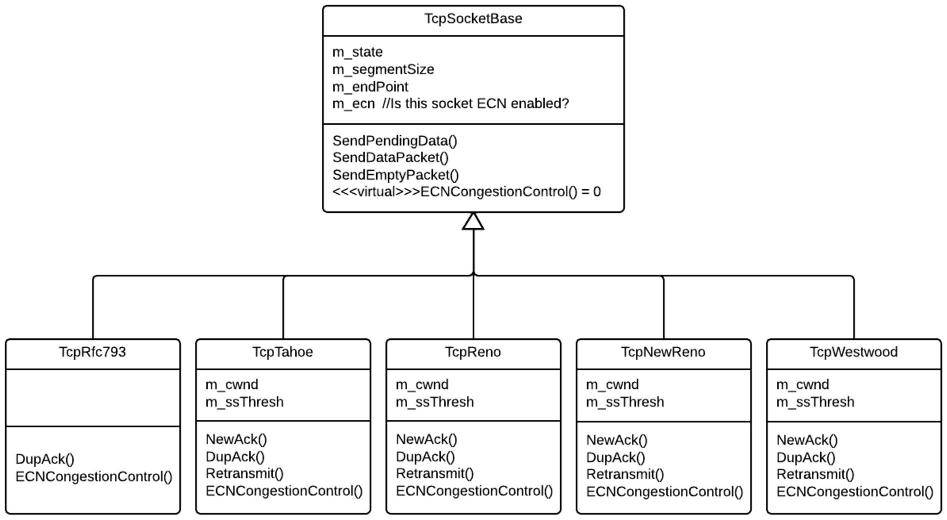

Figure 3: Partial class diagram of ns-3's TCP models with our ECN implementation

gestion event. This figure shows that the TCP sender at node A is not increasing its congestion window when it receives ACKs marked with the ECE flag. This is the expected behavior described in [11].

The next test was to verify that the ECN implementation was correctly responding to congestion notifications only once per window of data. To test this, the *Wait* parameter on the RED queue was changed from its default value to false. Figure 7 shows a TCP connection where two CE packets are produced at node B within one window of data. Notice that the congestion window is only reduced once. This result matches the behavior shown in the `ecn_twoecn_tahoe` test in ns-2.

The next set of tests mix packet drops with ECN notifications. For these tests, a ReceiveListErrorModel error model was placed on node B. The purpose of these tests are to show that the implementation is correctly treating multiple congestion events within one window of data as a single instance of congestion. The following graphs omit the ACKs for clarity.

Figure 8 shows a TCP Tahoe run where a drop closely follows a CE packet. On the reception of the ECE packet, node A reduces its congestion window by half. Subsequently, it receives three duplicate ACKs and reduces its congestion window to one and begins Slow Start. Despite these two congestion events, the slow start threshold is only decreased once, from 35 to 17. This can be observed in the congestion window graph; the subsequent Slow Start period ends at 17 packets. These results match the behavior shown in the `ecn_drop_tahoe` test in ns-2.

Figure 9 shows a similar run using TCP Reno. Here the congestion window is only reduced once since Reno does not slow start following the detection of a lost packet. Since the two congestion events occurred in the same window of data, the congestion window and slow start threshold are only reduced once. TCP New Reno produces a similar output.

In figure 10 we show a run where a packet drop occurs just before a CE packet. The TCP variant used in this experiment is TCP Tahoe. Since these are in the same

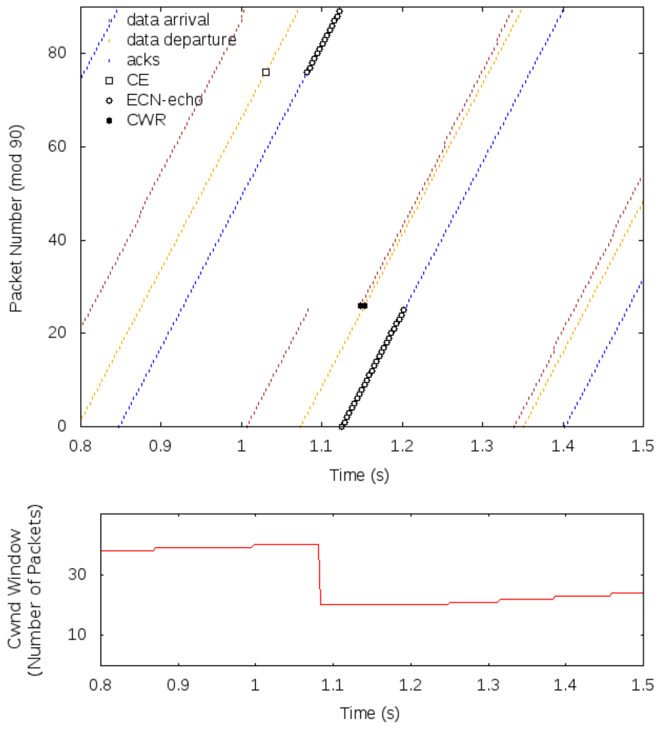

Figure 6: Zoomed in view of figure 5 showing ECN-echo ACKs not increasing cWnd

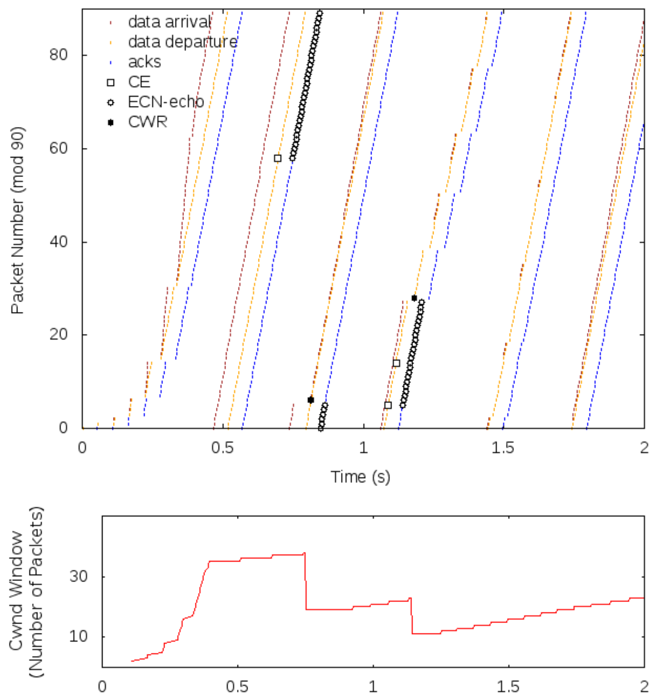

Figure 7: This run shows multiple CE packets within a single window of data results in only one congestion response.

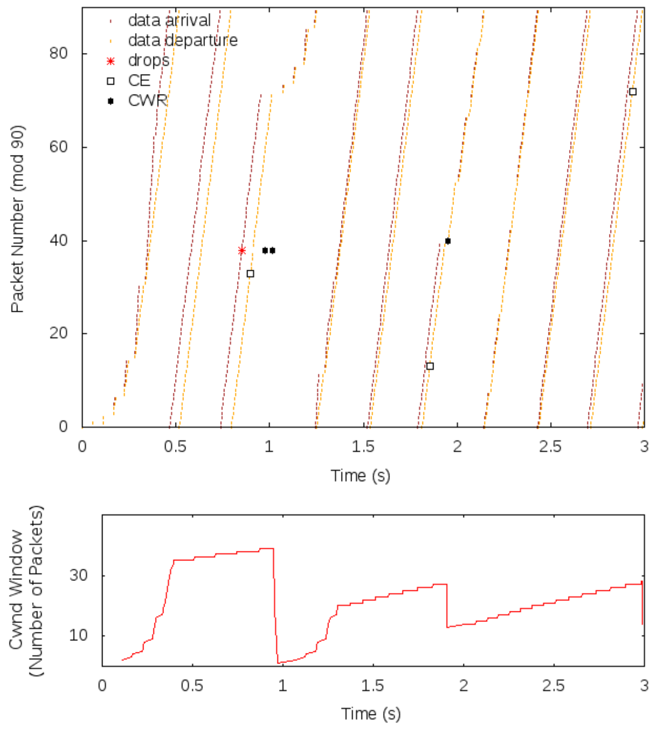

Figure 8: This run shows a CE packet followed by a drop treated as one congestion event in TCP Tahoe.

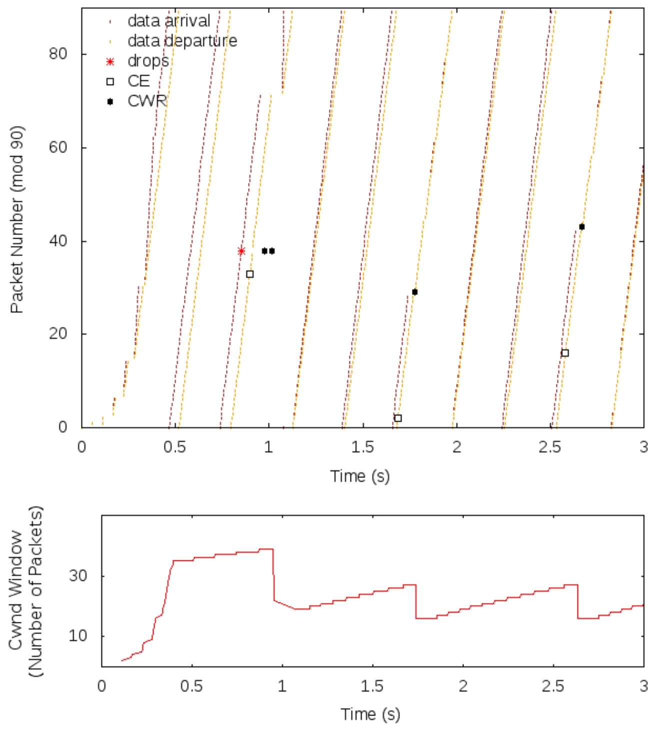

Figure 9: This run shows a CE packet followed by a drop treated as one congestion event in TCP Reno.

window of data they are treated as one congestion event. The congestion window and the slow start threshold are only reduced one time. This matches the behavior shown in the `ecn_drop1_tahoe` test in ns-2.

Figure 11 shows the same experiment as shown in figure 10 except the TCP variant used is New Reno. Again the multiple instances of congestion within a single window of data are treated together as a single instance. The congestion window and the slow start threshold are both halved one time. Following the congestion window reduction there is a significant reduction in queueing delay.

In Figure 12 TCP Tahoe is again used. In this experiment there is a larger gap between the dropped packet and the CE packet. The gap is large enough that node A has already started slow start prior to the arrival of the ECN-Echo from node D. The second instance of slow start starts at time .9266 with the arrival of the third duplicate ACK. The ECE packet arrives at node A at time .9513. This matches the behavior shown in the `ecn_drop2_tahoe` test in ns-2.

In figure 13 we show a non-ECN TCP run using TCP New Reno. The run contains six consecutive dropped packets. The large spike in the congestion window between 1.1 and 1.5 seconds is due to New Reno's fast recovery algorithm as described in RFC 2582[7]. Each duplicate ACK received during the fast recovery phase increases the congestion window by one MSS. This continues until the highest sequence number in flight at the time congestion was detected has been ACKed.

In figure 14 we show the same scenario as in figure 13 except ECN is turned on and a CE packet occurs just prior to the series of packet drops. In this case, all of the ACKs that

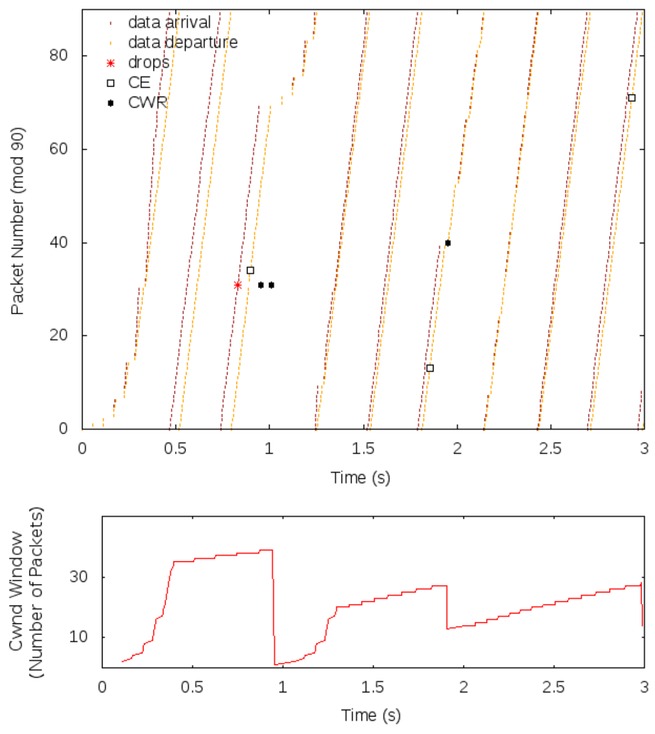

Figure 10: This run shows a drop followed by a CE packet treated as one congestion event in TCP Tahoe.

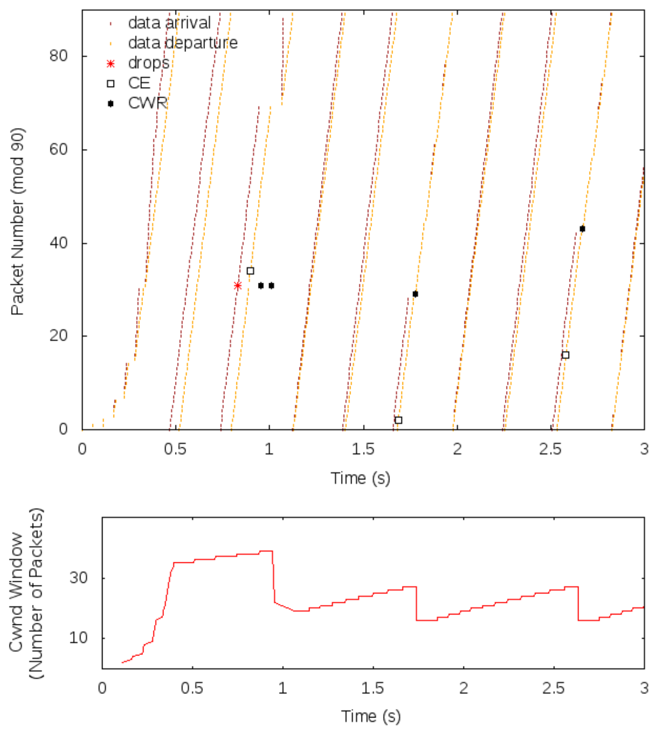

Figure 11: This run shows a drop followed by a CE packet treated as one congestion event in TCP New Reno.

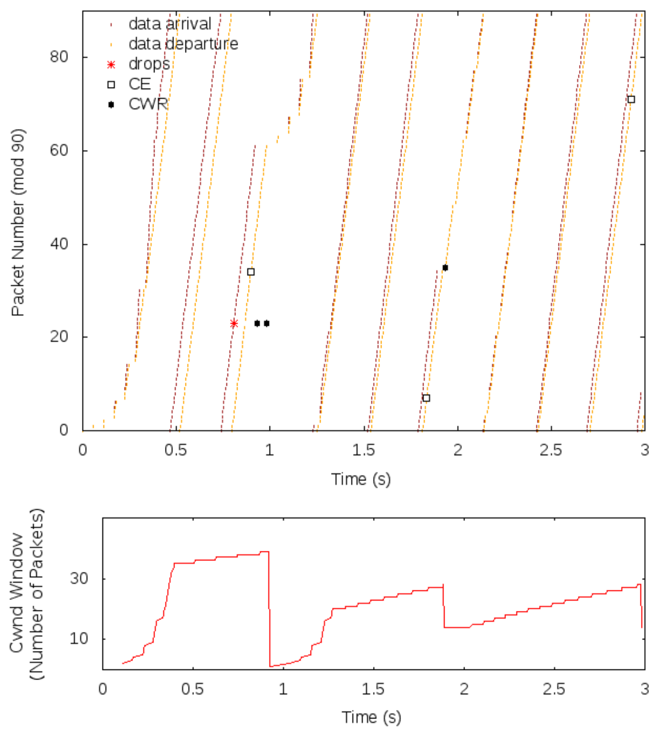

Figure 12: This run shows a drop followed by a CE packet treated as one congestion event in TCP Tahoe. The CE packet occurs after the sender has initiated slow start.

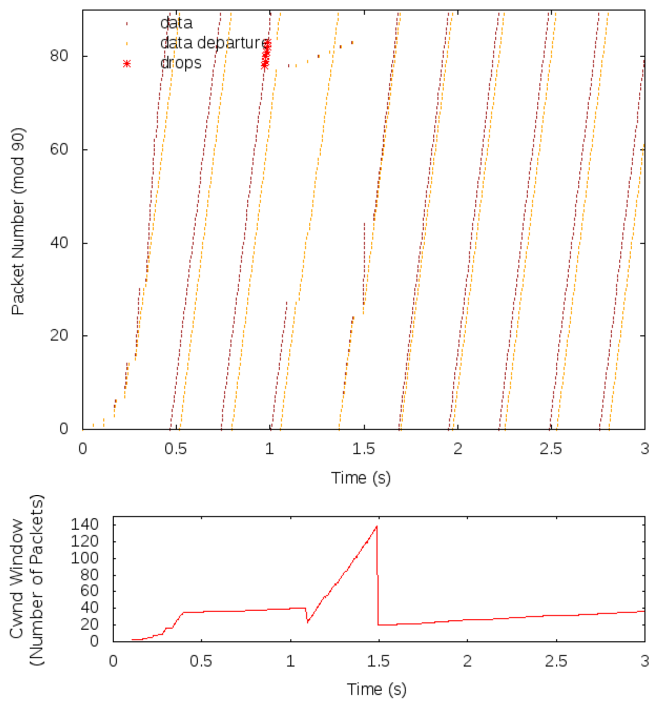

Figure 13: This run shows multiple drops without ECN in TCP New Reno.

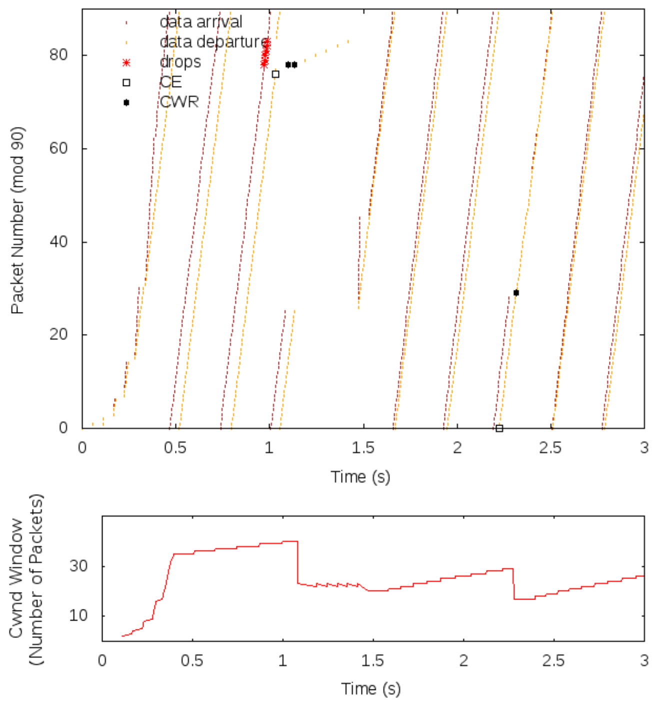

Figure 14: This run shows multiple drops following a CE packet in TCP New Reno.

the sender receives are marked as ECN-echo and therefore the congestion window is not increased.

Figure 15 shows a non-ECN TCP New Reno run where a large burst of packets is dropped leading to a retransmission timeout. Following the timeout, the congestion window is reduced to one and slow start is repeated. The slow start threshold is set to half of the congestion window at the time of the timeout.

In figure 16 we show the same scenario as in figure 15 except ECN is turned on. Here a CE packet occurs just prior the burst of packet drops. Once the sender receives the ECN-Echo ACK corresponding to the CE packet, the congestion window is cut in half and the slow start threshold is reduced. Subsequently a retransmit timeout occurs and the congestion window is reduced to one packet. Note however that the slow start threshold isn't reduced two times. Slow start continues up to approximately 17 packets just as it did in figure 15.

Figure 17 shows a similar result as in figure 16 however this time the CE packet occurs during the burst of dropped packets. This test shows that the sending TCP host is correctly recognizing ECE packets on both new and duplicate ACKs. The first ECE packet arrives on a duplicate ACK at around one second and initially cuts the congestion window in half prior to the retransmit timeout. The second arrives at around 2.3 seconds on a new ACK.

5. NEXT STEPS AND FUTURE WORK

Our code is currently publicly available in the ns-3 repositories[13]. Contained within the scratch folder are all of the tests used to generate the results presented here. Ideally all of the tests would be turned into ns-3 regression tests similar to what already exists for other TCP tests. Before our code

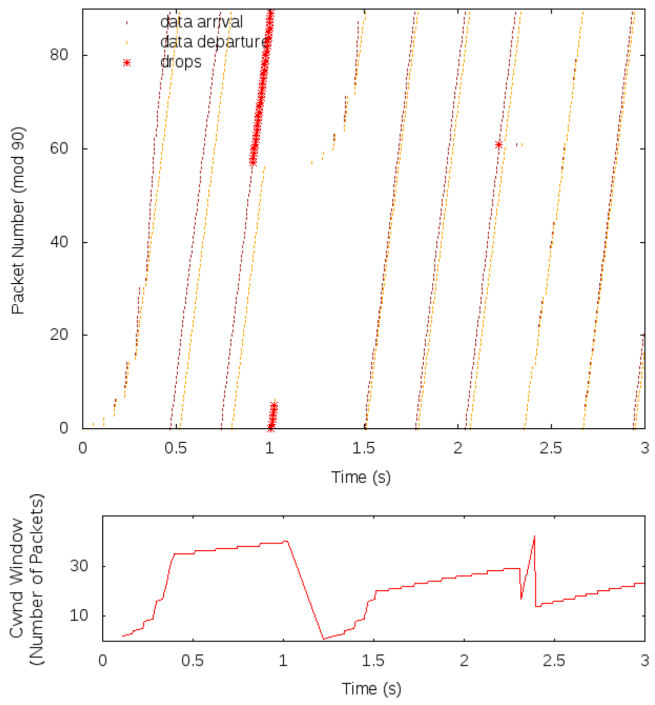

Figure 15: This run shows a large number of drops leading to a time out in TCP New Reno.

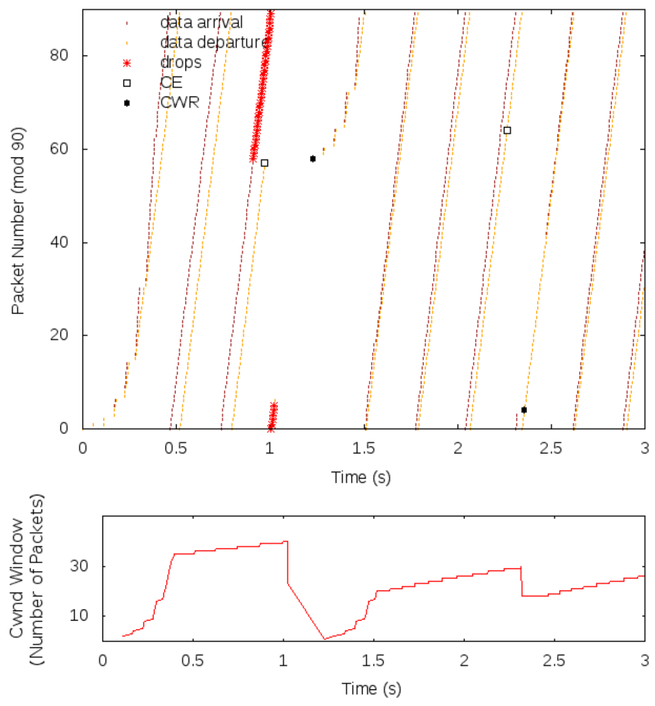

Figure 16: This run shows a CE packet followed by a large number of drops leading to a time out in TCP New Reno.

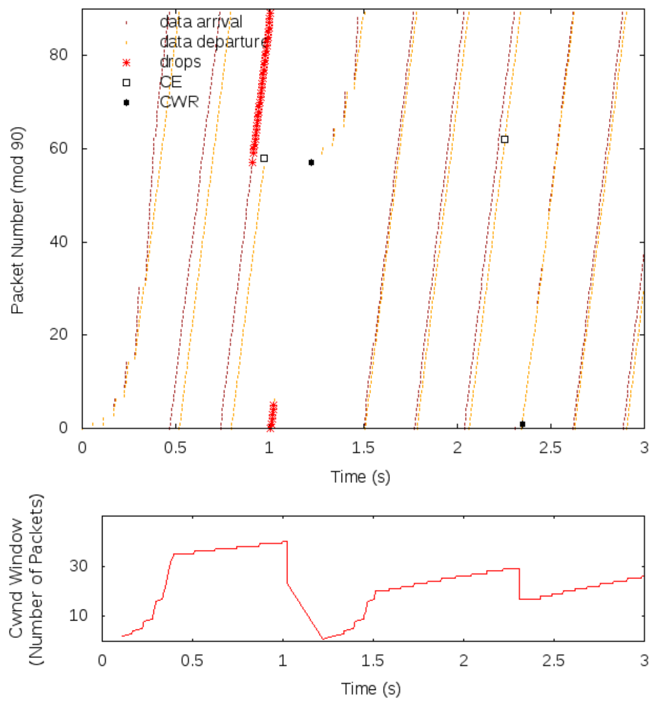

Figure 17: This run shows a CE packet occurring during a large number of drops leading to a time out in TCP New Reno.

could be merged into the main development tree, it would need to go through a code review where we are sure we would receive constructive suggestions which would lead to better overall product and easier integration into the code base.

We are now also working on a Data Center TCP variant (DCTCP)for ns-3. DCTCP is a TCP variant designed to provide high burst tolerance and low latency for short flows[4]. DCTCP has also been shown to use less buffer space than regular TCP. DCTCP uses ECN to provide feedback to the end hosts. The main differences between regular TCP with ECN and DCTCP involves how the ECN-Echo indications are used. We hope to have this code completed soon.

6. CONCLUSIONS

Explicit Congestion Notification is a congestion avoidance strategy that allows TCP end points to detect network congestion without packet loss. It is useful in situations where TCP traffic is time-sensitive or loss-sensitive, such as in an interactive application. In this paper we presented our new ECN implementation in ns-3 which is based on RFC 3168. We discussed how we extended the current TCP models to incorporate our work. Our implementation was validated using similar scenarios as were used in the original work in ns-2.

7. ACKNOWLEDGEMENTS

We wish to thank Tom Henderson for his extremely helpful suggestions and advice which helped us greatly with this implementation. Also we would like to thank Adrian Tam for his previous work on marking packets within RED queues.

8. REFERENCES

[1] The ECN Webpage at icir.org.

[2] The Network Simulator NS-2. `http://www.isi.edu/nsnam/ns/`.

[3] ns-3 manual. `http://www.nsnam.org/docs/release/3.15/doxygen/index.html` [Accessed 06/19/2014], 2012.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). *SIGCOMM Comput. Commun. Rev.*, 41(4):–, Aug. 2010.

[5] S. Floyd. TCP and explicit congestion notification. *SIGCOMM Comput. Commun. Rev.*, 24(5):8–23, Oct. 1994.

[6] S. Floyd and K. Fall. ECN implementations in the ns simulator, 1998.

[7] S. Floyd and T. Henderson. The NewReno modification to TCP's fast recovery algorithm", RFC 2582, 1999.

[8] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, Aug. 1993.

[9] M. Kadhum and S. Hassan. The effect of ECN on short TCP sessions. In *IEEE International Conference on Telecommunications and Malaysia International Conference on Communications (ICT-MICC 2007)*, pages 708–712, May 2007.

[10] M. Kadhum and S. Hassan. A study of ECN effects on long-lived TCP connections using red and drop tail gateway mechanisms. In *International Symposium on Information Technology (ITSim 2008)*, volume 4, pages 1–12, Aug 2008.

[11] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001.

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

[13] B. P. Swenson. ns-3-ecn. `http://code.nsnam.org/brian/ns-3-ecn`, 2014.

[14] A. S. Tam. ns-3 incast. `http://code.nsnam.org/adrian/ns-3-incast/rev/92cc560ae935` [Accessed 06/19/2014], 2011.