The ISP Column
A column on various things Internet

September 2025
Geoff Huston

Measuring Explicit Congestion Notification

Most of today's transport on the public Internet still uses TCP (and in this admittedly sweeping generalization I'll include QUIC, as QUIC can be seen as an updated form of TCP that happens to use UDP as an encapsulation protocol). TCP does not operate at a fixed transmission rate and instead the protocol uses a feedback loop between the sender and receiver to maintain a suitable rate that makes efficient use of the network while avoiding driving the network into conditions of sustained overload. TCP generally achieves this outcome by oscillating the sending rate, lifting the sending rate up to the point where the protocol determines that is has reached the onset of network path congestion, then backing off the sending rate, and then performing the same rate variation cycle again.

There are a number of ways that TCP can sense the onset of network path congestion.

Packet Loss as a Congestion Signal

"Classic" TCP uses an extremely simple loss-based congestion detection algorithm that is intended to save networks from collapsing under extreme overload. The steady state control algorithm used by TCP senders continually increases the rate of data being pushed into the network until the data load exceeds a link's capacity, at which point the link's driving buffer will start to fill. When this buffer is full, the router will be forced to discard excess packets. If the data session is a reliable transport session (TCP in the case of the Internet) the receiver will detect this packet loss, and signal this back to the sender in the reverse acknowledgement stream through so-called "*duplicate ACKs*". The sender will pause sending for a brief period (to allow the queue to drain) and then resume at half the sending rate when the packet loss occurred. This is a rough description of the TCP Reno congestion control algorithm for TCP, shown in Figure 1.

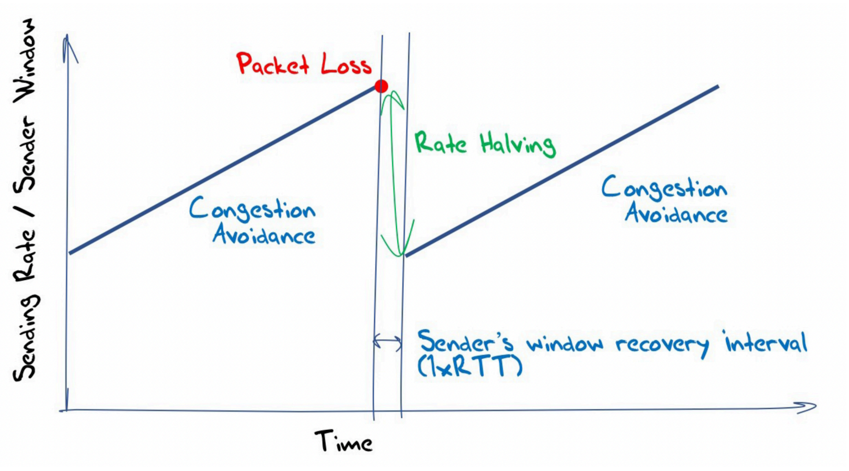

Sending Rate / Sender Window
Packet Loss
Rate Halving
Congestion Avoidance
Congestion Avoidance
Sender's window recovery interval (1xRTT)
Time

Figure 1 – TCP Reno Operation

How big should the network link's buffer be? Intuitively, it sounds like bigger should be better, but that's not necessarily the case. When the buffers are significantly larger than the delay bandwidth product of the link that they are driving, then the sender's delay after detection of packet loss does not fully drain the queue, so when sending is resumed there are still some packets that are resident in the queue (Figure 2). This standing queue acts as a delay element increasing the round-trip time and decreasing the sensitivity of the round-trip control signal. Conversely, making the queue too small implies that the packet loss event occurs too soon and the recovery period and session resumption at the slower speed does not fill the transmission path and there is unused capacity in the network (Figure 3).

This has led to the general rule of thumb in network provisioning that network buffers should be dimensioned to the product of the delay times the link bandwidth of the link being driven.

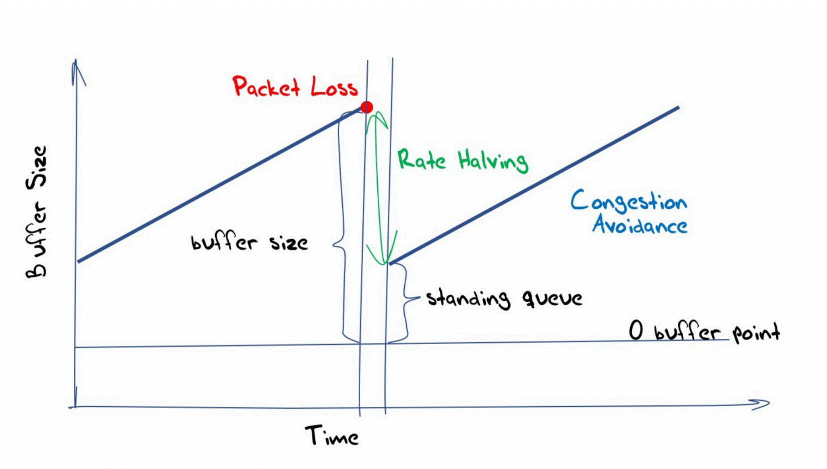


Figure 2 – TCP Reno with large queues

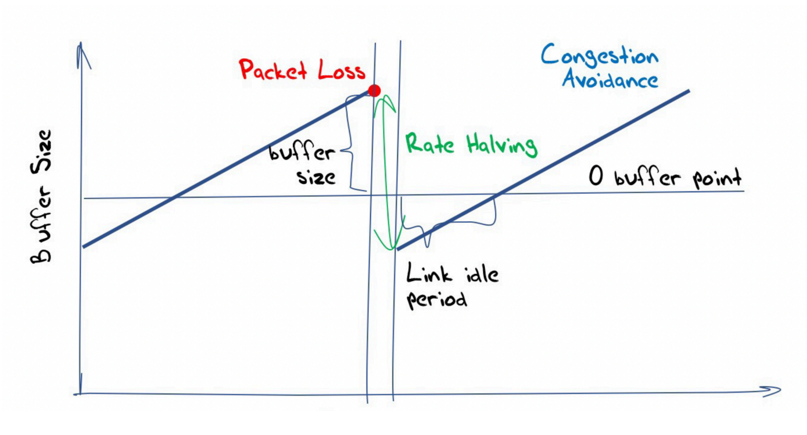


Figure 3 – TCP Reno with small queues

There is an issue with the use of duplicate ACKs to detect packet drops, namely the behaviour of *tail drop*. If the sender behaviour is bursty then a burst of packets may exceed the capacity of a link, and the trailing sequence of packets will be queued. It the final packets of that burst are dropped then there are no ensuing packets that will cause the sender to generate duplicate acks. The result is that the receiver may wait for up to a full RTT time interval to receive an indication of packet drop. This will impact on the efficiency of the protocol. The general advice for all TCP is to avoid packet bursts and instead use *packet pacing* to pace the sending of the packets in the current sending window across the estimated round trip time interval. This will drastically reduce the likelihood of tail drop and the reduced sending efficiency that tail drop induces.

The advice about the size of network buffers being equal to the capacity of the driven link is that that this is not necessarily sound advice when we move from a network with a single active transport session to a network with many simultaneous active sessions. There was a supposition that the high-volume TCP flows tend to dominate the queue behaviour and all other concurrent TCP flows synchronise against these dominant flows, and the aggregate behaviour of the elemental flows was similar to a single large flow. Subsequent experimental work has pointed to a queue capacity which is proportional to the inverse of the square root of the number of simultaneous active sessions. Link efficiency can be maintained for N desynchronised flows with a buffer that is dimensioned to the size of:

$$Size = \frac{BW \cdot RTT}{\sqrt{N}}$$

This is a radical result for high speed extended latency links in a busy network and the consequences on router design are significant. For example, a 1 Tb/s circuit carrying one TCP flow with an RTT_{min} of 100ms would require 12.5 GB of buffer and it's likely that this amount of memory can only be provisioned using high speed off-chip buffering. If this circuit carries 100,000 flows, then this result indicates that the buffer can be safely reduced to less than 40MB, reducing the buffering and worst-case latency by 99.7%. With such small buffers, the buffer would comfortably fit on a single chip switch ASIC.

Because of this dependence on the profile of usage of a link, there is no generally applicable "rule of thumb" for buffer provisioning that is independent of the profile of activity that the associated circuit is supporting.

Delay as an Alternative Congestion Signal

Part of the problem with a loss-based congestion control signal is that the loss signal is too late. Loss occurs when the transmission element is full and the backlog of queued packets waiting to access the transmission element reaches the capacity of the buffer. Once this point has been reached, the sender should pause sending to allow the backlog to clear and then resume at a slower pace. One the other hand, the optimal control point lies at the onset of queue formation.

The ideal control behaviour is for the sender to oscillate its sending rate around this queuing onset point (Figure 4).

The challenge lies in informing the sender as to the onset of queuing within the network. One approach, used by the BBR (Bottleneck Bandwidth and Round-trip propagation time) protocol, generates a periodic "pulse" of traffic into the network which is some 25% larger than the previous steady state sending rate for one round trip time interval. If the measured round-trip time increases for this pulse, then the sender can assume that the additional delay was due to the onset of queuing, and after allowing some time (one round trip time interval) for the queue to drain, the sender will resume its previous sending rate. If the pulse is absorbed by the network without any additional delay, then BBR interprets this situation as one where the network has additional unused capacity, and it will ramp up its sending rate for the next period.

BBR is an example of a *delay-based* congestion control algorithm. Both delay-based congestion control algorithms and *loss-based* congestion control algorithms are based on the sender being triggered by the second order effects from the buildup of queues within the network. There is another approach, namely for the network elements to mark packets when queues are being formed within the network element itself.

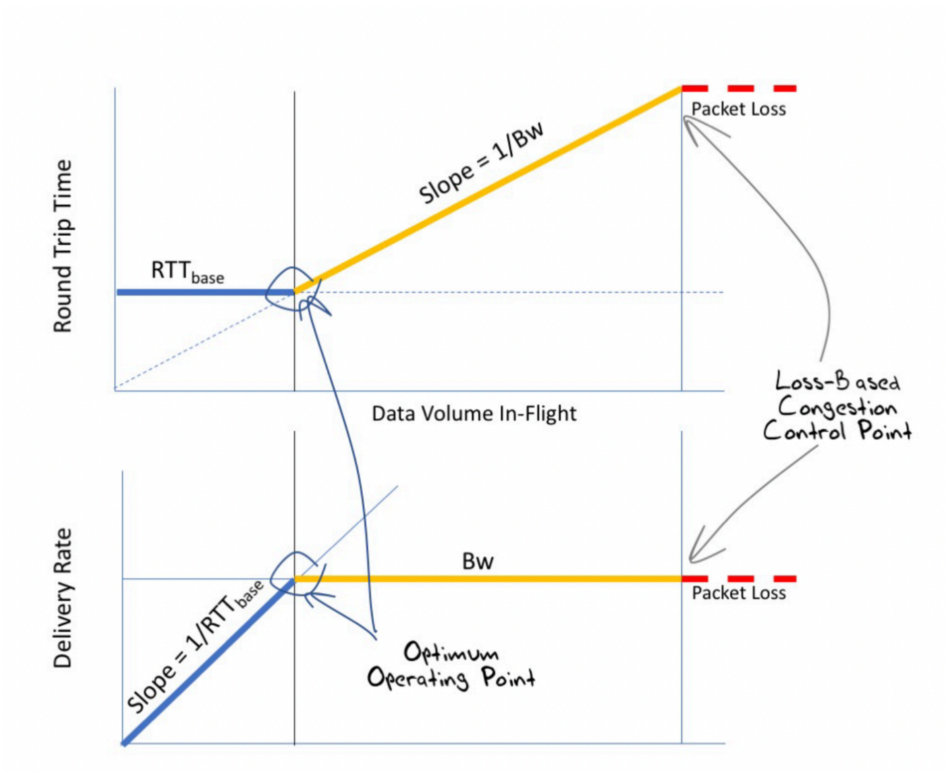


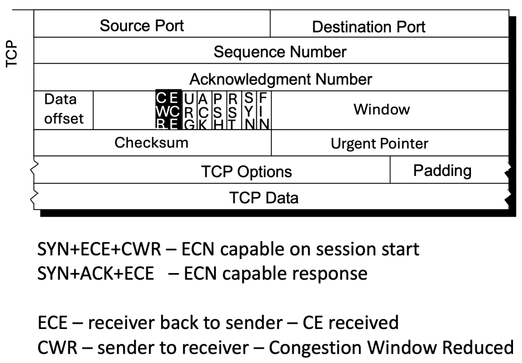
Figure 4 – Congestion Control Behaviour

These approaches are based on the sender reacting to the indirect effects of congestion on the carriage of packets across the network, either through packet loss or by increase in the round trip time between the sender and receiver (the time to transmit the packet and receive an acknowledgement). But why use such indirect measurements? Can we get the network itself to report when the network is experiencing congestion?

Network Notification of Congestion

A paper describing such an approach was published in May 1990: "A binary feedback scheme for congestion avoidance in computer networks" by Ramakrishnan and Jain. The paper describes the approach as follows: "The scheme uses a minimal amount of feedback from the network to the users, who adjust the amount of traffic allowed into the network. The routers in the network detect congestion and set a *congestion-indication* bit on packets flowing in the forward direction. The congestion indication is communicated back to the users through the transport-level acknowledgment." This work was taken up by RFC 2481, describing an experimental protocol that proposed to add Explicit Congestion Notification (ECN) to IP. The central idea is that you don't need to push the network to the point of packet loss (and retransmission) to inform a sender that its sending rate is exceeding the network's capacity. The network itself can inform the endpoints of the transport session of this situation.

The operation of this mechanism requires a level of cooperation between TCP and IP in the host. To start using ECN in a TCP session, then the first step is that the initial TCP SYN packet must set the set the ECN-Echo and CWR flags in the TCP header (as well as the SYN flag, of course). (These TCP flags in the TCP header are indicated in Figure 5.)

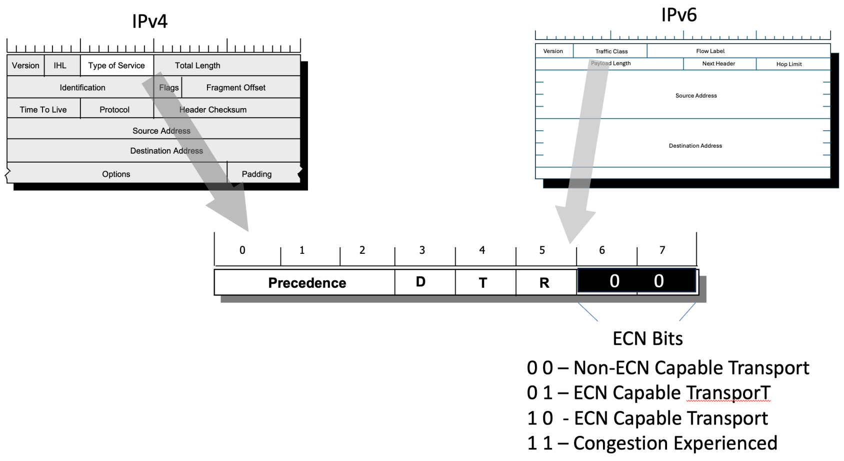


SYN+ECE+CWR – ECN capable on session start
SYN+ACK+ECE – ECN capable response

ECE – receiver back to sender – CE received
CWR – sender to receiver – Congestion Window Reduced

Figure 5 – TCP ECN Flags

A SYN packet with both the ECN-Echo and CWR flags set indicates that the TCP implementation transmitting the SYN packet is willing to participate in ECN as both a sender and receiver. An ECN-aware receiver will positively respond to incoming ECN-capable TCP SYN packet with a SYN-ACK packet that has the ECN-Echo bit set. Thereafter all TCP packets in this session in both directions with a non-zero length will have either the IP ECT(0) or IP ECT(1) bit set in the IP header of the session packets. This bit setting in the IP packet header is the responsibility of both end hosts in an ECN-aware session. (The IP header flags are shown in Figure 6.) ECN uses two bits of what was originally the IP Type of Service field in the IPv4 packet header and the Class of Service field in the IPv6 header.



ECN Bits
0 0 – Non-ECN Capable Transport
0 1 – ECN Capable TransporT
1 0 - ECN Capable Transport
1 1 – Congestion Experienced

Figure 6 – IP ECN Flags

The ECN control sequence is as follows:

```
Client -> Server: TCP Flags [SEW]
Server -> Client: TCP FLAGS [S.E]
Client -> Server: TCP Flags [.]
Client -> Server: IP ECT(0) LENGTH 569, TCP Flags [P.]
Server -> Client: IP ECT(0) LENGTH 2351, TCP Flags [P.]
Client -> Server: IP LENGTH 0, TCP Flags [.]
```

An ECN-configured router will mark congestion on those IP packets where either of the IP ECT bits are set. The router is supposed to mark such packets with a *congestion experienced* indicator (IP CE) (by setting both IP ECT-0 and IP ECT-1 to 1) if internal queues are forming within the router. The basic mode of ECN operation is that if an ECN-capable router would've selected a packet for discard and the packet has either of the IP ECT(0) or ECT(1) bits set, it will set the other bit (thereby setting the ECN bits of the IP header to IP CE) instead of discarding the packet, and it will then forward the packet onward.

When the destination host receives a TCP packet with the IP CE flag bits set, and the packet is part of an ECN-aware session, then the destination will echo back the CE condition to the sender using the TCP ECN-Echo bit. It will continue with this signal until the remote host acknowledges that its sending window has been reduced by setting the TCP CWR flag in the next packet sent to the remote end.

```
Client -> Server: IP CE
Server -> Client: TCP Flags E
Client -> Server: TCP Flags W
```

This ECN proposal was revived to the status of a standards track protocol 2001 in RFC 3168.

After the publication of RFC3168 nothing much happened for a couple of decades. I suspect that ECN fell into the expanding chasm of distrust between the network and host-based applications.

L4S

ECN was revived as part of the work on Low Latency, Low Loss and Scalable Throughout (L4S) framework (RFC 9330). To quote from the abstract of this specification: "L4S is based on the insight that the root cause of queuing delay is in the capacity-seeking congestion controllers of senders, not in the queue itself. With the L4S architecture, all Internet applications could (but do not have to) transition away from congestion control algorithms that cause substantial queuing delay and instead adopt a new class of congestion controls that can seek capacity with very little queuing. These are aided by a modified form of Explicit Congestion Notification (ECN) from the network. With this new architecture, applications can have both low latency and high throughput."

In many ways, this is similar to the intent of delay-based congestion control algorithms, seeking to control the traffic rate to operate at the point of onset of queuing within the network's routers, rather than at the onset of packet loss. The idea behind the ECN component of the L4S framework is that instead of performing packet marking as an alternative to packet discard when the queue is close to full (which was the approach used in the original ECN design), the L4S framework calls for ECN marking to occur at the onset of queuing. To distinguish these two marking behaviours, L4S exclusively uses the IP ECT(1) bit, working on the assumption that most ECN implementations have used IP ECT(0).

L4S routers need isolate L4S traffic from all other through the use multiple queues. The high amplitude of queue size oscillation in both "classic" ECN and loss-based congestion control is isolated from the small queue and associated low latency transit is needed to support the low latency aspect of L4S. Two queuing mechanisms have been defined for this purpose. One is referred to as Dual-Queue Coupled AQM, the other is an L4S-aware flow queuing approach. Dual-Queue Coupled AQM routers have two separate queues at the network bottleneck, one for L4S traffic and one for classic traffic, where the IP ECT(1) ECN field is used as an identifier for L4S packets to be distinguished from classic packets. The queue for L4S traffic has a shallower (or smaller) buffer size, which allows the L4S packets to experience very low queueing delay. Additionally, packets in the L4S queue are marked with IP CE in the ECN field (to notify the endpoints of impending congestion) as soon as they start building up in the queue (e.g. when the queue delay exceeds a low threshold of some 500µs or 1ms). The queue for classic traffic has a larger buffer to maintain full utilization since the queue needs to be large enough to cope with large amplitude of rate variations by a classic TCP loss-based congestion control behaviours. In addition, despite the use of separate queues, the congestion signalling is coupled between the queues so that the two types of traffic share the bottleneck bandwidth in a fair manner. The L4Saware flow queuing approach provides a separate queue for each individual transport flow and provides IP CE marking of IP ECT(1) packets via a shallow queue delay.

I won't go further into the detained workings of L4S here, as the subject deserves a far more detailed explanation, but suffice it to say that L4S is seen as a tool for a collection of applications, notably video streaming, which are latency-limited both in absolute terms and the amplitude of variation of latency.

The description of L4S is included here largely for completeness. In measuring the take up of ECN here we are using conventional browser web object retrieval traffic, which apparently falls into the category of Classic ECN use, rather than L4S.

Measuring ECN

We are using the ad-based measurement approach, as used in many other APNIC Labs measurement experiments. The client-side application in this case is the browser, so we are not in a position to measure the uptake of L4S, as that approach appears to be mainly used in the context of video streamers and related applications where low latency is a highly desired attribute, and operated across limited network paths which are provisioned with dual queues and AQM.

The interaction we are capturing here is one where the initial TCP packet contains the SYN, ECN-Echo and CWR flags all set. The experiment's servers are configured to respond to this with a SYN-ACK packet with the ECN-Echo bit set.

At this point all subsequent IP packets in this TCP session should have the IP ECT(0) (or IP-ECT(1)) bit set in the IP header, as long as the TCP payload size is non-zero.

The first metric we are looking at with ECN adoption is the proportion of tested clients who send TCP SYN packets with the ECN flag bits set ("ECN-Clients"). The internet-wide metric for TCP ECN-clients is shown in Figure 7. Over the past 6 months of the experiment the ECN-Client rate across the public Internet is relatively small, between 2% and 3% of end clients.

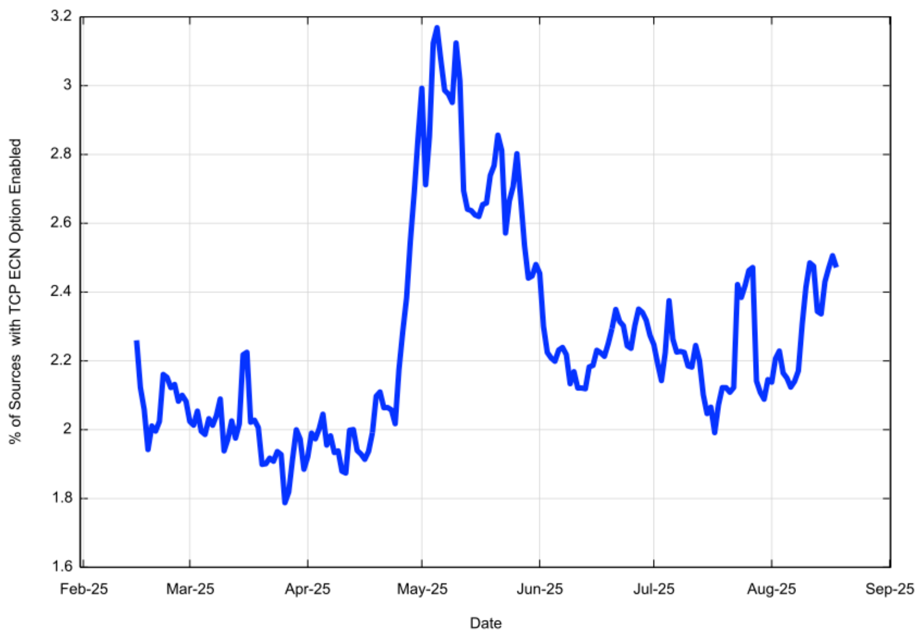


Figure 7 – TCP ECN Option Rate – Feb 2025 to August 2025

This is in many respects an odd outcome, in that it is consistently small! If this setting of the TCP options was a default behaviour of the Chrome browser hosted on an Android platform, then we would expect a take-up rate of around 60% or thereabouts of sample points, in line with the global uptake of this browser platform and the Android OS. If this was a default behaviour of Apple platforms, then we might expect to see a take-up rate of 15% to 20%, again roughly aligned to Apple's market share. Again, this is not what we observe here, despite reports that Apple has enabled TCP ECN in iOS and MacOS from some years now.

A possible reason for this low ECN result is that some edge environments, either the equipment used in consumer edge networks or in carrier edge equipment used in mobile networks, is performing scrubbing of the ECN option bits in outgoing TCP sessions. It's feasible that this may be part of an edge-based security firewall function where these additional TCP options are seen as spurious by the firewall. It's also feasible that IPv4 NAT gateways might be performing scrubbing of the TCP ECN settings. Is there any difference in the ECN bleaching rate between IPv4 and IPv6?

The distribution of visible ECN-Client capable hosts is not uniform across the Internet. If we look at the distribution of ECN-Client hosts on a country basis (Figure 8) it's clear that the distribution is quite uneven.

The list of the ten countries with the highest TCP ECN option rate is shown in Table 1. Note in this figure the colour scale is used for the map is a spectrum where red is 0%, yellow is 2% and green is 4%. Using a conventional 0% – 100% scale the entire map would be coloured red!

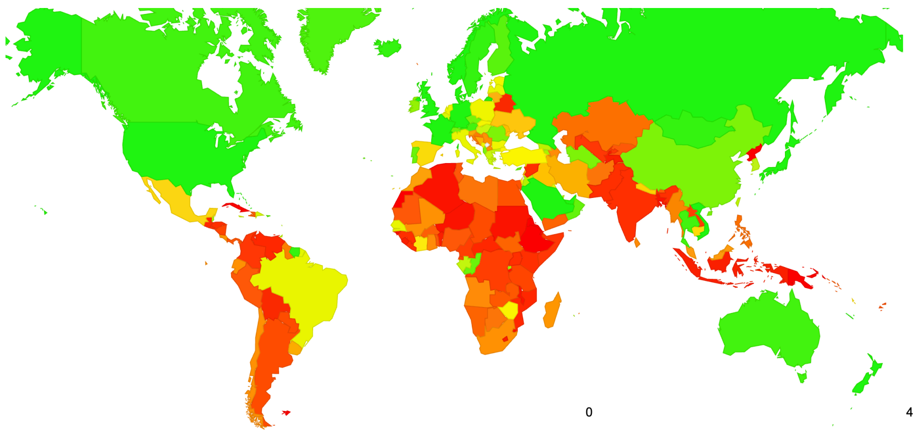


Figure 8 – Distribution of TCP ECN Option capability

Rank	CC	ECN Rate	Samples	Name
1	DE	6.9%	13,619,980	Germany
2	JP	6.4%	32,415,561	Japan
3	GB	5.3%	14,433,879	UK
4	US	5.3%	118,316,538	USA
5	BS	5.3%	105,261	Bahamas
6	AE	4.9%	2,457,698	United Arab Emirates
7	NZ	4.8%	1,002,670	New Zealand
8	BH	4.8%	300,563	Bahrain
9	VN	4.5%	18,930,474	Vietnam
10	LU	4.2%	153,307	Luxembourg

Table 1 – Top 10 countries with highest rate of TCP ECN Option capability

The next ECN-related behaviour we are interested in is "ECN bleaching". We'd normally expect that a TCP session that starts with the TCP ECN bits set would then have the IP ECN bits set in all subsequent IP packets in that session, where there is a non-zero TCP payload. However, there appears to be deployed network equipment that clears these bits in IP packets. They may be within the client-side edge networks, on the boundary of the client and the service provider networks, or within a server provider's network. Our objective here is not to measure the relative use of ECN bleaching in each of these environments, but simply to identify the extent to which it is taking place.

The average bleaching rate for all samples was 3.57%. Let's look at this bleaching rate data on a country-by-country basis (Table 2).

Rank	CC	Bleaching Rate	TCP ECN	IP ECN	Name
1	BH	**69.9%**	14,308	4,309	Bahrain
2	US	**63.8%**	6,260,495	2,266,271	USA
3	LT	**62.1%**	9,334	3,538	Lithuania
4	EE	**56.5%**	5,979	2,598	Estonia
5	BJ	**54.1%**	1,756	806	Benin
6	IS	**49.9%**	3,462	1,734	Iceland
7	BZ	**49.6%**	2,968	1,497	Belize
8	IL	**48.7%**	57,874	29,708	Israel
9	SK	**48.6%**	12,923	6,644	Slovakia
10	MN	**48.1%**	27,804	14,426	Mongolia

Table 2 – Top 10 countries with highest rate of IP ECN bleaching

It is also likely that IP-level ECN bleaching occurs in the network's environment, and in looking at the 823 IP access service networks where we have collected an adequate number of samples, we can observe some networks that consistently perform ECN bleaching on all connections (Table 3), and some where very minor levels of bleaching were observed (Table 4).

Rank	AS	Bleach Rate	TCP ECN	IP ECN	Samples	TCP ECN Rate	AS Name
1	35228	**100.0%**	214,773	32	217,280	98.8%	O2BROADBAND, GB
2	18126	**100.0%**	105,288	21	795,733	13.2%	Chubu Telecommunications, JP
3	41164	**100.0%**	3,241	1	93,309	3.5%	GET Norway, NO
4	131934	**100.0%**	2,383	1	21,762	11.0%	ICC Corporation, JP
5	6389	**99.9%**	253,405	139	298,937	84.8%	BELLSOUTH-NET-BLK, US
6	20294	**99.9%**	1,047	1	454,452	0.2%	MTN, UG
7	48503	**99.8%**	2,016	4	745,077	0.3%	Tele2 Kazakhstan, KZ
8	12929	**99.7%**	1,476	5	94115	1.6%	NETCOM, NO
9	6855	**99.5%**	2,646	12	234557	1.1%	Slovak Telekom, SK
10	12400	**99.4%**	9,081	50	733,066	1.2%	PARTNER-AS, IL

Table 3 – Top 10 Access Networks with highest rates of IP ECN bleaching

And the "best" in terms of preserving the IP ECN setting are the networks shown in Table 4.

Rank	AS	Bleach Rate	TCP ECN	IP ECN	Samples	TCP ECN Rate	AS Name
814	7122	**0.2%**	2,037	2,032	8,2139	2.5%	MTS-ASN, CA
815	12912	**0.2%**	2,051	2,046	390,856	0.5%	TM, PL
816	268323	**0.2%**	1,273	1,270	56,595	2.2%	AZZA TELECOM, BR
817	44486	**0.2%**	1,010	1,008	1,862	54.2%	SYNLINQ, DE

818	40021	**0.2%**	1,011	1,009	175,043	0.6%	CONTABO, US
819	15377	**0.2%**	1,014	1,012	46,257	2.2%	FREGAT, UA
820	39608	**0.1%**	1,776	1,775	32,446	5.5%	LANETUA, UA
821	135905	**0.0%**	2,260	2,259	4,081	55.4%	VNPT, VN
822	396303	**0.0%**	12,085	12,082	12,798	94.4%	NATOLAB-LEGACY, US
823	13886	**0.0%**	1,043	1,043	1,403	74.3%	CLOUD-SOUTH, US

Table 4 – Top 10 Access Networks with lowest rates of IP ECN bleaching

Is there any difference in ECN behaviour between IPv4 and IPv6? On the 29th August we observed the following data for IPv4 and IPv6 (Table 5).

	V4	**V6**	**Total**
Clients	13,542,448	4,725,783	18,268,231
TCP ECN Option	217,050	183,115	400,165
TCP ECN Rate	1.6%	3.9%	2.2%
IP ECT	147,512	126,684	274,196
TCP to IP	68.0%	69.2%	68.5%
Bleaching Rate	32.0%	30.8%	31.5%

Table 5 – Comparison of RCN Bleaching between IPv4 and IPv6

Interestingly, the IPv6 TCP ECN options rate was twice as great as the IPv4 rate. But once the TCP session carries the ECN Options field the conversion to the IP ECN signals is similar at just under 70%, so the ECN bleaching rates are very similar.

Conclusions

What can we say about ECN in today's public Internet?

As a general-purpose method of signalling congestion within the network, ECN in the public Internet appears to suffer from the issues of extremely small deployment and an excessive level of network middleware interference. The days of a single vendor providing the entire suite of hardware and software for a networked environment, as was the case for Digital Equipment Corporation and their proprietary DECnet network, have long gone, and the prospects for a tight integration of network device signalling and network protocol behaviour have pretty much gone as well. Protocols in the general heterogenous Internet can rely on just two behaviours. As network queues become populated then the total elapsed transit time for packets will get longer, as the queue occupancy times get added to the total elapsed transit time. And when the queues are full then packets will inevitably get dropped.

I hope I've been sufficiently careful to distinguish between the general public Internet environment and more specialised environments when considering the role of ECN.

For example, in Ultra Ethernet (UE), used in the context of AI Data Centres, ECN is an important component of UE's) congestion control mechanism, where network devices mark packets to signal queue formation, allowing the sending endpoint to proactively reduce its transmission rate rather than allowing the queues build to the point of dropping packets. The objective is to signal the sender to trim its sending rate prior to queue overload, supporting the intended outcome in UE of a high-speed lossless local switching fabric based on Ethernet. Whether this specification that includes this profile of ECN becomes an integral part of

AI Data Centre engineering remains to be seen of course, but it is important to note that there are other contexts here ECN has been mentioned.

Despite the reclassification of the RFC specification from Experimental to Standards Track (RFC 3168), ECN is a technology that has not excited widespread deployment in the public Internet over the past twenty years, nor has it motivated a widespread effort to remove the current points of ECN bleaching in the general use public network.

We've done a lot of work to mitigate some of the side effects of this packet loss as a consequence of queueing behaviour on TCP performance without enrolling the active assistance of network elements through ECN signalling.

The first of these measures is *sender pacing*. This refers to the sender attempting to evenly spread the set of to-be-transmitted packets across the current RTT interval. It has a number of distinct benefits. Pacing is not susceptible to various form of ACK compression or ACK thinning. The reduced reliance on network queues to smooth out sender-side burstiness reduces overall queue requirements which in turn can lead to lower network latencies. An approach, useful for video, is to pace at roughly the intended video rate, again removing the behaviour of "pulsing " the network with sharp bursts of traffic.

Given these benefits, you might wonder why sender-side pacing wasn't adopted as a default transport behaviour from the outset? I guess that the rationale was in the desire for transport protocols in end systems to be as simple as possible, and make the network perform the role of rater adaptation and pacing traffic flows. If the sender is passing data into the network path at a rate that is greater than the path capacity, then a queue will form at the bottleneck point, and that queue will impose a pacing discipline on the traffic at the bottleneck link that is exactly matched to the available capacity of that link. Subsequent experience has shown that reliance on network buffers to perform rate adaptation and pacing leads to the addition of noise to the implicit timing signals contained the ack-pacing behaviour of the transport protocol, which tends to impair the efficiency of the transport protocol. Current thinking on this topic can be summarized by the term "buffer bloat." Over-provisioning of buffers in networks leads to the formation of standing queues which add to latency and reduce the effectiveness of the feedback signals that are contained in the ACK stream. Under-provisioning of these buffers leads to premature packet loss and over-reaction by the transport protocol, leaving unused network capacity within the network. Given that optimal buffer dimensioning depends not only on the delay bandwidth product of the driven link but the number of simultaneous transport connections using that link, there is no statically defined optimal buffer size. Sender pacing relieves the pressure on the network buffers and mitigates their potential impact in terms of impairing transport efficiency. Enable sender pacing!

Secondly, we've improved the behaviour of TCP on packet drop. When a packet is dropped then "Classic" TCP will acknowledge subsequent packets by sending the sequence of the highest in-order byte received so far as an acknowledgement. These are *duplicate ACKs*. When a sender receives a duplicate ACK, it performs its congestion response, which, in classic TCP, involves a short sending pause, followed by a halving of the sending rate. When a sender receives three of more duplicates ACKs in sequence it performs a full session restart, reverting to an initial sending window and performing a *slow start* process to restart the congestion-sensing process. The improvement here is Selective Acknowledgement (SACK) (RFC 2018), which addresses many of these issues by allowing the TCP session to repair multiple lost packets in the same RTT interval without necessarily dropping the sessnion's sending rate. In environments where non-congestion loss is a factor, such as radio networks, this is a crucial behaviour that prevents a TCP session from collapse in lossy environments.

Finally, there is the ongoing work on delay-based TCP congestion control protocols. The early work on delay-based protocols lead to disappointing outcomes as delay-based protocols tend to defer to loss-based systems as the loss-based systems are far more tolerant of delay variation caused by queue formation. The approach of BBR was to deliberately stress the network path (by lifting the sending rate by some 25% during 1 RTT and looking at the delay impact of this transient burst). In terms of achieving high network utilisation efficiency without queue buildup the original BBR protocol was very successful. Its problem was that it was not overly fair to other concurrent sessions!

In summary, where does ECN fit in this picture of TCP congestion control on the public Internet? Frankly, I'm not very optimistic about ECN's prospects. Much of today's technology environment has been focussed on reducing external dependencies, rather than increasing them. ECN relies on the opposite of this trend, as it requires a tight interdependence between active network elements and end-to-end protocol behaviour to produce a lossless congestion control behaviour for TCP.

APNIC's daily measurement of ECN support on the public Internet can be found at https://stats.labs.apnic.net/ecn.

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

Author

Geoff Huston AM, M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net