The Internet Protocol Journal

May 2020 Volume 23, Number 1

A Quarterly Technical Publication for Internet and Intranet Professionals

In This Issue

You can download IPJ back issues and find subscription information at:
www.protocoljournal.org

ISSN 1944-1134

From The Editor

In mid-February, I traveled to Melbourne, Australia, to attend the *Asia Pacific Regional Internet Conference on Operational Technologies* (APRICOT). I normally attend around 10 or 12 similar Internet-related events in a year, be they *Network Operator Group* (NOG) conferences, *Regional Internet Registry* (RIR) events, or meetings of *The Internet Engineering Task Force* (IETF). This year, most of these events have either been cancelled or have "gone virtual" as the world tackles the COVID-19 pandemic.

The pandemic has clearly demonstrated the resilience and flexibility of the Internet and the people and organizations that rely on it for work, education, and entertainment. The various lock-downs or shelter-in-place orders have also given many of us an opportunity to take a closer look at some of the underlying technologies of the Internet, as we participate in online events or perhaps read more books and articles. This journal continues to receive many interesting articles on all aspects of networking, and in addition to the normal issues in print (and PDF format), we are also planning to expand our online presence in the near future.

Buffering is a central concept in packet-switched networks. Applications such as streaming audio or video rely on buffers to compensate for the fact that packets do not arrive at a fixed rate or even in a fixed order. Memory buffers are also used within the switches of the network to account for variations in network bandwidth and throughput. In our first article, Geoff Huston discusses network buffers and explains the numerous mechanisms that are used or have been proposed to tackle network congestion.

Previous articles in this journal have discussed various aspects of unsolicited e-mail, commonly referred to as "spam." This time, John Levine explains recent developments in anti-spam efforts, specifically *Domain-based Message Authentication, Reporting & Conformance* (DMARC) and *Authenticated Received Chain* (ARC).

As always, we welcome your feedback and suggestions on anything you read in this journal. Letters to the Editor may be edited for clarity and length and can be sent to **ipj@protocoljournal.org**. Please make sure your subscription details are accurate.

—Ole J. Jacobsen, Editor and Publisher
ole@protocoljournal.org

What's the Right Network Buffer Size?

by Geoff Huston, APNIC

Packet-switched networks need to use memory buffers within the switches of the network. In a simple example, if two packets arrive at a switch at the same time and are destined to the same output port, then one packet needs to wait in a local buffer while the other packet is sent on, assuming that the switch does not want to needlessly discard packets. Not only do these buffers address such timing issues that are associated with multiplexing, they are also useful in smoothing packet bursts and performing rate adaptation that is necessary when packet sources are self-clocked. However, there is a question that has never been answered satisfactorily: What's the "right" size for the memory buffer of a switch? If buffers are generally good and improve data throughput by reducing the incidence of packet drop, then more (or larger) buffers are better, right? Not necessarily, because buffers also add additional delay to packet transit through the network if the packet gets parked into one of more buffers in transit. If you want to provide a low-jitter packet-transit service, then deep buffers in the network are decidedly unfriendly! The result is the rather enigmatic observation that network buffers have to be as big as they need to be, but no bigger.

Buffers in a packet-switched communication network serve at least two purposes. They impose some order on the highly erratic instantaneous packet rates that are inherent when many diverse packet flows are multiplexed into a single common transmission system, and they compensate for the propagation delay inherent in any congestion feedback control signal and the consequent coarseness of response to congestion events by end systems.

The Internet adds an additional dimension to this topic. Most Internet traffic is still controlled by the rate adaptation that various forms of *Transmission Control Protocol* (TCP) congestion-control algorithms use. The overall objective of these rate-control mechanisms is to make *efficient use of the network*, such that no network resource is idle when there is latent demand for more resources, and *fair use of the network*, such that if the network has multiple flows, then each flow will be given a proportionate share of the network resources, relative to the competing resource demands from other flows.

The study of buffer sizing is not one that occurs in isolation. The related areas of study encompass various forms of queueing disciplines that these network elements use, the congestion-control protocols that data senders use, and the mix of traffic on the network. The area also encompasses considerations of hardware design; *Application-Specific Integrated Circuit* (ASIC) chip layouts; and the speed, cost, and power requirements of switch hardware.

The topic of buffer sizing was the subject of a workshop at Stanford University in early December 2019. The workshop drew together academics, researchers, vendors, and operators to look at this topic from their perspectives. It hosted 98 attendees from 12 countries, with 26 from academia and 72 from industry. The following are my notes from this highly stimulating workshop[0].

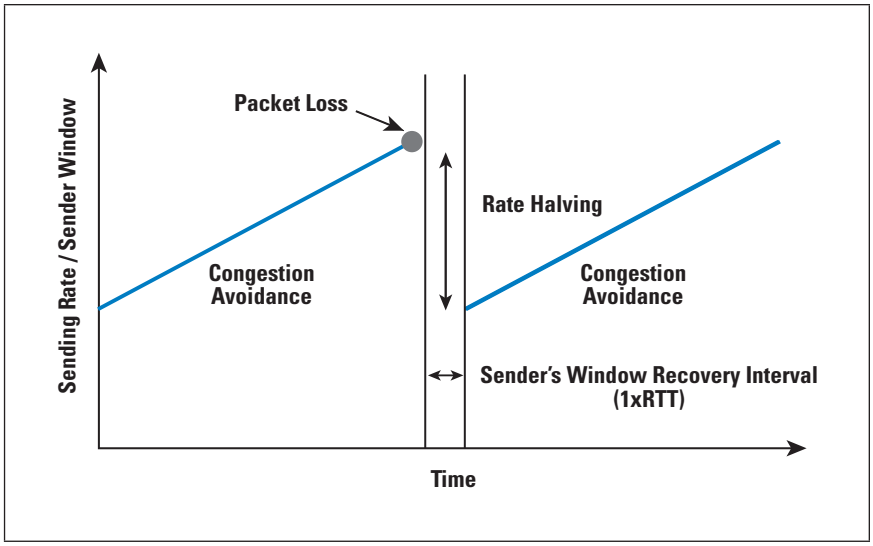
Background

In an autonomously managed packet network, packet senders learn from reflections of data that packet receivers provide, in a similar fashion to the way a radar system "learns" from a reflected signal. In a reliable flow-controlled TCP session the receiver sends an ACK packet to acknowledge the receipt of one or more data packets. Each ACK describes how many in-sequence bytes the receiver removed from the network. The sender can use this ACK signal to guide the injection of additional data into the network. One aim of each packet sender is a position of *stability*, where every packet passed into the network is matched against a packet leaving the network. In the TCP context, this behaviour is termed *ACK Pacing*.

While the sender can use ACK pacing to determine a stable sending rate, it cannot readily determine a *fair* and *efficient* sending rate. The unknown factor in this model is that the sender is not aware of the right amount of network resources to claim for the data transaction that would sustain a *fair* and *efficient* outcome. The TCP approach to solve this problem is to use a process of *dynamic discovery* where the sender probes the network by gently increasing sending rates until it receives an indication that the sending rate is too high. It then backs off its sending rate to a point that it believes is lower than the sustainable maximum rate and resumes the probe activity[1].

This classic model of TCP flow management is termed *Additive Increase, Multiplicative Decrease* (AIMD). The sending rate is increased by a constant amount over each time interval (usually the time to send a packet to the receiver and the receiver to send an acknowledgement packet back to the sender, or a *Round Trip Time* (RTT) interval). In response to a packet-loss event, indicated by *Duplicate ACKs* that suggest the next in-sequence packet has been lost and the receiver considers successive packets to be out of order, the sender decreases the sending rate by a multiplicative ratio. The classic model of TCP uses an additive factor of 1 TCP *Message Segment Size* (MSS)[9] of data per RTT and a rate halving (divide by 2) in response to a packet loss. The result is a "sawtooth" TCP behaviour[2] (Figure 1). This control is determined by the sender maintaining a *Congestion Window* value, which is the maximum amount of unacknowledged data that the sender can have. Increasing the sending rate is achieved by increasing this value, and a decrease is achieved by reducing this value. When the value is reduced, then the sender must wait for the amount of unacknowledged data to drop below the new value before sending new data into the network.

Figure 1: TCP AIMD Congestion-Control Behaviour



We can mathematically model this behaviour of rate halving in response to packet loss and linear increase otherwise. If the packet-loss function is assumed to be a random loss function with a probability p, then the data-flow rate is proportional to the inverse square root of the packet-loss probability, as given in following equation (0)[2]:

$$BW = \frac{MSS}{RTT} \cdot \frac{C}{\sqrt{p}} \quad \text{where } C = \sqrt{\frac{3}{2}} \qquad (0)$$

This result implies that the achievable capacity of an AIMD TCP flow is inversely proportional to the square root of the packet-loss probability.

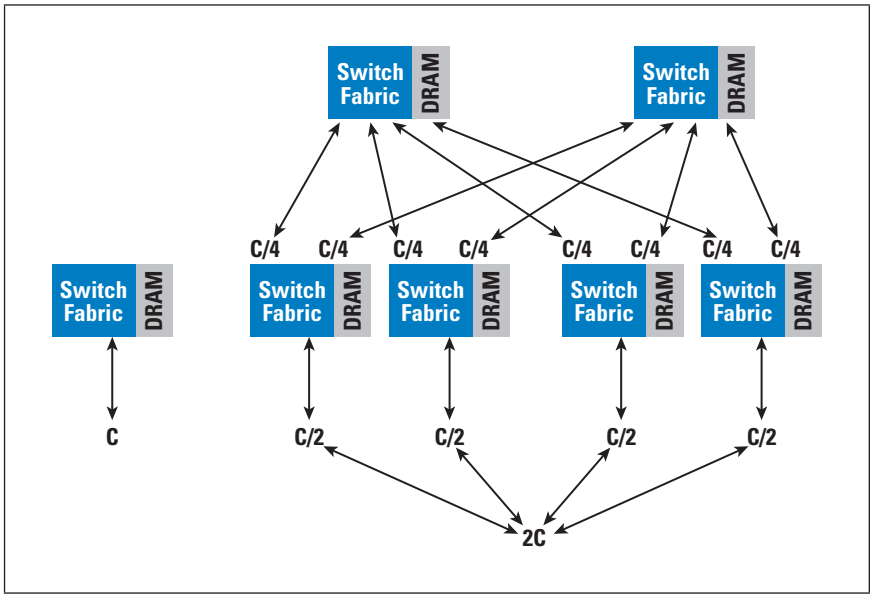
But packet loss is not a random event. If we assume that packet loss is the result of buffer overflow, then we also need to consider buffers and buffer depth in more detail. An informal standard for the Internet is that the buffer size should be equal to the delay-bandwidth product of the link (the derivation of this "rule of thumb" result is explained in the next section).

$$Size = MSS \cdot RTT \qquad (1)$$

As network link speeds increase, the associated buffers similarly need to increase in size, based on this engineering rule of thumb. The rapid progression of transmission systems from megabits per second to gigabits per second and the prospect of moving to terabit systems in the near future pose, particular scaling issues for silicon-based switching and buffer systems. As networks increase in scale, the switching scaling factors tend to show multiplicative properties.

For example, if we have a single switch of capacity C and we want to double the effective switching capacity but cannot increase the capacity of the switching chip, then how many switching chips will we need to produce a composite switch of capacity 2C? The answer is not 2 but 6, as shown in Figure 2. A packet will also need to traverse up to three switch fabrics, so the aggregate buffer size of the path through the switch fabric may triple in size.

Figure 2: Doubling Switch Capacity



Self-clocking packet sources imply that congestion events within the network are inevitable, and any control mechanism that is imposed on these sources requires some form of *feedback* that allows the source to craft an efficient response to congestion events. However, this feedback is constrained by propagation delays and this lag creates some coarseness in the response mechanisms. If the response is too extreme, the sources will over-react to congestion and the network will head into instability with oscillations between periods of intense use and high packet loss and periods of idle operation. If the response is too small, the congestion events will extend over time, leading to protracted periods of operation with full buffers, high lag, and high packet loss.

Robust control algorithms need to be stable for general topologies with multiple constrained resources, and ways of achieving this stability are still the subject of investigation and experimentation. If feedback based on rate mismatch is available, then feedback based on queue size is not all that useful for stabilising long-lived flows. Feedback based on queue size is, however, important for clearing transient overloads.

Over more than three decades of experience with congestion-management systems, we have seen many theories, papers, and experiments.

Clearly, there is no general agreement on a preferred path to take with congestion-control systems. However, a consistent factor here is the network buffer size, and the sizing of these buffers in relation to network capacity. One view, possibly extreme, says that buffers are at the root of all performance issues.

The task of dimensioning buffers in a switching system has implications right down to the design of the ASIC that implements the switch fabric. On-chip memory can be fast, but it is limited in capacity to some 100 MB or less. Larger memory buffers need to be provisioned off-chip, requiring I/O logic to interface to the memory bank, and the speed of the off-chip system is typically slower than on-chip memory. Hybrid systems have to compromise between devoting chip capacity to switching, memory, and external memory interfaces. And layered on top of these design trade-offs is the continuing need to switch at higher speed across larger numbers of ports.

One objective of the Buffer Size Workshop was to continue the conversation about buffers, determine their relationship to congestion control, and improve our understanding about the interdependence among buffer size, queuing control, self-clocking algorithms, network dimensioning, and traffic profiles.

How Big Should a Buffer Be in the Internet?

The single AIMD flow model predicts poor outcomes for flows operating across buffers that are too deep or too shallow. Too deep and the flow's loss response of halving the congestion window does not clear the buffer, and a standing queue forms that contributes to an increased latency imposed in the flow (Figure 3).

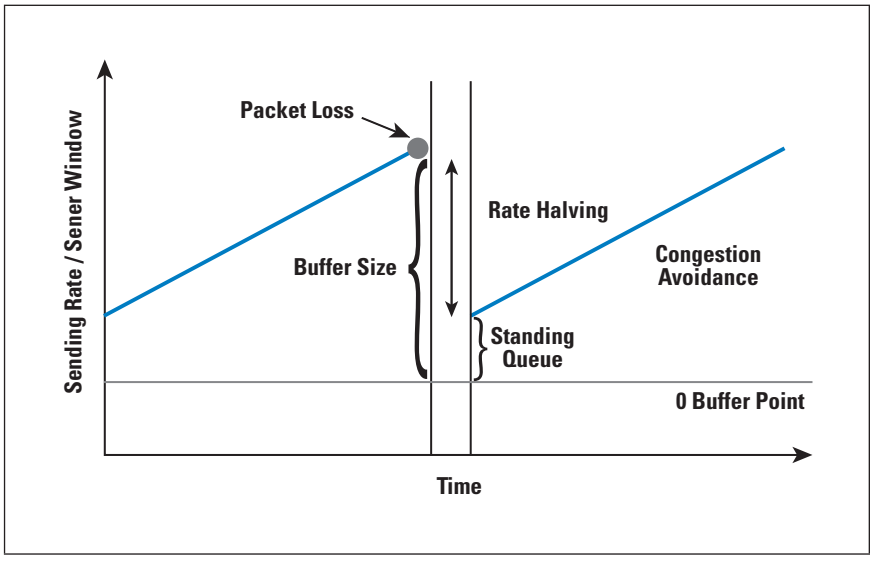
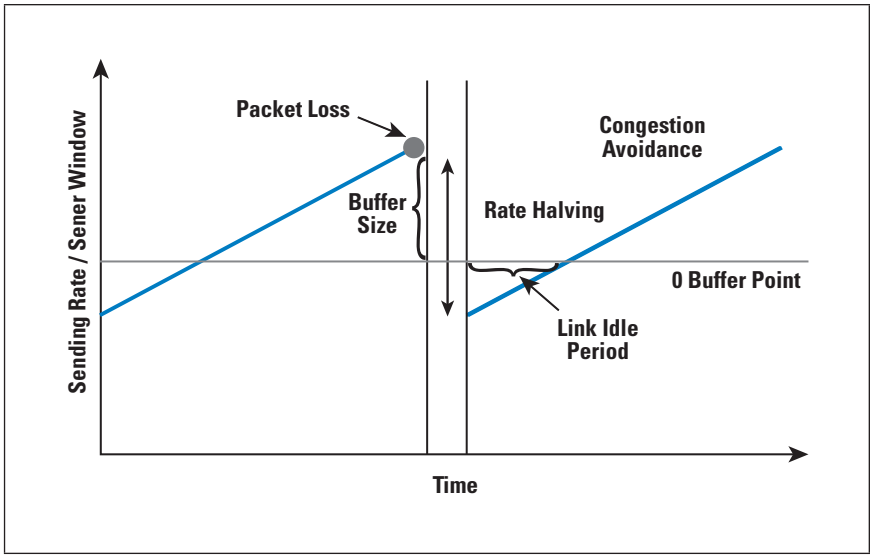


Figure 3: Deep Buffers and Rate Halving Congestion Response

If the buffer is too shallow, then rate halving drops the sending rate below the bottleneck capacity, and the link will be under-used until the additive increase brings the rate up to the link capacity (Figure 4).

Figure 4: Shallow Buffers and Rate Halving Congestion Response



The delay-bandwidth product rule of thumb generates some extremely large queue-capacity requirements in medium-delay, high-capacity systems. A 10-Gbs system using a 100-ms RTT link requires a 125-MB memory pool per port that can read and write at 10 Gbps. A 1-Tbps system would require a 12.5-GB memory pool per port that can read and write at 1 Tbps. A 16-port switch would require 200 GB of high-speed buffer memory using this same design guideline.

Such numbers are challenging for switch designers, and it is reasonable to review the original work to understand the derivation of this provisioning rule.

This model of provisioning the queue to the bandwidth-delay product is derived from a AIMD control algorithm of a single flow using an additive value of 1 segment per RTT and halving the congestion window on packet loss, coupled with the objective of using the buffer to keep the link busy during the period of window deflation.

However, something a little deeper in the oscillation of the AIMD flow-control process affects the selection of the buffer size. In the purely hypothetical situation of a single flow operating across a single switch with a lossless transmission medium, the only source of packet loss is buffer exhaustion. If the switch has no buffer at all, then the AIMD algorithm will operate in a steady state between half capacity and full capacity, leaving approximately one-fourth of the capacity unused by the flow.

The objective is to use a buffer as a reservoir to fill the transmission link while the sender pauses, waiting for the receiver's count of unacknowledged data to fall below the new congestion-window value. If the buffer size is set to the link bandwidth times the link RTT, then the buffer will be drained at the point when the sender's unacknowledged data reaches the congestion-window value and the sender can resume sending.

While this result is from a theoretical analysis of a single flow through a single link, experiments by Villamizar and Song in 1994[3] pointed to a more general use of this dimensioning guideline in the case of multiple flows across multiple links. The rationale for this experimental observation was a supposition that synchronisation occurs across the dominant TCP flows, and the aggregate behaviour of the elemental flows was similar to a single large flow. This work was the foundation of today's common assumption that buffers in the network should be provisioned at a size equal to the round-trip delay multiplied by the capacity in order to ensure efficient loading of the link; see equation (1).

This supposition has been subsequently questioned. The scenario of a link loaded with a diversity of flows in RTT, duration, and burst profiles implies that synchronisation across such flows is highly unlikely, obviously having implications for buffer-size calculation. If there are two concurrent TCP flows, they have the same RTT, and they resonate in the increase and decrease events, then the buffer requirement will be the same for an efficient use of the network and a fair sharing of the available bandwidth. But if the increase and decrease of the two sessions are exactly out of phase, then a fair and efficient outcome would be created by a buffer size that is three-quarters of the original single flow. The real world typically sees a number of concurrent flows where both the RTT and the phase of the TCP duty cycle all vary. A Stanford TCP research group study in 2004[4] used the central-limit theorem to point to a radically smaller model of buffer size. You can maintain link efficiency for N desynchronised flows with a buffer that is dimensioned to the size of:

$$Size = \frac{BW \cdot RTT}{\sqrt{N}} \qquad (2)$$

This result is radical for high-speed extended latency links in a busy network. The consequences on router design are enormous: "For example, a 1 Tb/s ISP router carrying one TCP flow with an RTT_{min} of 100ms would require 12.5 GB of buffer and off-chip buffering. If it carries 100,000 flows, then the buffer can be safely reduced to less than 40MB, reducing the buffering and worst-case latency by 99.7%. With small buffers, the buffer would comfortably fit on a single chip switch ASIC."[5]

Queue Management

The default operation of a queue within a switch is to accept new packets while there is still space in the queue and discard all subsequently arriving packets until the output process has cleared space in the queue. If an incoming packet burst arrives at a switch and the queue capacity is insufficient to hold the burst, then the tail of the burst will be discarded. This tail-drop behaviour can compromise the performance of the flow, because the clocking information for the tail end of the burst has been lost.

One mitigation of this behaviour is *Active Queue Management* (AQM), where the process of queue formation triggers "early" drop. In other words, a packet drop will occur even when there is space in the queue to accept the packet. The ideal outcome of AQM is that packet drop in a large burst will occur inside the burst and the trailing packets following the dropped packet (which are not dropped as there is still space in the queue) will carry a coherent clocking signal in the ACK packet train that allows the flow to repair the loss quickly without losing the implicit clocking signal. Loss-based congestion-control algorithms will react to this packet drop by dropping their congestion-control window size, reducing their sending rate without collapsing the sending rate back to zero.

Drop-based TCP control algorithms react predictably to packet loss. However, the Internet is not entirely homogenous with respect to flow-control algorithms, and we are seeing increasing interest in flow systems that account for variance of the RTT measurements in a flow, or so-called *delay-based* TCP control systems. Delay-based paced control algorithms react differently to queue drop, and a "pure" delay-based flow-control system is indifferent to a loss signal. The question is: Are there AQM functions that can support a mix of congestion-control algorithms? Indeed, is the question of what form of AQM to use a more important question than the size of the underlying buffer?

Explicit Network Feedback

For many years there has been considerable debate between an end-system approach that uses *only* the received ACK stream to infer the network congestion state in the data forwarding direction from a packet loss signal (Figure 5), and an approach that uses some form of *explicit* signalling from the network that can directly inform the source of the network state. Very early efforts in such direct signalling through *Internet Control Message Protocol* (ICMP) *Source Quench* messages were quickly discounted because of the various issues related to its potential for *Denial-of-Service* (DoS) attacks and its inability to authenticate the messages.

The *Explicit Congestion Notification* (ECN) proposal[6] tried to address the most obvious failing of the earlier approach by placing the congestion signal inside the end-to-end IP packet exchange. Switching elements that were experiencing the onset of local congestion load in their buffers were expected to set a *Congestion Experienced* bit in the IP packet header of packets that were contributing to this load condition. Receivers were expected to translate this bit into the ACK packet header, so that the sender received an explicit congestion signal rather than having to infer congestion from an ACK signal that reflects packet loss (Figure 6).

Figure 5: Loss-Based Congestion-Control Behaviour

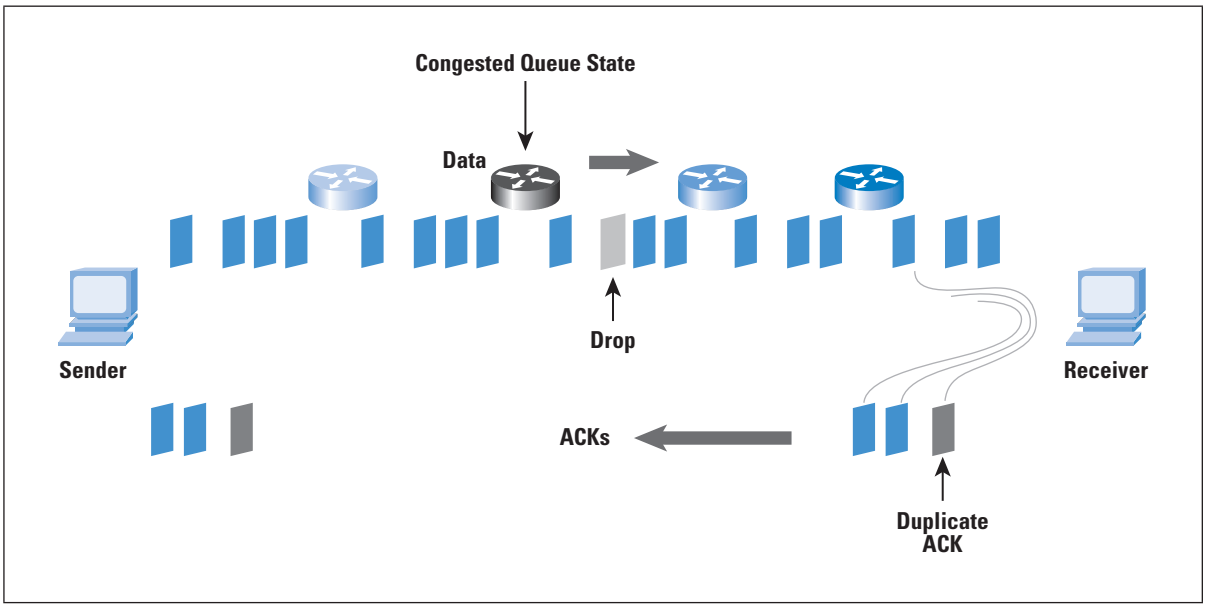
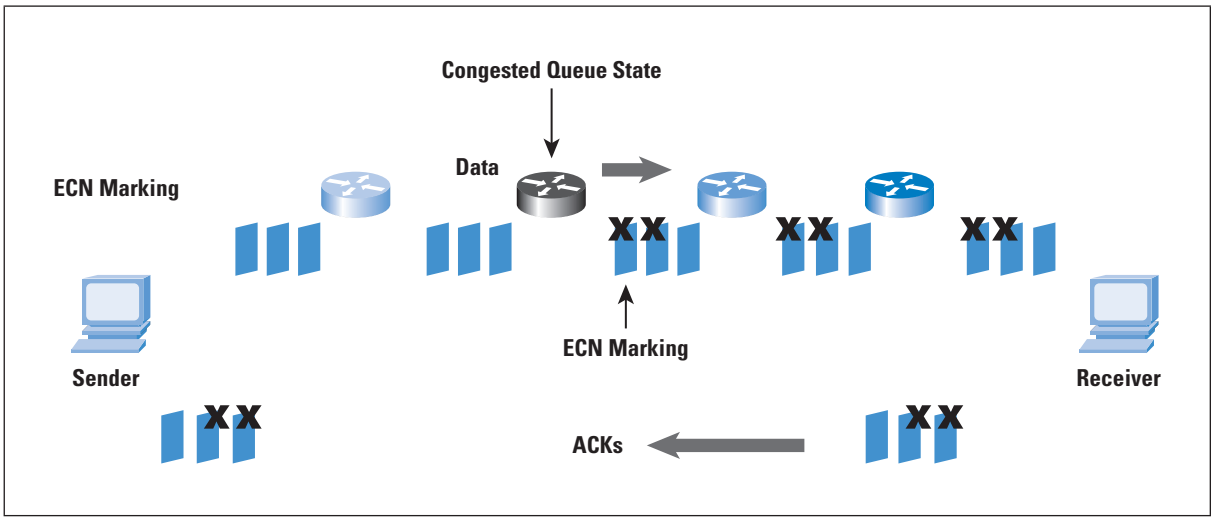


Figure 6: ECN Marking



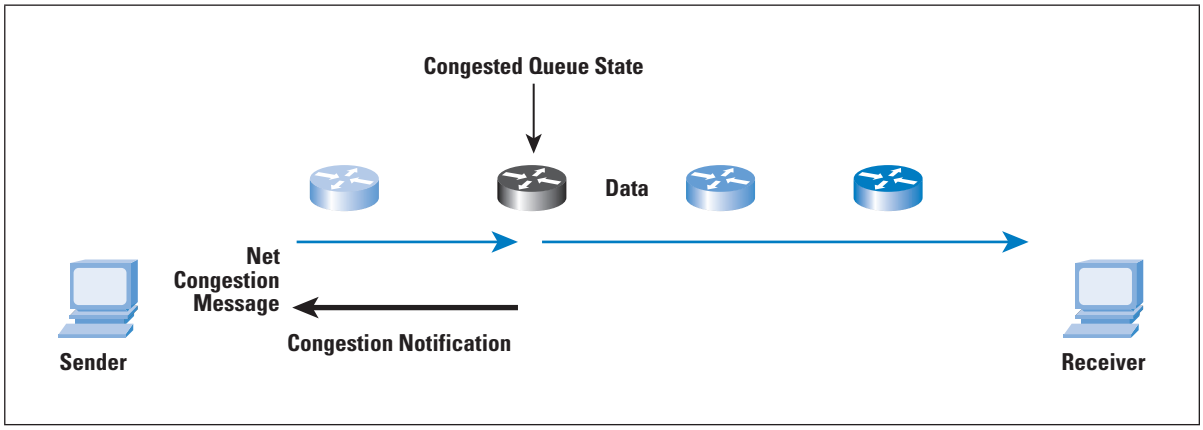
The advantage of ECN is that the sender is not placed in the position of being informed of a congestion condition well after the condition has occurred. Explicit notification allows the sender to be informed of a condition as it is forming, so that it can take action while there is still a coherent ACK pacing signal coming back from the receiver (that is, before packet loss occurs). This measure mimics the intention of delay-based flow systems, but with increased precision assuming that all switches were to perform this congestion marking.

However, ECN is only a single bit marking. Is that enough? Would a richer marking framework facilitate a more precise sender response? What if we had a marking regime that marks based on the distance from the current rate to a desired fair-efficient rate? Or use a larger vector to record the congestion state in multiple queues on the path?

The conclusion from one presentation is that the single-bit marking, while coarse and non-specific, is probably sufficient to moderate self-clocking TCP flows such that they do not place pressure on network buffers, leaving the buffers to deal with short-term bursts from unconstrained sources.

Another presentation at the workshop explored a network-level direct-feedback message, analogous to the ICMP *Packet Too Big* messages in *Path MTU Discovery* (PMTUD). To short-circuit the delays associated with completing the entire round trip, this approach envisages the switch experiencing the onset of congestion to explicitly message the source of this congestion condition (Figure 7).

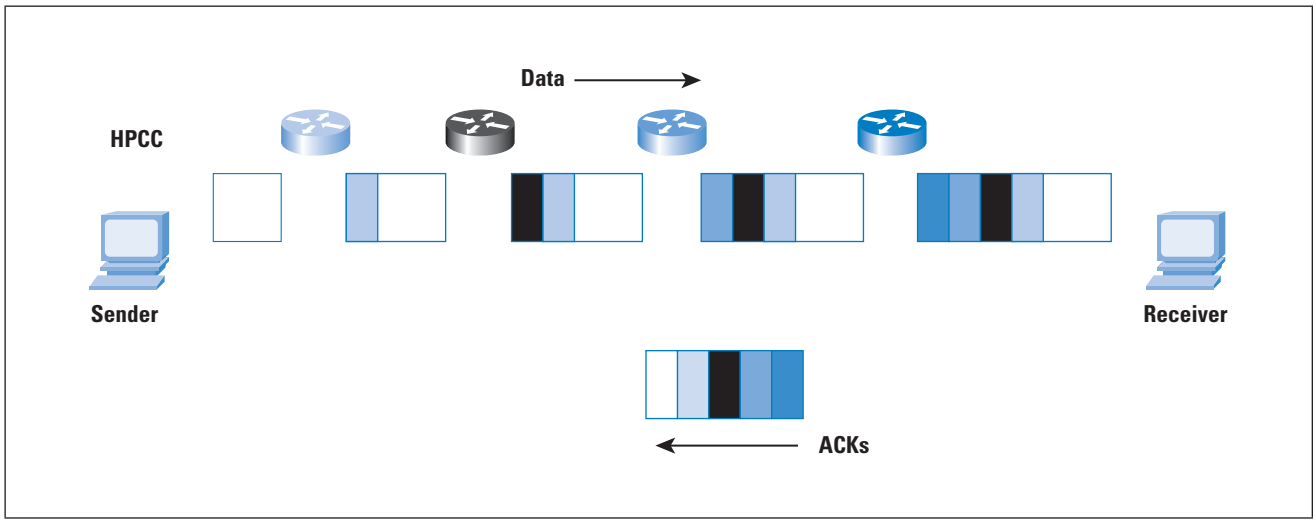
Figure 7: Network-Congestion Signalling



Another presentation looked at the attachment of a detailed telemetry log to each packet in a data-centre application. In the *High-Precision Congestion Control* (HPCC) framework each switch attaches the time, queue length, byte count, and link bandwidth to the data packet. The receiver takes this data and attaches it to the corresponding ACK, so that the sender can form a detailed model of the recent state of path capability. HPCC allows the sender to calculate a fair sending rate and then rapidly converge to this rate, while at the same time bounding the formatting of queues and bounding queuing delays. The domain of application of this approach appears to be the data centre, and the objective is to achieve high speed with bounded delay for *Remote Direct Memory Access* (RDMA)-style applications (Figure 8).

There is a degree of debate between *congestion-based* TCP control and delay-based mechanisms. On the one hand, we hear that delay-based mechanisms can operate the flows at the onset of queue formation in the network. On the other hand, we hear that attempting to set the flow to a fixed delay and operating with fairness to other flows is intrinsically impossible and that we need to operate flows with congestion moderation.

Figure 8: High-Precision Congestion Control



Near and Far Buffers

What is the cost/power trade-off of buffers on-chip and off-chip? And if we are considering off-chip, what do we actually mean, because there are different implementation approaches to off-chip memory. As a general observation, the performance of off-chip memory is not remotely close to what is required by a high-capacity, high-speed switch. This performance is not improving over time because memory speed is not scaling at the same rate as transmission or switch speeds, so the gap in performance between transmission and switching and memory speed is only getting larger over time.

One switch chip fabricator, Broadcom, implements both deep and shallow buffers. On its switch fabric chip Broadcom uses small, fast buffers and wraps the switch fabric with everything it can to reduce the dependency on deep buffers.

Recent operational data at Intel suggests that shallow buffers may be "good enough," but because of limitations in instrumenting technologies there is insufficient confidence in these results to allow switch chip designs to completely discount external memory interfaces and a local cache and use on-chip memory exclusively. Current switch designs use between 10% and 50% of chip area on memory management. This observation applies to high-capacity, high-speed switches, because at lower capacity and lower speeds there is no such constraint and you can use large pools of off-chip memory (relative to transmission speeds), although some constraint in the amount of memory will likely produce a better outcome in these contexts as well.

The question of future requirements is always present in chip design, given the long times between phrasing requirements and deployment into networks. Where is this situation heading? Memory buffers are not growing as fast as chip bandwidth. Clock speeds are not increasing, and scaling chip bandwidth is currently achieving parallelism rather than increasing the chip clock speed.

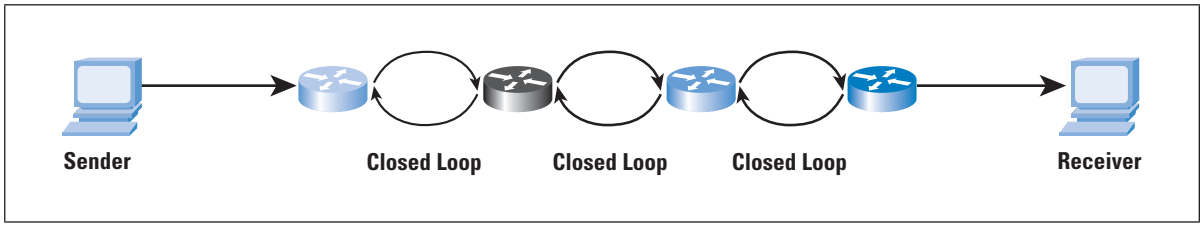
While doubling switch capacity may be feasible, contemplating an increase in capacity by factors of 20, 50, or even 100 seem like particularly tough challenges.

Today packet rates are typically achieved with multiple parallel pipelines, and orchestrating such highly parallel mechanisms creates its own complexity in design. No one is yet prepared to call an end to the prodigious outcomes of *Moore's Law* in the semiconductor realm, but it looks like clock speeds are not keeping up, and pin density and even increasing gate density are becoming challenging. Is doubling the number of ports on a switch chip good enough? If a chip has twice the switch bandwidth, does it need twice the on-chip memory capacity? Or less? The answer lies in external factors such as congestion-control algorithms, queue-management disciplines, and delay management.

Hop-by-Hop Flow Control

Hop-by-Hop Flow Control represents a revival of a very early attribute of packet-switched networks, where an end-to-end path is composed of a sequence of flow-controlled hops. Each switching element sends at its line rate into the buffer of the next switch. When a queue forms at the receiver, the hop flow control can pause the flow coming from the adjacent switch and resume it when the queue is cleared. Yes, this process sounds very reminiscent of X.25 and the *Digital Data Communications Message Protocol* (DDCMP) component of DECnet, and it's the opposite of the intent of the end-to-end approach. However, the approach can produce direct back pressure on a bursting source with no packet drop and yield highly efficient use without extensive buffer-induced delays. Essentially the self-clocking nature of the flow is replaced with a network clocking function (Figure 9). Admittedly, this approach is not universally applicable, and it appears to offer a potential match to the intra-data-centre environment where traffic patterns are highly bursty, propagation times are low, paths are short, and volumes and speeds are intense.

Figure 9: Hop-by-Hop Flow Control



Flow-Aware Buffer Management

It appears that the move towards shorter buffers relative to the link speed is inevitable. But how to manage the feedback systems to allow self-clocking data sources to adjust to the shortened buffer space is still an outstanding issue.

One approach starts with a basic traffic characterisation of a relatively small proportion of "elephant flows" (high volume, long duration) mixed with a far higher count of "mice" flows (low volume, short duration). While elephant flows are highly susceptible to congestion signalling, mice flows are not.

If the network could classify all currently active flows into either elephants or mice, then the network could use different queuing regimes for each traffic class. This sorting adds to the cost and complexity of packet switches, and if scaling pressures are a factor in switch design, then it's not clear that the additional cost of switch complexity would be offset by a far superior efficiency outcome in the switching function.

Assuming that such a flow classification could be achieved dynamically, we can consider differential responses. For short flows, there is little benefit to be gained by any form of explicit congestion control other than placing all such flows into their own queuing regime. For long-lived large flows, we could contemplate an explicit network-congestion signal. It could take the form of an explicit packet back to the network-generated source. The advantage of this approach is that the feedback of excessive sending rate is faster than a full RTT interval, allowing the sender to give a timely response. However, this idea does seem like a reprise of the ill-fated ICMP Source Quench message, and all that was problematical with ICMP Source Quench is probably still an issue in this form of network-congestion notification.

We can exploit this concept of the use of various queue regimes for different flow types in a different way by using a short buffer for long flows in the expectation that the implicit congestion signal of packet drop would allow the long-duration flow to stabilise into the available network resource, while short unregulated bursts could have access to a deeper buffer, allowing effective use of the buffer as a rate-adaptation tool to mitigate the burst.

This concept is taken even further in one project, which used the observation that if a buffer is too deep, then the flow-rate reduction following packet drop will leave a standing queue in the buffer, and if the buffer is too shallow, then the rate reduction will leave a period of an empty queue and an idle transmission system. This observation means that a flow-aware buffer manager could adjust its buffer size following observation of the post-reduction behaviour, reducing the buffer if standing queues form and increasing it if the queue is idle. It's an interesting approach to fair-queuing flow management, treating the per-flow buffer as an elastic resource that can resize itself to adapt to the congestion-management discipline of the flow.

ISP Network Buffer Profile

P4 is a language used to program the data plane of network devices. The language can express how a switch should process packets ("P4" itself comes from the original paper that introduced the language, *Programming Protocol-independent Packet Processors*[7].)

Barefoot's *Tofino* is an example of a new class of programmable Ethernet packet switches that are controlled through P4 constructs, and these units can currently handle aggregate capacity of some 12.8 Tbps of data-plane capacity. This capability allows for a measurement regime that can expose packet characteristics at a nanosecond level of granularity. By tapping the packet flow of a high-speed trunk transmission system into and out of a switching element in the network and attaching the taps to a P4 switch unit, it is possible to match the times of ingress and egress of individual packets and generate a per-packet record of queuing delay within the switching element at a nanosecond level of granularity.

This capability provides a new level of insight into burst behaviour in high-speed carriage systems *Internet Service Providers* (ISPs) use. The major observation from an exercise conducted on a large ISP network was that network buffers are lightly used except for "microbursts," bursts of some 100 microseconds or so, where the queue adds a delay element of more than 10 ms on a 10 GigE port. Further analysis reveals an estimate of packet drop rates if the network buffers were reduced in size, and for this case the analysis revealed that an 18-msec buffer would be able to sustain a packet drop rate of less than 0.005%.

If buffer-congestion behaviours in such ISP networks are, in fact, microbursts, then network measurement tools that operate at the per-minute or even at the per-second level of granularity are simply too crude. P4-based measurements that can resolve behaviours at the nanosecond level offer new insights into buffer behaviours in networks that carry a large volume of diverse flows. Even though the per-flow control cycle of the data-plane flows is of the scale of some milliseconds and longer, the microburst behaviour is that of a load model that exhibits sub-millisecond burstiness. The timescale of end-to-end congestion control operates at a far coarser level than the observed behaviour of congestion within a switch running a conventional traffic load.

This discussion leads to the observation that large-scale systems are creating extremely rapid queue size fluctuations, and it is unrealistic to expect that end-to-end control algorithms can control the queue size. It might be that at best these control algorithms can contribute to influencing the distribution of queue sizes.

Sender Pacing

The Internet can be seen as a process of statistical multiplexing of a collection of self-clocked packet flows where the flows exhibit a high degree of variance and a low level of stability. The reaction to this unconstrained input condition so far is to use large buffers that can absorb the variations in traffic. How large is "large enough" becomes the critical question in such an environment. The work on buffer sizing as being in proportion to the bandwidth-delay product of the transmission elements is an outcome of a process that measures the properties of the control algorithm for traffic flows.

It then derives estimates of buffer sizes that should be capable of carrying such a volume of traffic that it will efficiently and fairly load the transmission system.

The exercise assumes that the buffer dimension is a free parameter in network design, and control algorithms are fixed. Buffer speed inside the network has to double at a cycle of some 2 years, and the buffer size has to double in a similar timeframe. The product of size and speed is a quadrupling every 2 years. The current tactical response to this escalation of buffer requirements due to transmission capacity increases has been to reduce the size of the buffers relative to the transmission capacity. However, this response is not a long-term sustainable solution because such under-provisioned network buffers will impair overall network efficiency in these self-clocking flow regimes.

The future prospects for self-clocked traffic flows are not looking all that bright given that the growth demands for network buffer-based mitigation of unconstrained sender behaviours appears to be in excess of what can be satisfied within constraints of constant unit cost of network infrastructure. Without overall economies of scale where larger service-delivery systems achieve lower unit costs of service delivery, the management of traffic and content assumes a different trajectory that tends to drive towards greater distribution and dispersal rather than continued aggregation and amalgamation. For the large hyper-scaled content enterprises in today's Internet, this outcome is certainly not optimal.

It is a potentially fruitful thought process to consider this topic from an inverted perspective and look at the desirable control-algorithm behaviour that efficiently uses the network transmission resources when the available buffering is highly restricted. This thought process leads to the consideration of "pacing," where the server uses high-precision timers to smooth data flows as they leave the server, attempting to create a stable traffic flow that matches bottleneck capacity on the path. The more accurate this estimation of bottleneck bandwidth, the lower the demand for buffer capacity due to burst adaptation. Residual buffer demand is presumably based on the demands of statistical multiplexing of disjoint flows. Given that the senders are under the control of the service-delivery platforms and there are orders of magnitude fewer high-volume senders than receivers, this form of change is actually far less than the change required by, say, the IPv6 transition.

It is this thinking that lies behind the *Bottleneck Bandwidth and Round-trip Propagation Time* (BBR) protocol work. The sender's flow-control algorithm generates an estimate of the bottleneck bandwidth and the minimum RTT interval, and then paces packet delivery so as to feed traffic into the bottleneck at exactly the bottleneck capacity, which should not involve the formation of a queue at the bottleneck.

The BBR control algorithm periodically probes up to revise its previous bandwidth estimate, and probes down to revise its previous minimum RTT, and accounts for other congestion-formation signals, such as ECN. This probing up and down, or dithering, is not precisely specified in the core BBR algorithm, and these parameters are being revised in the light of deployment experience to determine dithering settings that are both efficient and fair. The expectation is that BBR will not drive the formation of standing queues in the network and will pace the flow at the maximal rate that the network path can fairly sustain.

However, BBR is not the only way to perform flow pacing, and a large number of outstanding questions remain. How does pacing at the sender affect the queue management at the edge close to the client? What are the cross impacts of burst traffic with pacing? How should a pacing-control algorithm react to packet loss? Or to out-of-order packet delivery? Can strict flow pinning still be required for *Equal Cost Multiple Path* (ECMP) routing or does pacing relax such requirements for strict path pinning? Are pacing or self-clocking the only options, or are there other approaches?

One perspective is that we are sitting between two constraint sets. Escalating volume and speed in the core parts of the network implies that bandwidth-delay product model buffer sizing is an unsustainable approach. The scaling back of buffer sizes in the network means that self-clocked protocols will potentially become more unstable and compromise achievable network efficiency and fairness. From this perspective sender pacing looks to be a promising direction to pursue.

Is There a Buffer Sizing Problem?

In the Internet we are currently seeing a diversity of responses to network provisioning. Some network operators use equipment with generous buffers. These buffers are overly generous according to the buffer-bloat argument. Other network operators field equipment with scant buffers that run the risk of starving data sources while leaving idle network capacity.

There is a mix of congestion-control algorithms (CUBIC, NewReno, BBR, *Low Extra Delay Background Transport* [LEDBAT], etc.) and a mix of queue-management regimes (*Controlled Delay* [CoDel], *Random Early Detection* [RED], *Weighted Random Early Detection* [WRED])[8]. A diversity of deployment environments exists, including mobile networks, Wi-Fi, wired access systems, LANs, and data centres. And there is a mix of parameters of the desired objective here, whether it is some form of fairness, loss, jitter, start-up speed, steady-state throughput, stability, efficiency, or any combination of these factors. It is little wonder that it's challenging to formulate a clear picture of common objectives and to determine what actions are needed to achieve whatever we might want!

There is the assumption that large network buffers absorb imprecision in clocking (timing "slop") and allow simpler coarse rate-control algorithms to operate effectively without needing high-precision tuning. Small network buffers provide little leeway and tolerance for such approximate approaches. This mistrust of the level of precision of control that end systems exercise is a pervasive view within the networking community, and it could even be characterized as an entrenched view. So entrenched is this view that probably no experimental result could convince the community as a whole that network buffers can be far smaller than they are today, all other factors being equal. This fact is true despite the overwhelming evidence that overly large buffers compromise network performance, a position that has been described as "buffer bloat." There are other reasons why large buffers are a problem for networks and users. As we scale up the size and speed of the network, large very-high-speed buffers are also increasing in cost. If we are going to admit compromises and trade-offs in network design, is reducing the relative size of the buffer an acceptable trade-off?

And if we want to reduce buffer size and maintain efficient and fair performance, how can we achieve it? One view is that sender pacing can remove much of the pressure on buffers, and self-clocking flows can stabilise without emitting transient bursts that buffers will need to absorb. Another view, one that does not necessarily contradict the first, is that the self-clocking algorithm can operate with higher precision if there were some form of feedback from the network on the state of the network path. This feedback can be as simple as a single bit (ECN) or a complete trace of path element queue state (HPCC).

This topic remains a rich area of unanswered questions. What does it imply when the timescale of buffer-congestion events are orders of magnitude smaller than the timescale of self-clocking flows? Are flows overly reliant on loss signals and too insensitive to delay variation? Can paced delay-based algorithms like BBR coexist with loss-based oscillating algorithms such as CUBIC and NewReno? Would the general adoption of sender pacing change the picture of buffer sizing in the Internet?

How big should buffers be in the network? Or perhaps the opposite is the more practical question: How small can we provision buffers in an increasingly faster and larger network and still achieve efficient and fair outcomes in a variety of deployment environments?

All of these questions are good and legitimate for further research, experimentation, and measurement.

References and Further Reading

[0] Workshop on Buffer Sizing, Stanford University, December 2–3, 2019. `https://buffer-workshop.stanford.edu`

[1] Van Jacobson and Mike Karels, "Congestion Avoidance and Control," *ACM SIGCOMM Computer Communications Review*, Volume 18, Issue 4, August 1988.

This foundational paper is frequently cited by many papers on TCP behaviour.

[2a] Matt Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis J. Ott, "The Macroscopic Behaviour of the TCP Congestion Avoidance Algorithm," *ACM SIGCOMM Computer Communications Review*, Volume 27, Issue 3, July 1997.

Matt and Jamshid published a second paper of this topic in October 2019, where they argue that this macroscopic model will soon be completely obsolete:

[2b] Matt Mathis and Jamshid Mahdavi, "Deprecating the TCP Macroscopic Model," *ACM SIGCOMM Computer Communications Review*, Volume 49, Issue 5, October 2019.

[3] Curtis Villamizar and Cheng Song, "High Performance TCP in ANSNET," *ACM SIGCOMM Computer Communications Review*, Volume 24, No. 5, October 1994.

An effort to generalise the buffer sizing theory into observed practice in networks. It has been commonly acknowledged as the rationale for using bandwidth-delay product as the buffer sizing model for network equipment. This small-scale study was within a single network, and the results have been applied in a far more diverse set of deployment scenarios than the single setup that was analysed in this paper.

[4] Guido Appenzeller, Isaac Keslassy, and Nick McKeown, "Sizing Router Buffers," *ACM SIGCOMM Computer Communications Review*, Volume 34, Issue 4, September 2004.

A widely cited paper that provides an analysis of multiple diverse flows over a single common buffer, concluding that an efficient and fair buffer size model is related to the inverse of the square root of the number of active flows that traverse this common link (and buffer).

[5] Nick McKeown, Guido Appenzeller, and Isaac Keslassy, "Sizing Router Buffers (Redux)," *ACM SIGCOMM Computer Communications Review*, Volume 49, No. 5, October 2019.

A very recent review of the buffer sizing conversation and highlighting some of the significant experiments with small buffers in large networks since the 2004 paper. The paper includes numerous questions about future requirements for buffer sizes.

[6] Sally Floyd, K. K. Ramakrishnan, and David L. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, September 2001.

The specification of re-purposing two bits in the IPv4 packet header for routers to use to mark congestion events into active flows.

[7] Pat Bossharty, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker, "Programming Protocol-independent Packet Processors," *ACM SIGCOMM Computer Communications Review*, Volume 44, No. 3, July 2014.

The specification of a programming language for packet processors that has been used in recent very-high-speed packet-switch processors.

[8] "TCP Congestion Control," Wikipedia,

https://en.wikipedia.org/wiki/TCP_congestion_control

[9] Geoff Huston, "MSS Values of TCP," *The Internet Protocol Journal*, Volume 22, No. 3, December 2019.

[10] Geoff Huston, "Buffers and Protocols," Presentation at RIPE 80 Meeting, May 12, 2020. Slides and video available at:

https://ripe80.ripe.net/archives/video/316/

GEOFF HUSTON, B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. He has been closely involved with the development of the Internet for many years, particularly within Australia, where he was responsible for building the Internet within the Australian academic and research sector in the early 1990s. He is author of numerous Internet-related books, and was a member of the Internet Architecture Board from 1999 until 2005. He served on the Board of Trustees of the Internet Society from 1992 until 2001. At various times Geoff has worked as an Internet researcher, an ISP systems architect, and a network operator. E-mail: **gih@apnic.net**

Mail Security with DMARC and ARC

by John Levine

Electronic mail is both one of the most useful services of the Internet and the most frustrating. The best thing about mail is that anyone can send a message to anyone else without pre arrangement, while the worst thing about mail is that anyone can send a message to anyone else without prearrangement. As mail became ever more ubiquitous in the 1990s and 2000s, an increasing fraction of it was mail the recipients didn't want. In 2005, Dave Crocker[1] and John Klensin[2] wrote articles in this journal about the spam problem. Since then, several of the anti-spam techniques described in Crocker's article have become ubiquitous as the spam problem has become worse.

One can distinguish between *spam*, unsolicited mail sent in bulk, and *phishing*, mail sent to trick the recipient into revealing account credentials or other private information. (Some phishes are sent in bulk, some are sent to specific victims, known as *Spear Phishing*.) Starting in 2007, PayPal, which had long been among the biggest phishing targets, started working with some large consumer mail systems to keep PayPal phishes out of recipient mailboxes. The idea was that the recipient systems could identify genuine e-mail from PayPal, and reject anything else purporting to be from PayPal. In 2012 an industry group started the DMARC project to generalize this technique. In 2015 the *Domain-based Message Authentication, Reporting & Conformance* (DMARC) specification was published as an independent track RFC[3], and now it is ubiquitous in large mail systems.

DMARC works by tying the address in the RFC 5322[4] **`From:`** header of a message to mail authentication, and letting a domain offer policy advice to mail recipients. If a message is successfully validated by *Sender Policy Framework* (SPF) or *DomainKeys Identified Mail* (DKIM), and the domain in that validation matches the one in the **`From:`** header, the message is DMARC "aligned." Sending mail systems can publish DMARC policy records in the *Domain Name System* (DNS) requesting recipient systems to quarantine (send to the spam folder) or reject unaligned mail. This system works quite well for the original intended application of DMARC, business-to-consumer mail, where the sending organization generally has full control over all mail sent from its domain. It works particularly well for PayPal, where all the mail is some variation of "log into your account to see what's new," so if a few messages are accidentally lost because DMARC miscategorizes a legitimate message it's not a big problem.

Underlying and Previous Work

DMARC depends on two existing mail authentication schemes, *Sender Policy Framework* (SPF)[5] and DKIM[6]. SPF does path validation of the domain in the RFC 5321[7] **`MAIL FROM:`** address. A domain can publish an SPF record that uses a complex syntax to specify a set of IP addresses.

If the message was sent from one of those addresses, SPF validation passes. (This description is oversimplified; see [5] for the full details.) SPF has the virtue of being easy to implement because it requires no changes to outgoing messages and a single DNS record to implement, but it can describe only a limited subset of the ways that mail is delivered. For the most part it can handle only mail sent directly from sender to recipient, without any forwarding or remailing, and does not deal well with mail sent by third parties on the sender's behalf. While SPF provides a `-all` code that advises recipients to reject mail from the domain if SPF validation fails, most mail systems disregard the advice because the false positive rate is so high.

DKIM does message content validation by adding cryptographic signature headers to a message that the recipient can check using a key in the DNS. Each signature is stored in a `DKIM-Signature` header field that contains several subfields, including the name of the domain that added the signature. If a DKIM signature validates, it means both that the message hasn't been modified since it was signed and that the domain in the signature takes responsibility for the message. Since DKIM validates the contents of the message rather than the path, it is unaffected by forwarding.

DKIM is considerably harder to implement than SPF because it requires modifications to mail software to add the signature headers to each outgoing message. It also requires the signing system to create public/private key pairs, publish the public key in the DNS, and configure the private key into the signing software. DKIM validation fails when a message is modified in transit, such as when a mailing list adds a subject tag or a message footer, and sometimes simply because a *Message Transfer Agent* (MTA) hasn't been configured to add the signature in the first place. (In large enterprises it can be remarkably hard to track down all of the computers sending mail. DMARC helps address this problem, as we will see later.)

DKIM had an optional add-on called *Author Domain Signing Practices* (ADSP)[8], which was sort of a proto-DMARC. A domain could publish an ADSP record in the DNS saying that if a message with the domain in the `From:` header didn't have a valid DKIM signature from the same domain, recipients should discard the message. ADSP was never deployed beyond experiments, and the *Internet Engineering Task Force* (IETF) has since made it an historic specification.

DMARC Deployment

One of the reasons that previous sender policy approaches like SPF `-all` and ADSP failed is that there is no way to test them other than turning them on to see what happens. For a small domain with one or two mail servers that might be possible, but for a large organization the risk is impossibly high since they rarely have complete knowledge of all of the systems sending their mail and how those systems are configured.

DMARC offers a variety of features to check the alignment of the mail of a domain before publishing any policy advice. It has powerful reporting features that let a domain owner ask other systems to send reports about the mail purporting to be from the domain. Domains invariably ask for reports before publishing any policies, so they can see what mail they send is and isn't aligned. This information lets them fix alignment issues before they do publish policies.

DMARC Validation

When a message arrives, DMARC validation involves first finding a DMARC policy record for the From: header domain, then validating SPF and the DKIM signature(s) on the message, and then perhaps doing something to the message. The first step is to find the policy record for the **`From:`** domain of the message, a DNS TXT record. If that domain is **`marketing.mybiz.example`**, the first place to look is **`_dmarc.marketing.mybiz.example`**. If there is a TXT record in DMARC syntax, for example, it starts with **`v=DMARC1;`** that's the policy record. If not, it looks for a policy record in the "organizational domain."

The DMARC specification is deliberately vague about how to find the organizational domain, but in practice everyone uses the Mozilla *Public Suffix List* (PSL)[9] where the organizational domain is the superdomain just below the public suffix. In this case if the domain were a typical *Top-Level Domain* (TLD) that accepts registrations at the second level, the organization domain would be **`mybiz.example`**, so it would look for a TXT record at **`_dmarc.mybiz.example`**. If there is a TXT record in DMARC syntax, that's the policy record; otherwise there is no policy record for this domain.

The DMARC record is a list of key=value pairs, with rules for checking alignment, what to do with unaligned mail, and where to send aggregate and failure reports. A typical record might be:

```
v=DMARC1; p=none; rua=mailto:dmarc-a@example.net;
ri=3600; ruf=mailto:dmarc-f@example.net
```

There is no policy (**`none`**), abuse and failure reports are mailed to the given addresses (**`rua`** and **`ruf`**), and the requested report interval is an hour (**`ri`** is 3,600 seconds.) The point of the second check for the organizational domain is twofold. First, the second check makes it easier to deploy DMARC across a large enterprise, since one DMARC organizational record can cover all of an organization's subdomains. The other is that it covers non-existent subdomains of the organizational domain, for when hostile or buggy mailers send mail purporting to be from such a subdomain.

The next step in validation is to check whether the **`From:`** header domain is aligned with the SPF identity of the message. The SPF validation process can produce a result of *None, Neutral, Pass, Fail, Softfail, Temperror,* or *Permerror*. For DMARC alignment, only a *Pass* result is acceptable.

The DMARC policy record can require strict SPF alignment, meaning the **`From:`** domain and SPF identity have to be the same, or relaxed SPF alignment, meaning they need only be in the same organizational domain. In the previous example, if the **`From:`** domain were **`marketing.mybiz.example`**, an SPF identity of **`mail.mybiz.example`** or just **`mybiz.example`** would be sufficient for relaxed SPF alignment. Relaxed alignment is the default.

Next, the validator checks for DKIM alignment. For each valid DKIM signature on the message, the validator compares the **`From:`** domain to the **`d=`** domain of the signature. The policy record can specify strict or relaxed DKIM alignment, again requiring either an identical signature domain or just one in the same organizational domain. If at least one valid DKIM signature is aligned, the message is DKIM aligned. If the message is either SPF or DKIM aligned, it is DMARC aligned.

If the message is aligned, we're done other than perhaps saving some statistics for later reporting. If it's not aligned, the situation is potentially much more complex if the recipient system opts to follow the policy advice, as most mail systems (at least by mail volume) now do.

The policy record can specify policy advice of **`none`**, **`quarantine`**, or **`reject`**. It can also specify an optional percentage of how often to apply the policy. Advice of **`none`** means to do whatever the recipient would have done with the message anyway. Advice of **`quarantine`** means to treat the message extra skeptically, perhaps by filing it in a spam folder or marking it as suspicious. Advice of reject asks the recipient to **`reject`** the message at the end of the *Simple Mail Transfer Protocol* (SMTP) session and not handle it further. If the percentage is less than 100, the advice is to treat that percentage of unaligned mail from the domain according to the advice, and the rest one step less. For example, if the advice were **`reject`** and the percent was 25, one-fourth of unaligned mail would be rejected and the other three-quarters would be quarantined. (The percent has no effect if the policy is **`none`**.)

As noted previously, the point of the percentage is to allow domain owners to enable policies gradually, see what happens, and limit the damage from misconfigurations.

DMARC Reporting

DMARC has two powerful reporting features. A domain can ask for daily aggregate reports of what IP addresses have sent mail with the domain in the **`From:`** header, with details about DMARC alignment and DKIM and SPF validation. Many large mail systems including Google, Yahoo/AOL, Comcast, and Fastmail send aggregate reports.

It is also possible to request copies of messages that fail DMARC validation, but for privacy reasons very few systems do. The only significant mail system in the U.S. that sends failure reports is LinkedIn.

Even for a site that has no plans to publish a DMARC policy, the reports are useful and interesting. They can provide insight into where your mail is actually going, and who else might be sending mail purporting to be from you.

To request each kind of report, the domain policy record includes a tag with a list of mailto: *Uniform Resource Identifiers* (URIs), each with an optional size limit of the maximum message report size the system can handle. The default aggregate report interval is once a day.

Aggregate reports constitute an *Extensible Markup Language* (XML) file attached to an e-mail message in gzip or ZIP compressed form. The XML file includes a section (a "record") for each sending IP address, with subsections (a "row") for each combination of authentication results. For example, here's a section of a report Google sent to my *Smail* system describing mail it received from two IP addresses:

```
<record>
   <row>
     <source_ip>2001:470:1f07:1126:0:43:6f73:7461</source_ip>
     <count>1</count>
     <policy_evaluated>
       <disposition>none</disposition>
       <dkim>pass</dkim>
       <spf>pass</spf>
     </policy_evaluated>
   </row>
   <identifiers>
     <header_from>taugh.com</header_from>
   </identifiers>
   <auth_results>
     <dkim>
       <domain>iecc.com</domain>
       <result>pass</result>
       <selector>k1912</selector>
     </dkim>
     <dkim>
       <domain>taugh.com</domain>
       <result>pass</result>
       <selector>k1912</selector>
     </dkim>
     <spf>
       <domain>taugh.com</domain>
       <result>pass</result>
     </spf>
   </auth_results>
</record>
```

```
<record>
   <row>
     <source_ip>209.85.220.55</source_ip>
     <count>4</count>
     <policy_evaluated>
       <disposition>none</disposition>
       <dkim>fail</dkim>
       <spf>fail</spf>
     </policy_evaluated>
   </row>
   <identifiers>
     <header_from>taugh.com</header_from>
   </identifiers>
   <auth_results>
     <dkim>
       <domain>googlegroups.com</domain>
       <result>pass</result>
       <selector>20161025</selector>
     </dkim>
     <spf>
       <domain>googlegroups.com</domain>
       <result>pass</result>
     </spf>
   </auth_results>
</record>
```

The first record for the IPv6 address reports on a message sent from my mail server. It has a valid SPF and two valid DKIM signatures, one with the **`From:`** header domain and one for the server domain, so it was DMARC aligned. The second record describes four messages with valid SPF and DKIM signatures, but with SPF and DKIM domains that don't match the **`From:`** header, so they wouldn't have been DMARC aligned. Since the second group of messages have **`googlegroups.com`** authentication identifiers, they're probably the same message, modified and remailed to a Google Groups mailing list. [Since I know I sent only one message to the list that day, this message leaks the number of *Gmail* subscribers to the list. I've seen similar leakage for much larger lists; for example, like the one operated by the *North American Network Operators' Group* (NANOG).]

Larger mail systems receive reports with larger numbers of messages and more report sections. The reports are intended to be mechanically handled. Some open source software is available to analyze reports and put summaries in a database[10]. More often the reports are sent directly to specialist services like *Dmarcian*[11] or *Agari*[12] that offer freemium report analysis services, simple analysis for free, or more sophisticated analysis and remediation advice for a fee.

The other kind of report is a failure report. When a message arrives that has the domain address in the **`From:`** header and fails DMARC validation, the recipient system may (but usually doesn't) send the message back in a failure report. The report is a multipart/report e-mail message containing a structured report section and a full or partial copy of the failing message.

A typical report section follows:

```
Feedback-Type: auth-failure
User-Agent: Lua/1.0
Version: 1.0
Original-Mail-From: nanog-bounces@nanog.org
Original-Rcpt-To: xxx@linkedin.com
Arrival-Date: Thu, 26 Dec 2019 19:22:54 +0000
Message-ID: <20191226191849.6BBF111BA67D@ary.qy>
Authentication-Results: dmarc=fail (p=none; dis=none)header.from=iecc.com
Source-IP: 50.31.151.76
Delivery-Result: delivered
Auth-Failure: dmarc
Reported-Domain: iecc.com
```

The message in a failure report might be a legitimate one that was unaligned when sent, or modified on the way to become unaligned. Or it might be a fraudulent one, either an attempted phish, or just a random spam message where the spamware happened to pick your domain for the fake return address. For this particular report, it's obviously a real message relayed through the NANOG mailing list.

The original failure report included the full address of the recipient, meaning that by looking at the failure reports, anyone who posts to NANOG can see who subscribes to LinkedIn. This kind of data leakage explains why most sites don't send failure reports at all, and most of the ones that do limit what they send, typically including only the headers of a failing message and redacting recipient address details.

Using DMARC Reporting to Prepare for Policy Publication

Before publishing a DMARC policy of **`quarantine`** or **`reject`**, domain operators should be confident that as close as possible to 100% of the mail they send is DMARC aligned. They might send unaligned mail if SPF records of a domain do not cover all of the IP addresses that send valid mail, causing SPF validation to fail. Some outgoing *Mail Transfer/Transfer Agents* (MTAs) might have DKIM configured incorrectly or not at all, so there's no aligned DKIM signature. Large organizations often can have MTAs sending mail that the network managers didn't know about; for example, if a department set up its own local server, or contracted with a third-party mail sender.

The data from DMARC reports tells the operator what IP addresses are sending unaligned mail, and generally makes it straightforward to figure out why it's unaligned. Mitigation might involve updating the domain SPF records to include missing MTAs, fixing the DKIM signing configuration in MTAs, or enforcing rules about unapproved mail servers or third-party mail senders. (Many third parties can do DKIM signing with a client's domain, but that requires either sharing the private signing keys(s) or delegating a DNS subtree that the third party can manage.)

After the operator has the mail sufficiently under control, it can gradually turn on sending policies. DMARC provides the quarantine policy as an intermediate step between no policy and reject so there is a chance for recipients to retrieve miscategorized mail. It can also use the percentage parameter in the policy record to apply policies gradually and limit the damage if mistakes occur.

DMARC vs. Discussion Lists

DMARC was originally intended for domains at organizations like banks that send primarily business-to-business and business-to-customer mail, and little or no person-to-person mail. When the organization considers when to publish a DMARC policy, and what policy to publish, it should remember that some fraction of its legitimate mail will arrive unaligned because of intermediate processing that DMARC cannot describe. Since the organization presumably knows what mail it sends, it can weigh the benefits of less phishing versus the cost of lost mail and make a decision that is reasonable for the organization.

In 2014, AOL and Yahoo, two large consumer mail systems, had separate security breaches in which intruders stole copies of millions of their users' address books. The stolen data was quickly sold to spammers, who used it to send spam to AOL and Yahoo users that appeared to be from the recipients' friends. This situation caused an expensive support problem at AOL and Yahoo as users complained about the spam and asked why their friends were spamming them. First AOL, and then Yahoo, "solved" the problem by quickly publishing DMARC `p=reject` policies that told every mail system that implements DMARC to reject any AOL or Yahoo mail that didn't come directly from AOL or Yahoo. This decision was very different from the ones made by organizations described previously. In this case the benefit of the policy was to mitigate the cost of an operational failure, with little if any benefit for most of their users, while creating major problems for their discussion list users.

A small but important part of the mail from users of any consumer mail system is unaligned yet legitimate mail that recipients want. That happens typically because the mail is routed indirectly from the sending user to the ultimate recipients. A particular point of contention is e-mail discussion lists where the normal actions of list managers make most of the mail unaligned. This situation can cause non-receipt of mail sent to subscribers on mail systems that enforce DMARC policies on incoming mail, and it can also cause removal of subscribers from lists because of bounces caused by DMARC failures. (Yahoo was aware of the mailing list issues but decided to publish `p=reject` anyway, according to someone who was there at the time.) Another source of unaligned mail is third-party mailing services. A small organization like an athletic club or scout troop often has an announcement list where the return address on the announcements is the personal address of the organization's secretary, who may use AOL or Yahoo.

A variety of proposed workarounds have been made for the problems that DMARC causes to mailing lists, none of which are very satisfactory. Initially, the easiest approach was to tell people sending mail from addresses with DMARC policies to subscribe from another address. That approach stopped being practical when AOL and Yahoo flipped the switch.

Since then, mailing list software has taken a range of approaches to ensure that the messages the list sends out are aligned. In a few cases, lists tried to turn off any features that would modify messages in ways that would invalidate DKIM signatures, hoping that DKIM signatures on incoming messages would remain valid when resent from the list. This idea didn't work very well, partly because remailed messages weren't SPF aligned (the list uses its own envelope address for bounce management), and users want the changes that lists make, such as adding subject line tags to identify the list.

Mailing lists have settled on two general anti-DMARC approaches[13]. The most common is to put the list address into the **From:** header so the list can add a DKIM signature with its own domain and make the message DMARC aligned. For example, if the incoming message included:

```
From: Steve C <steve@aol.com>
To: nodule@lists.example.com
```

The list might rewrite it as:

```
From: Steve C via the nodule list <nodule@lists.example.com>
To: somelist@lists.example.com
Reply-To: Steve <steve@aol.com>
```

The rewritten **From:** header usually includes the author's address comment and the list name. The author's actual address is placed in a **Reply-To:** header, or occasionally a **Cc:** header. This approach allows DMARC alignment, since the list can add a lists.example.com DKIM signature, but makes mail from the list harder to handle. Mail user agents treat **Reply-To:** in different ways, leading to confusion about whether someone is replying to the author of a message, or to the list, or both. Adding to the confusion, some lists only rewrite the headers for messages in author domains that publish a DMARC policy, so messages from the same list have different headers.

Another approach is to rewrite the **From:** header to replace the problematic author address with one that is DMARC aligned but still represents the author. For example, my mailing lists would rewrite the headers in the previous example like this, changing the author address only by adding a local domain suffix:

```
From: Steve C <steve@aol.com.dmarc.fail>
To: nodule@lists.example.com
```

The domain **dmarc.fail** is a real domain I registered. (It was available.) I publish an MX record for ***.dmarc.fail**, to catch any mail sent to rewritten addresses. The rewritten message as sent by the mailing lists has a **dmarc.fail** DKIM signature, so it's properly DMARC aligned. When the list software rewrites an address, it creates a forwarding entry for the rewritten address that redirects back to the original address. The forwarding entries are deleted after a few days so that replies to the author sent shortly after the original message go back to the author, but the forwarding is limited, so it's not a useful vector to relay third-party spam.

This technique works fairly well. Since only the **From:** header is changed, there's no effect on **Reply-To:** or other mail behavior, and the author's identity is easy to recognize. Other systems have implemented the same idea in perhaps less passive-aggressive ways. The IETF's working mailing lists rewrite the address into the local part; for example:

```
From: Steve C <steve=40aol.com@lists.ietf.org>
```

The commercial LISTSERV mailing list service rewrites the address into an opaque local address and puts the real address in **Reply-To:**

```
From: Steve C <00000006b01fa96f-dmarc-request@lists.example.com>
Reply-To: Steve C <steve@aol.com>, Nodule list <nodule@lists.example.com>
```

The primary disadvantage of the address rewriting is that it requires access to the local mail system of the list to manage the set of temporary forwarding addresses, rather than doing it entirely inside the list software.

The other anti-DMARC approach that some lists take is message wrapping, enclosing the message as a *Multi-Purpose Internet Mail Extensions* (MIME) part within an outer message from the list. Most mailing lists have a MIME digest option, to send the day's messages as a set of MIME parts within a single daily message, so this process in effect turns each message into a one-message digest. The outer message typically has the list address in the **From:** header, while the inner message is unmodified.

Technically, this approach should work well, because it uses existing well-standardized features of RFC 5322 mail. Having done some experiments to see how workable it is, I found that in practice it works very badly because mail user agents treat MIME attached messages as an afterthought. While the inner message is typically displayed legibly, it is often not possible to reply to the inner message without clumsy extra steps, or in some cases at all, and multipart messages or those with attachments are handled inconsistently. The IETF experimented with several varieties of MIME wrapping before deciding that rewriting the **From:** header was the best of a bad lot.

While all of these approaches allow mailing lists to send DMARC-aligned mail, none of them are very satisfactory, and none let mailing lists work as well as they did before DMARC.

ARC

While the amount of mail that large providers get from mailing lists is small, on the order of 1% to 2% of the non-spam total, it is mail that the recipients care about deeply. After years of complaints, several large mail providers developed *Authenticated Received Chain* (ARC) to help them handle wanted but unaligned user mail.

An obvious way to handle unaligned mail from mailing lists would be to whitelist them. Large mail systems have a pretty good idea of where the lists are (the number of mailing list hosts worldwide is probably only about 10,000), so they could just accept the mail from the lists that they know their users want. The problem with this concept is that mailing lists don't do a very good job of spam filtering, and spam leaks through them all the time.

In particular, most lists check only that the address in the **`From:`** header is subscribed to the list before forwarding a message. If a subscriber's account is compromised and starts sending spam, any message sent to a list will generally get forwarded to the list. Even without an account being compromised, if a stolen address book happens to contain your address and the address of a list to which you subscribe, spamware can forge mail from you to the list, and again the list will forward it. I've seen this happen multiple times, and it is quite frustrating since the person whose address is being forged can't do anything about it.

The goal of ARC is to add a "chain of custody" to a message that shows what happened to it each time it was forwarded. This technique lets the ultimate recipient system retroactively make spam filtering decisions based on what happened to the message at the forwarding systems.

ARC builds on existing mail technology. It adapts the *Authentication-Results* (A-R) header[14] that many mail systems apply to incoming messages that records the authentication status of the message at the time an MTA received the message. Here is a typical A-R header that my MTA applied to an incoming message from Apple's **`me.com`**:

```
Authentication-Results: iecc.com; spf=pass spf.mailfrom=xxx@me.com
    spf.helo=mr85p00im-hyfv06011401.me.com smtp.remote-ip="17.58.23.191";
    dkim=pass header.d=me.com header.s=1a1hai header.a=rsa-sha256;
    dmarc=pass header.from=me.com (p=quarantine, pct=100)
```

The first field is the name of the system that added the header, followed by groups of authentication results, in this case for SPF, DKIM, and DMARC. Each group includes the result and relevant items like the envelope **`MAIL FROM`** and sending IP for SPF. All of the fields are optional other than the system name, and they are added only for the kinds of authentication the system checked. ARC combines a modified A-R header and two DKIM-like signature headers into an ARC *seal*, which is intended to describe the passage of a message through a system such as a mailing list manager. A single message may have multiple ARC seals if it has passed through multiple forwarding systems.

Each seal is numbered, starting with 1 for the first seal applied. Each header in an ARC seal has an **i=** clause to indicate which seal it's part of.

The headers in an ARC seal look like this:

```
ARC-Message-Signature: i=1; a=rsa-sha256; d=microsoft.com; s=abcd; h=From:Date:...
ARC-Authentication-Results: i=1; mx.microsoft.com 1; spf=pass …; dkim=pass …
ARC-Seal: i=1; a=rsa-sha256; s=abcd; d=microsoft.com; cv=none; b=j7M/jt9eVP…
```

The *ARC-Message-Signature* (AMS) is almost identical to a DKIM signature, with the added **i=** field. It is intended to cover the usual headers and body of the message, at the time the message was sent from the signing system. If the system makes changes to the message, the AMS is applied after those changes. When a message is received, the most recent AMS signature will be valid unless an inter-mediate system has modified the message since the ARC seal was applied and not added a seal of its own.

The *ARC-Authentication-Results* (AAR) header reports the authentication status at the time the sealing system received the message; that is, before any modifications reflected in the AMS.

The *ARC-Seal* header is a DKIM-like signature that covers only the three headers in the ARC seal, to validate the seal itself. It also indicates whether the chain of ARC seals in the message was intact when the message was sealed, using the **cv=** (chain value) field. If this seal is the first one, the chain value is "none" for no previous seal. For any subsequent seal, the chain value is "pass" if the previous seal was valid (the DKIM-like signatures validated) and the previous seal had **cv=none** or **cv=pass**. Otherwise the chain value is "fail."

If a mail system receives a message from a trustworthy source with a valid ARC chain, it can use the information in the ARC seals to make exceptions to its DMARC policy. As a simple example, assume a message that is not DMARC aligned arrives, but it has a valid chain of ARC seals. In one of the seals, an AAR header shows that the message was DMARC aligned (**dmarc=pass)** and the **header.from** domain was the same as the one currently in the message. That means the lack of alignment is due to changes made by the forwarding system. If the forwarding system is considered trustworthy, for example, a host that hosts discussion lists, the receiving system can decide to deliver the message. More complicated analysis is possible, but I expect this sort of analysis looking for typical mailing list operations is likely to be the most common. Since malicious systems can add fake ARC seals, this analysis makes sense only for mail from trust-worthy sources. Identifying sources trustworthy enough to apply ARC exceptions may be a problem for mail systems too small to develop reliable data on hosts that send mail to them. There are some efforts to provide shared lists of reputable mailing list hosts that will likely be good enough, since the number of active list hosts is not large and changes slowly.

At this point the implementation of ARC has started, but it is not yet common enough to let mailing lists stop doing anti-DMARC header munging. *Python* and *Perl* libraries for DKIM have both added ARC support[15]. The Sympa 6.2 mailing list manager has ARC support, as does GNU Mailman 3.1, but not Mailman 2.x.

Large mail systems including Google's *Gmail* and Microsoft's `outlook.com` have some ARC support, and both *Gmail* and `outlook.com` put ARC seals on forwarded and mailing list mail, but neither is yet using it for mail filtering other than experimentally. Few mailing lists yet add ARC seals, partly because of the lack of ARC support in the list software they currently use, and partly because the list managers are unaware of ARC.

Conclusions

DMARC started as a relatively simple technique to deter phishing of high-profile commercial domains such as those of banks and payment providers. Consumer mail systems AOL and Yahoo then repurposed it to deal with spam forging their users' addresses. While this repurposing largely solved the spam forgery problem of mail systems, it caused severe collateral damage to e-mail discussion lists. While many lists have tried to work around the DMARC problems, all of the workarounds have drawbacks that make them ultimately unsatisfactory. To help undo the DMARC damage, a group of large mail providers invented ARC, which makes it somewhat possible to examine the history of a message and see how a message that is not DMARC aligned got that way.

The ongoing evolution of DMARC, mailing lists, and ARC is yet another round of security measures with unexpected consequences. With any luck, ARC will be the end of this sequence of effect, side-effect, and counter-effect, but we won't know until ARC is more widely deployed, hopefully in a few years.

References and Further Reading

[1] Dave Crocker, "Challenges in Anti-Spam Efforts," *The Internet Protocol Journal*, Volume 8, No. 4, December 2005.

[2] John Klensin, "Another Look at Spam," *The Internet Protocol Journal*, Volume 8, No. 4, December 2005.

[3] Murray Kucherawy and Elizabeth Zwicky, Eds., "Domain-based Message Authentication, Reporting, and Conformance (DMARC)," RFC 7489, March 2015.

[4] Peter W. Resnick, "Internet Message Format," RFC 5322, October 2008.

[5] Scott Kitterman, "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1," RFC 7208, April 2014.

[6] Murray Kucherawy, David Crocker, and Tony Hansen, "DomainKeys Identified Mail (DKIM) Signatures," RFC 6376, September 2011.

[7] John C. Klensin, "Simple Mail Transfer Protocol," RFC 5321, October 2008.

[8] John Levine, Mark Delany, Eric Allman, and Jim Fenton, "DomainKeys Identified Mail (DKIM) Author Domain Signing Practices (ADSP)," RFC 5617, August 2009.

[9] See **https://publicsuffix.org/** for the PSL, and **https://wiki.mozilla.org/Public_Suffix_List** for a description of its use and history.

[10] See **https://www.taugh.com/rddmarc/**

[11] Dmarcian: **www.dmarcian.com**

[12] Agari: **www.agari.com**

[13] Mailman and DMARC, **https://wiki.list.org/DEV/DMARC**

[14] Murray Kucherawy, "Message Header Field for Indicating Message Authentication Status," RFC 8601, May 2019.

[15] See **https://pypi.org/project/dkimpy/** for the *Python* library, and **https://metacpan.org/release/Mail-DKIM** for the *Perl* library.

JOHN R. LEVINE writes, speaks, and consults on the Internet, electronic mail, cybersecurity, and related topics. He speaks to many trade, policy, and general groups. He has testified at the *Federal Trade Commission Spam Forum* on the mechanics of spam, to the *Senate Commerce Committee on Spyware*, and is part of the *Industry Canada Task Force on Spam*. He has spoken at the Internet Law and Policy Forum and at many conferences. He is frequently interviewed in the print and electronic media and has extensive working relationships with reporters. John consults and provides advice and expertise on e-mail and Internet systems, security, and software. He co-founded the *Domain Assurance Council*, a non-profit industry consortium that establishes standards for e-mail certification and security. Levine has served as an expert witness on a variety of computer topics including e-mail spam, compiler software, and graphic image file formats. He has written many books on the Internet and other computer topics. His books range from the best-selling *Internet for Dummies*, with over seven million copies of eleven editions in print in dozens of languages, *Fighting Spam for Dummies*, and *Windows Vista: The Complete Reference*, to books on computer language tools and graphics programming. E-mail: **johnl@taugh.com**

Letter to the Editor

Ole,

I enjoyed reading Geoff Huston's article "MSS Values of TCP," (*The Internet Protocol Journal*, Volume 22, No. 3, December 2019). I had not been familiar with the variation of *Maximum Segment Size* (MSS) values that are used in the broad Internet.

I have never run into a client or server device that has been unable to operate with greater than a 576-byte *Maximum Transmission Unit* (MTU). Even the early Intel 8088-based MS DOS PCs in the 1980s with 3Com 3c501 *Network Interface Cards* (NICs) could handle 1500-byte MTU. In modern times only a tiny fraction of our tens of thousands of connected hosts are capable of Ethernet Jumbo packets (SAN nodes replicating data between data centers; our routers along just those paths are configured for 9,216-byte MTU).

Geoff touched only briefly on encapsulation influencing resulting TCP MSS values. Our *Wide-Area Network* (WAN) connections between our office locations use *Internet Protocol Security* (IPsec) tunnels, and we also use IPsec tunnels between our office locations and our virtual routers inside the Amazon *Virtual Private Cloud* (VPC). In addition to those, we have some *Generic Routing Encapsulation* (GRE) tunnel connections with some information providers and partners (Zscaler is an example). With the recent availability of low-cost high-speed Layer 2 connections between some sites, we have been implementing *Media Access Control security* [MACsec] (lower router resource consumption than IPsec). We also are beginning to use *Virtual Extensible LAN* (VXLAN) between our two data centers. With all of these encapsulations in our network, we've been avoiding IP fragmentation of TCP packets by configuring TCP **`adjust-mss`** on our Cisco routers. We also use **`adjust-mss`** on our Wireless LAN Controllers.

We still face plenty of *User Datagram Protocol* (UDP) packets from video devices that would need to be fragmented if attached with default configuration. When our IT group controls such devices, we configure their MTU to fit within our IPsec tunnels.

Regards, —*Richard Berke,* **`Richard.Berke@troweprice.com`**

The author responds:

Richard,

Thanks for your comments about MSS sizes and the related topic of MTU selection.

We need to go back to the 1970s to find some variation from the current ubiquity of the 1,500-octet MTU that dominates today's communications, and very little of that early network environment has survived. However, we can piece together some of the thinking behind the original design of the Internet Protocol and the selection of MTU and MSS values.

Smaller packet sizes made packets less susceptible to bit-error-rate corruption and could reduce jitter (which was a major consideration behind the design of *Asynchronous Transfer Mode* (ATM) cells), but at the same time smaller packets had a reduced payload efficiency. Various mainframe vendors tuned their network products to match their intended deployment environment, including the choice of supported packet sizes.

As a "network of networks," the Internet was envisaged to work across various permutations of networks, all with differing MTU sizes. The fragmentation model of IP Version 4 came from this approach, where an IP router was permitted to fragment a packet if it was too large for the next network.

IP Version 4 permitted IP packets between 20 and 65,535 octets in size. While in a strict sense the minimum MTU is 20 octets, that is without any payload at all. A 21-octet MTU would make some level of progress in sending a payload, albeit extremely inefficiently.

Where does 576 octets come from? IP hosts were not required to accept the entire protocol-permitted range of packet sizes. The specification required IP hosts to reassemble and accept IP packets up to 576 octets in size. Why 576? It is such an odd number. I could understand a value of 532, 542, 572, or even 592 octets, all based on a 512-octet payload and various permutations of minimum or maximum IP headers and optionally including a TCP header. However, I can't get to 576 octets that way, so I don't have any credible explanation as to why this value was chosen.

By the time we were designing IPv6 in the early 1990s the thinking had changed, and fragmentation was frowned upon. It was slow and insecure, and the experts advised its avoidance wherever possible. What should the minimum unfragmented MTU be for IPv6? Ethernet framing was ubiquitous by this time, so a 1,500-octet MTU size seemed like a good first answer. But the Internet had a new aspect by then: it was no longer a "network of networks" but was the base substrate network upon which other networks were overlaid. Various other headers, including IP-in-IP, were being added. So, we needed to specify a universal minimum unfragmented IPv6 packet size that would be relevant in many kinds of IP-in-IP contexts. The value of 1,280 octets as the new minimum unfragmentable packet size was chosen for IPv6. Why 1,280? I understand that this number was chosen because it's the sum of 1,024 and 256.

My view is that the marginal loss of payload efficiency is small enough that for the public Internet a 1,220-octet MTU can be used with some confidence that it will not encounter MTU mismatch issues.

Regards,

—*Geoff Huston,* **gih@apnic.net**

Keeping the DNS Secure During the Coronavirus Pandemic

The Internet's value in bringing people together has never been more apparent than it is now. While most of us are under some form of "stay at home" order in an effort to slow the spread of the coronavirus, the Internet provides us with a lifeline. It brings us information and entertainment, allows some of us to continue our work and education, and brings us what we need most at times of isolation—social connections.

The role of the *Internet Corporation for Assigned Names and Numbers* (ICANN) community, Board, and organization in maintaining a secure, stable, and unified Internet has always been important, but at this time, when reliance on the Internet has skyrocketed, our collective role has become all the more vital. ICANN's mission frames our concern about cybercriminals who are exploiting the pandemic by perpetrating scams and victimizing Internet users. Some are selling phony cures, treatments, and vaccines. Some are using domain names as part of their efforts to prey on people at this time when many are experiencing anxiety, fear, and loneliness. The U.S. *Federal Trade Commission* reports that it has fielded more than 7,800 coronavirus-related complaints. The agency noted that U.S. consumers alone have collectively lost more than U.S. $5 million.

Of course, ICANN cannot involve itself in content issues, both because of our Bylaws as well as practically, but that does not mean we are unaware or unconcerned about those who are using the *Domain Name System* (DNS) to victimize others. It is this concern that prompted me to contact the registries and registrars thanking them for their efforts and actions aimed at helping to mitigate and minimize the abusive domain names being used to maliciously take advantage of the coronavirus pandemic. For example, the *Registrar Stakeholder Group*[1] has posted a useful guide, entitled "Registrar approaches to the COVID-19 Crisis" that provides a number of steps and resources the registrar community can use in their efforts.

Many of our contracted parties already support a *Framework to Address Abuse*,[2] which deals with DNS abuse and website content abuse. I continue to commend them for making this commitment to protect the DNS from those who would maliciously exploit domain names. In my correspondence to the registries and registrars, I expressed ICANN org's appreciation for their work during the pandemic.

Additionally, I'm pleased to tell you that ICANN org has joined registries, registrars, security experts, law enforcement, Internet engineers, and others, in the COVID-19 *Cyber Threat Coalition* (CTC)[3]. The CTC's mission is to, "operate the largest professional-quality threat lab in the history of cybersecurity out of donated cloud infrastructure and with rapidly assembled teams of diverse, cross-geography, cross-industry threat researchers."

I am proud that so many in the Internet ecosystem are joining together during this crisis to stop those who prey on the desperate. We will continue to keep you advised of our engagement efforts to mitigate the misuse of domain names during these critical times.

—Göran Marby, President and Chief Executive Officer, ICANN

[1] `https://rrsg.org/`

[2] `http://dnsabuseframework.org`

[3] `https://www.cyberthreatcoalition.org/`

Global Encryption Coalition Formed

Encryption is a critical technology that helps keep people, their information, and communications private and secure. However, some governments and organisations are pushing to weaken encryption, which would create a dangerous precedent that compromises the security of billions of people around the world. Actions in one country that undermine encryption threaten us all.

As a global coalition, we call on governments and the private sector to reject efforts to undermine encryption and pursue policies that enhance, strengthen and promote use of strong encryption to protect people everywhere. We also support and encourage the efforts of companies to protect their customers by deploying strong encryption on their services and on their platforms.

The mission of the *Global Encryption Coalition* is to promote and defend encryption in key countries and multilateral gatherings where it is under threat. It also supports efforts by companies to offer encrypted services to their users.

With a steering committee led by the *Center for Democracy and Technology* (CDT), *Global Partners Digital* (GPD) and the *Internet Society* (ISOC), the Global Encryption Coalition is composed of national coalitions, civil society groups, corporations, academics, and technologists around the world who agree to support its founding statement.

For more information, visit: `https://www.globalencryption.org/`

APNIC Launches Networing from Home Events

With most Asia Pacific economies forced into various states of lockdown to minimize COVID-19 infections, *Network Operator Group* (NOG) meetings and other technical events in the region have either been cancelled or postponed. NOGs are a great forum for network engineers to share experience with their peers, work out solutions to common technical problems, and build the strong relationships that help the Internet operate. There are 22 NOGs in the APNIC region, but sadly that means a lot of events have been cancelled in 2020.

Networking from Home is a new virtual event initiative to provide a place for technical folk in the region to share their experience and expertise with their peers, just like they would at a NOG event.

There will be four free online events—one each held in the time zones of South East Asia, South Asia, East Asia, and Oceania—and they will be a digestible 2.5 hours long. Presentations will be short and punchy and interaction is encouraged! APNIC's Geoff Huston will deliver a different keynote at each event, and he will be supported by a range of great speakers suggested by the NOG communities.

If you have a great presentation in mind, get in touch with the Program Committees for the events. For more information, visit: **https://nfh.apnic.net/**

Check your Subscription Details!

If you have a print subscription to this journal, you will find an expiration date printed on the back cover. For the last couple of years, we have "auto-renewed" your subscription, but now we ask you to log in to our subscription system and perform this simple task yourself. The subscription portal is here: **https://www.ipjsubscription.org/** This process will ensure that we have your current contact information, as well as delivery preference (print edition or download). For any questions, contact us by e-mail at: **ipj@protocoljournal.org**

Our Privacy Policy

The *General Data Protection Regulation* (GDPR) is a regulation for data protection and privacy for all individual citizens of the *European Union* (EU) and the *European Economic Area* (EEA). Its implementation in May 2018 led many organizations worldwide to post or update privacy statements regarding how they handle information collected in the course of business. Such statements tend to be long and include carefully crafted legal language. We realize that we may need to provide similar language on our website and in the printed edition, but until such a statement has been developed here is an explanation of how we use any information you have supplied relating to your subscription:

- The mailing list for *The Internet Protocol Journal* (IPJ) is entirely "opt in." We never have and never will use mailing lists from other organizations for any purpose.
- You may unsubscribe at any time using our online subscription system or by contacting us via e-mail. We will honor any request to remove your name and contact information from our database.
- We will use your contact information only to communicate with you about your subscription; for example, to inform you that a new issue is available, that your subscription needs to be renewed, or that your printed copy has been returned to us as undeliverable by the postal authorities.
- We will never use your contact information for any other purpose or provide the subscription list to any third party other than for the purpose of distributing IPJ by post or by electronic means.
- If you make a donation in support of the journal, your name will be listed on our website and in print unless you tell us otherwise.

Thank You!

Publication of IPJ is made possible by organizations and individuals around the world dedicated to the design, growth, evolution, and operation of the global Internet and private networks built on the Internet Protocol. The following individuals have provided support to IPJ. You can join them by visiting **http://tinyurl.com/IPJ-donate**

Fabrizio Accatino
Michael Achola
Martin Adkins
Christopher Affleck
Scott Aitken
Jacobus Akkerhuis
Antonio Cuñat Alario
Nicola Altan
Matteo D'Ambrosio
Jens Andersson
Danish Ansari
Finn Arildsen
Tim Armstrong
Richard Artes
Michael Aschwanden
David Atkins
Jac Backus
Jaime Badua
Bent Bagger
Eric Baker
Santosh Balagopalan
Michael Bazarewsky
David Belson
Hidde Beumer
Pier Paolo Biagi
John Bigrow
Orvar Ari Bjarnason
Axel Boeger
Keith Bogart
Mirko Bonadei
Roberto Bonalumi
Julie Bottorff Photography
Gerry Boudreaux
L de Braal
Kevin Breit
Thomas Bridge
Ilia Bromberg
Václav Brožík
Christophe Brun
Gareth Bryan
Stefan Buckmann
Caner Budakoglu
Darrell Budic
Scott Burleigh
Jon Harald Bøvre
Olivier Cahagne
Antoine Camerlo
Tracy Camp
Ignacio Soto Campos
Fabio Caneparo
Roberto Canonico
David Cardwell
John Cavanaugh
Lj Cemeras
Dave Chapman
Stefanos Charchalakis
Greg Chisholm
David Chosrova
Marcin Cieslak
Brad Clark
Narelle Clark
Joseph Connolly
Steve Corbató
Brian Courtney
Dave Crocker
Kevin Croes
John Curran
André Danthine
Morgan Davis
Jeff Day
Julien Dhallenne
Freek Dijkstra
Geert Van Dijk
David Dillow
Richard Dodsworth
Ernesto Doelling
Michael Dolan
Eugene Doroniuk
Karlheinz Dölger
Joshua Dreier
Lutz Drink
Andrew Dul
Joan Marc Riera Duocastella
Holger Durer
Mark Eanes
Peter Robert Egli
George Ehlers
Peter Eisses
Torbjörn Eklöv
Y Ertur
ERNW GmbH
ESdatCo
Steve Esquivel
Jay Etchings
Mikhail Evstiounin
Paul Ferguson
Ricardo Ferreira
Kent Fichtner
Michael Fiumano
The Flirble Organisation
Gary Ford
Jean-Pierre Forcioli
Susan Forney
Christopher Forsyth
Andrew Fox
Craig Fox
Fausto Franceschini
Valerie Fronczak
Tomislav Futivic
Edward Gallagher
Andrew Gallo
Chris Gamboni
Xosé Bravo Garcia
Osvaldo Gazzaniga
Kevin Gee
Greg Giessow
John Gilbert
Serge Van Ginderachter
Greg Goddard
Tiago Goncalves
Octavio Alfageme Gorostiaga
Barry Greene
Jeffrey Greene
Richard Gregor
Martijn Groenleer
Geert Jan de Groot
Christopher Guemez
Gulf Coast Shots
Sheryll de Guzman
Rex Hale
Jason Hall
James Hamilton
Stephen Hanna
Martin Hannigan
John Hardin
David Harper
Edward Hauser
David Hauweele
Marilyn Hay
Headcrafts SRLS
Hidde van der Heide
Johan Helsingius
Robert Hinden
Asbjorn Hojmark
Damien Holloway
Alain Van Hoof
Edward Hotard
Bill Huber
Hagen Hultzsch
Kevin Iddles
Mika Ilvesmaki
Karsten Iwen
David Jaffe
Ashford Jaggernauth
Martijn Jansen
Jozef Janitor
John Jarvis
Dennis Jennings
Edward Jennings
Aart Jochem
Brian Johnson
Curtis Johnson
Richard Johnson
Jim Johnston
Jonatan Jonasson
Daniel Jones
Gary Jones
Jerry Jones
Anders Marius Jørgensen
Amar Joshi
David Jump
Merike Kaeo
Andrew Kaiser
Christos Karayiannis
David Kekar
Jithin Kesavan
Jubal Kessler
Shan Ali Khan
Nabeel Khatri
Dae Young Kim
William W. H. Kimandu
John King
Russell Kirk
Gary Klesk
Anthony Klopp
Henry Kluge
Michael Kluk
Andrew Koch
Ia Kochiashvili
Carsten Koempe
Richard Koene
Alexader Kogan
Antonin Kral
Robert Krejčí
Mathias Körber
John Kristoff
Terje Krogdahl
Bobby Krupczak
Murray Kucherawy
Warren Kumari
George Kuo
Dirk Kurfuerst
Darrell Lack
Yan Landriault
Markus Langenmair
Fred Langham
Andrew Lamb
Richard Lamb
Sig Lange
Tracy LaQuey Parker
Rick van Leeuwen
Simon Leinen
Robert Lewis
Christian Liberale
Martin Lillepuu
Roger Lindholm
Sergio Loreti
Eric Louie
Guillermo a Loyola
Hannes Lubich
Dan Lynch
Sanya Madan
Miroslav Madić
Alexis Madriz
Carl Malamud
Jonathan Maldonado
Michael Malik
Yogesh Mangar
Bill Manning
Harold March
Vincent Marchand
Gabriel Marroquin
David Martin
Jim Martin
Ruben Tripiana Martin

Timothy Martin
Carles Mateu
Juan Jose Marin Martinez
Ioan Maxim
David Mazel
Miles McCredie
Brian McCullough
Joe McEachern
Alexander McKenzie
Jay McMaster
Mark Mc Nicholas
Carsten Melberg
Kevin Menezes
Bart Jan Menkveld
William Mills
David Millsom
Desiree Miloshevic
Joost van der Minnen
Thomas Mino
Rob Minshall
Wijnand Modderman
Mohammad Moghaddas
Charles Monson
Andrea Montefusco
Fernando Montenegro
Joel Moore
John More
Maurizio Moroni
Brian Mort
Soenke Mumm
Tariq Mustafa
Stuart Nadin
Michel Nakhla
Mazdak Rajabi Nasab
Krishna Natarajan
Naveen Nathan
Darryl Newman
Thomas Nikolajsen
Paul Nikolich
Travis Northrup
Marijana Novakovic
David Oates
Ovidiu Obersterescu
Tim O'Brien
Mike O'Connor
Mike O'Dell
John O'Neill
Jim Oplotnik
Packet Consulting Limited
Carlos Astor Araujo Palmeira
Alexis Panagopoulos
Gaurav Panwar
Manuel Uruena Pascual
Ricardo Patara
Dipesh Patel
Alex Parkinson
Craig Partridge
Dan Paynter
Leif Eric Pedersen
Rui Sao Pedro
Juan Pena
Chris Perkins
Michael Petry
Alexander Peuchert
David Phelan
Derrell Piper
Rob Pirnie
Marc Vives Piza
Jorge Ivan Pincay Ponce
Victoria Poncini
Blahoslav Popela
Eduard Llull Pou
Tim Pozar
David Raistrick
Priyan R Rajeevan
Balaji Rajendran
Paul Rathbone
William Rawlings
Bill Reid
Petr Rejhon
Robert Remenyi
Rodrigo Ribeiro
Glenn Ricart
Justin Richards
Mark Risinger
Gregory Robinson
Ron Rockrohr
Carlos Rodrigues
Magnus Romedahl
Lex Van Roon
Alessandra Rosi
David Ross
William Ross
Boudhayan Roychowdhury
Carlos Rubio
Timo Ruiter
RustedMusic
Babak Saberi
George Sadowsky
Scott Sandefur
Sachin Sapkal
Arturas Satkovskis
PS Saunders
Richard Savoy
John Sayer
Phil Scarr
Elizabeth Scheid
Jeroen Van Ingen Schenau
Carsten Scherb
Ernest Schirmer
Philip Schneck
Dan Schrenk
Richard Schultz
Timothy Schwab
Roger Schwartz
SeenThere
Scott Seifel
Yury Shefer
Yaron Sheffer
Doron Shikmoni
Tj Shumway
Jeffrey Sicuranza
Thorsten Sideboard
Greipur Sigurdsson
Andrew Simmons
Pradeep Singh
Henry Sinnreich
Geoff Sisson
Helge Skrivervik
Darren Sleeth
Richard Smit
Bob Smith
Courtney Smith
Eric Smith
Mark Smith
Craig Snell
Job Snijders
Ronald Solano
Asit Som
Ignacio Soto Campos
Evandro Sousa
Peter Spekreijse
Thayumanavan Sridhar
Paul Stancik
Ralf Stempfer
Matthew Stenberg
Adrian Stevens
Clinton Stevens
John Streck
Martin Streule
Viktor Sudakov
Edward-W. Suor
Vincent Surillo
T2Group
Roman Tarasov
David Theese
Douglas Thompson
Lorin J Thompson
Joseph Toste
Rey Tucker
Sandro Tumini
Angelo Turetta
Phil Tweedie
Steve Ulrich
Unitek Engineering AG
John Urbanek
Martin Urwaleck
Betsy Vanderpool
Surendran Vangadasalam
Ramnath Vasudha
Philip Venables
Buddy Venne
Alejandro Vennera
Luca Ventura
Tom Vest
Dario Vitali
Michael L Wahrman
Laurence Walker
Randy Watts
Andrew Webster
Tim Weil
Jd Wegner
Westmoreland Engineering Inc.
Rick Wesson
Peter Whimp
Russ White
Jurrien Wijlhuizen
Derick Winkworth
Pindar Wong
Phillip Yialeloglou
Janko Zavernik
Muhammad Ziad Ziayuddin
Jose Zumalave
Romeo Zwart
Bernd Zeimetz
廖 明沂.

Call for Papers

The *Internet Protocol Journal* (IPJ) is a quarterly technical publication containing tutorial articles ("What is...?") as well as implementation/operation articles ("How to..."). The journal provides articles about all aspects of Internet technology. IPJ is not intended to promote any specific products or services, but rather is intended to serve as an informational and educational resource for engineering professionals involved in the design, development, and operation of public and private internets and intranets. In addition to feature-length articles, IPJ contains technical updates, book reviews, announcements, opinion columns, and letters to the Editor. Topics include but are not limited to:

- Access and infrastructure technologies such as: Wi-Fi, Gigabit Ethernet, SONET, xDSL, cable, fiber optics, satellite, and mobile wireless.
- Transport and interconnection functions such as: switching, routing, tunneling, protocol transition, multicast, and performance.
- Network management, administration, and security issues, including: authentication, privacy, encryption, monitoring, firewalls, troubleshooting, and mapping.
- Value-added systems and services such as: Virtual Private Networks, resource location, caching, client/server systems, distributed systems, cloud computing, and quality of service.
- Application and end-user issues such as: E-mail, Web authoring, server technologies and systems, electronic commerce, and application management.
- Legal, policy, regulatory and governance topics such as: copyright, content control, content liability, settlement charges, resource allocation, and trademark disputes in the context of internetworking.

IPJ will pay a stipend of US$1000 for published, feature-length articles. For further information regarding article submissions, please contact Ole J. Jacobsen, Editor and Publisher. Ole can be reached at **ole@protocoljournal.org** or **olejacobsen@me.com**

Supporters and Sponsors

Supporters

Internet Society

cisco

Diamond Sponsors

Google

Ruby Sponsors

ICANN

Sapphire Sponsors

Your logo here!

Emerald Sponsors

Afilias

Akamai

APNIC

APRICOT
Asia Pacific Regional Internet Conference on Operational Technologies

COMCAST

DE-CIX

EQUINIX

JPRS
JAPAN REGISTRY SERVICES

Juniper Networks

lacnic

LinkedIn

linx

NSRC
Network Startup Resource Center

NTT Communications

RIPE NCC
RIPE NETWORK COORDINATION CENTRE

TEAM CYMRU

VERISIGN

WIDE
PROJECT

Corporate Subscriptions

AFRINIC
The Internet Numbers Registry for Africa

amsix
amsterdam internet exchange

DNS-OARC

IWL

ISC
Internet Systems Consortium

Limelight Networks

PKNIC

qa cafe

SIDN

For more information about sponsorship, please contact `sponsor@protocoljournal.org`

The Internet Protocol Journal
Link Fulfillment
7650 Marathon Dr., Suite E
Livermore, CA 94550

CHANGE SERVICE REQUESTED

The Internet Protocol Journal is published quarterly and supported by the Internet Society and other organizations and individuals around the world dedicated to the design, growth, evolution, and operation of the global Internet and private networks built on the Internet Protocol.

Email: ipj@protocoljournal.org
Web: www.protocoljournal.org

Printed in the USA on recycled paper.