The Internet Protocol Journal

March 2019 Volume 22, 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

The *Transmission Control Protocol* (TCP) is a core component of the Internet Protocol Suite. TCP has proven robust and flexible in the face of changing network infrastructures, but may not be the most efficient way to retrieve the many components of today's complex web pages. The *Quick UDP Internet Connection* (QUIC) protocol is an alternative to TCP for web traffic. QUIC was initially developed and deployed by Google and is now being standardized in the *Internet Engineering Task Force* (IETF). In our first article, Geoff Huston examines the motivations for QUIC and describes the protocol and its implementation.

According to *Wikipedia:* "A *checksum* is a small-sized datum derived from a block of digital data for the purpose of detecting errors that may have been introduced during its transmission or storage. It is usually applied to an installation file after it is received from the download server. By themselves, checksums are often used to verify data integrity but are not relied upon to verify data authenticity." In preparation for the "rolling" of the root *Key Signing Key* of the *Domain Name System* (DNS), tests were developed to create so-called *key-tags*. This key-tag generation process "...became an adventure in itself that included beautiful discrete math, flawed functions, carefully crafted primes, multiple cryptographic libraries, and some brilliant people," according to Roy Arends, author of our second article, "The Quest for the Missing Checksums." IPJ doesn't normally delve into complex mathematics, but in this case the interplay of various software libraries and methods provides some valuable lessons for anyone involved in code generation and testing.

We would like to remind you that this journal depends on the generous support of numerous individuals and organizations. If you would like to help support IPJ, please contact us for further details. Comments, suggestions, book reviews, and articles are always welcome. Send your messages to `ipj@protocoljournal.org`

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

A Quick Look at QUIC

by Geoff Huston, APNIC

Q*uick UDP Internet Connection* (QUIC) is a network protocol initially developed and deployed by Google, and is now being standardized in the *Internet Engineering Task Force* (IETF). In this article we'll take a quick tour of QUIC, looking at the goals that influenced its design, and the implications QUIC might have on the overall architecture of the Internet Protocol Stack.

QUIC is not exactly a recent protocol, as the concept appears to have been developed by Google in 2012, and initial public releases of this protocol were included in Chromium version 29, released in August 2013. QUIC is one of many transport-layer network protocols that attempt to refine the basic operation of the *Transmission Control Protocol* (TCP).

Why are we even thinking about refining TCP?

TCP is now used in billions of devices and is perhaps the most widely adopted network transport protocol that we've witnessed so far. If this protocol weren't fit for our use, then we would have moved on and adopted some other protocol or protocols instead. Part of the reason for the broad adoption of TCP is its incredible flexibility. The protocol can support a diverse variety of uses, from micro-exchanges to gigabyte data movement, transmission speeds that vary from hundreds of bits per second to tens and possibly hundreds of gigabits per second. TCP is the workhorse of the Internet. But even so, there is room for refinement. TCP is used in many different ways, and its design represents a set of trade-offs that attempt to be a reasonable fit for many purposes but not necessarily an ideal fit for any particular one.

One of the aspects of the original design of the Internet Protocol Suite was that of elegant brevity and simplicity. The specification of TCP[1] is not a single profile of behavior that has been cast into a fixed form that was chiseled into the granite slab of a rigid standard. TCP is malleable in many important ways. Numerous efforts over the years have shown that it is possible to stay within the standard definition of TCP, in that all the packets in a session use the standard TCP header fields in mostly conventional ways, but also to create TCP implementations that behave radically differently from each other. Critically, the TCP standard does not strictly define how the sender can control the amount of data in flight across the network. There is a convention to adopt an approach of slowly increasing the amount of data in flight while there are no visible errors in the data transfer (as shown by the stream of received acknowledgement [ACK] packets) and quickly responding to signals of network congestion (packet drop, as shown by duplicate acknowledgements) by rapidly decreasing the sending rate.

Variants of TCP use different controls to manage this "slow increase" and "rapid drop" behavior[2] and may also use different signals to control this data flow. These signals include measurements of end-to-end delay, or inter-packet jitter (such as the recently published *Bottleneck Bandwidth and Round-trip Propagation Time* (BBR) protocol[3]). All of these variants still manage to fit with the broad parameters of what is conventionally called TCP.

It is also useful to understand that most variants of TCP need to be implemented only on the data sender (the "server" in a client/server environment). The common assumption of all TCP implementations is that clients will send a TCP ACK packet on successful receipt of both in-sequence and out-of-sequence data. It is left to the server's TCP engine to determine how the received ACK stream will be applied to its internal model of network capability and how it will modify its subsequent sending rate accordingly. The implication is that deployment of new variants of TCP flow control is essentially based on deployment within service-delivery platforms and does not necessarily imply changing the TCP implementations in all the billions of clients. This feature also contributes to the flexibility of TCP.

But despite its considerable flexibility, TCP has its problems, particularly with web-based services. These days most web pages are not simple monolithic objects. They typically contain many separate components, including images, scripts, customized frames, and others. Each of these is a separate web "object," and if you are using a browser that is equipped with the original implementation of the *HyperText Transfer Protocol* (HTTP) each object will be loaded in a new TCP session, even if the objects are served from the same IP address. The overheads of setting up both a new TCP session and a new *Transport Layer Security* (TLS)[4] session for each distinct web object within a compound web resource can become quite significant, and the temptation to reuse an already established TLS session is close to overwhelming. But this approach of multiplexing a number of data streams within a single TCP session also has issues. Multiplexing multiple logical data flows across a single session can generate unwanted interdependencies between the flow processors and generate *Head of Line Blocking* situations. It appears that while it makes some logical sense to share a single end-to-end security association and a rate-controlled data-flow state across a network across multiple logical data flows, TCP represents a rather poor way of achieving this outcome. The conclusion is that if we want to improve the efficiency of such compound transactions by introducing parallel behaviors into the protocol, we need to look beyond TCP.

Why not just start afresh and define a new transport protocol that addresses these shortcomings of TCP? The answer is simple: *Network Address Translators* (NATs)!

NATs and Transport Protocols

The original design of IP allowed for a clear separation between the network element that allowed the network to accept an IP packet and forward it onto its intended destination (the "Internet" part of the IP protocol suite) and the end-to-end transport protocol that enabled two applications to communication via some form of "session." The transport protocol field in the IPv4 packet header and the Next header field of the IPv6 packet header uses an 8-bit field to identify the end-to-end protocol. This design assumed that the network had no need to "understand" what end-to-end protocol was being used within a packet. Ideally an IP packet switch will not differentiate in its treatment of packets depending on the inner end-to-end protocol.

Some 140 protocols are listed in the IP protocol field registry[5]. TCP and the *User Datagram Protocol* (UDP) are just two of these protocols (protocol values 6 and 17, respectively). In theory at any rate, there is room for a least 100 more. However, in the public Internet the story is somewhat different. TCP and UDP are widely accepted protocols, and the *Internet Control Message Protocol* (ICMP) (protocol 2) is generally accepted, but little else. How did this situation happen?

NATs changed the assumption about network devices not looking inside the packet (to be precise, port-translating NATs changed that assumption). NATs are network devices that look inside the IP packet and re-write the port addresses used by TCP and UDP[6]. What if an IP packet contains an end-to-end transport protocol identifier value that is neither TCP nor UDP? Most NATs will simply drop the packet, on the basis of a security paradigm that "what you don't recognize is likely to be harmful." The pragmatic result is that NATs have limited the choice of transport protocols of an application in the public Internet to just two: TCP and UDP.

If the aim is to deploy a new transport protocol—but not confuse active network elements that are expecting to see a conventional TCP or UDP header—then how can we achieve this goal?

This question was the challenge of the QUIC developers.

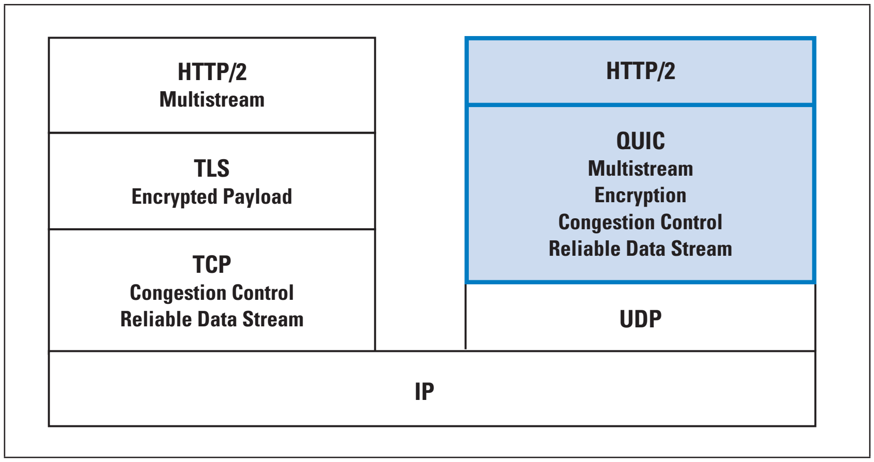
QUIC over UDP

The solution that QUIC chose was a UDP-based approach. UDP is a minimal framing protocol that allows an application to access the basic datagram services that IP offers. Apart from the source and destination port numbers, the UDP header adds a length header and a checksum that covers the UDP header and UDP payload. It is essentially an abstraction of the underlying datagram IP model with just enough additional information to allow an IP protocol stack to direct an incoming packet to an application that has bound itself to a nominated UDP port address. If TCP is an overlay across the underlying IP datagram service, then it's a small step to think about layering TCP as a payload within a UDP packet.

Using our standard Internet model, QUIC is—strictly speaking—a datagram transport application. An application that uses the QUIC protocol sends and receives packets using UDP port 443.

Technically, this change is very small to an IP packet, adding just 8 bytes to the IP packet by placing a UDP header between the IP and TCP packet headers (Figure 1). The implications of this change are far more significant than these 8 bytes would suggest. However, before we consider these implications, let's look at some QUIC services.

Figure 1: The QUIC Protocol Architecture



QUIC and the Connection ID

If the choice of UDP as the visible end-to-end protocol for QUIC was a choice dictated by the inflexibility of the base of deployed NAT devices in the public Internet and their collective inability to accommodate new protocols, the way that NATs handle UDP packets has further implications for QUIC.

NATs maintain a *translation table*. In the most general model, a NAT takes the 5-tuple of incoming packets, using the destination and source IP addresses, the destination and source port addresses, and the protocol field, and performs a lookup into the table to find the associated translated fields. The address headers of the packet are rewritten to these new values, *checksums* are recomputed, and the packet is passed onward. Certain NAT implementations may use variants of this model. For example, some NATs use only the source IP address and port address on *outbound* packets as the lookup key, and the corresponding destination IP address and port address in *incoming* packets.

Typically, the NAT generates a new translation table entry when a triggering packet is passed from the *inside* to the *outside* and subsequently removes the table entry when the NAT assumes that the translation is no longer needed. For TCP sessions it is possible to maintain this translation table quite accurately.

New translation-table entries are created in response to *outbound* TCP SYN connection establishment packets and removed either when the NAT sees the TCP FIN exchange or in response to a TCP RST packet or when the session is idle for an extended period.

UDP packets do not have these clear packet exchanges to start and stop sessions, so NATs need to make some assumptions. Most NATs create a new translation table entry when they see an outbound UDP packet that has not matched any existing translation table. The entry is then maintained for some period of time (as determined by the NAT) and is then removed if there are no further packets that match the session signature. Even when there are further matching UDP packets, the NAT may use an overall UDP session timer and remove the NAT entry after some predetermined time interval.

For QUIC and NATs, this situation is a potential problem. The QUIC session is established between a QUIC server on UDP port 443 and the NAT-generated source address and port. However, at some point in the session lifetime the NAT may drop the translation-table entry, and the next outbound client packet will generate a new translation-table entry that may use a different source address and port. How can the QUIC server recognize that this next-received packet, with its new source address and source port number, is actually part of an existing QUIC session?

QUIC uses the concept of *Connection Identifiers* (Connection IDs). Each endpoint generates connection IDs that will allow received packets with that connection ID to be routed to the process that is using that connection ID. During QUIC version negotiation these connection IDs are exchanged, and thereafter each sent QUIC packet includes the current connection ID of the remote party.

This form of semantic distinction between the identity of a connection to an endpoint and the current IP address and port number that QUIC uses is similar to the *Host Identity Protocol* (HIP)[7]. This protocol also uses a constant endpoint identifier that allows a session to survive changes in the endpoint IP addresses and ports.

QUIC Streams

TCP provides the abstraction of a reliable order byte stream to applications. QUIC provides a similar abstraction to the application, termed within QUIC as *streams*. The essential difference here is that TCP implements a single behavior, while a single QUIC session can support multiple streams profiles.

Bidirectional streams place the client and server transactions into a matched context, as is required for the conventional request/response transactions of HTTP/1. A client would be expected to open a bidirectional stream with a server and then issue a request in a stream which would generate a matching response from the server. It is possible for a server to initiate a bidirectional *push stream* to a client, which contains a response without an initial request.

Control information is supported using unidirectional *control streams*, where one side can pass a message to the other as soon as they are able. An underlying *unidirectional stream* interface, used to support control streams, is also exposed to the application.

Not only can QUIC support many different stream profiles, it can also support different stream profiles within a single end-to-end QUIC session. This concept is not a novel one, of course, and the HTTP/2 protocol is a good example of an application-level protocol adding multiplexing and stream framing in order to carry multiple data flows across a single transport data stream. However, a single TCP transport stream as used by HTTP/2 may encounter *Head of Line Blocking* where all overlay data streams fate-share across a single TCP session. If one of the streams stalls, all overlay data streams could be affected and could stall as well.

QUIC allows for a slightly different form of multiplexing where each overlay data stream can use its own end-to-end flow state, and a pause in one overlay stream does not imply that any other simultaneous stream is affected.

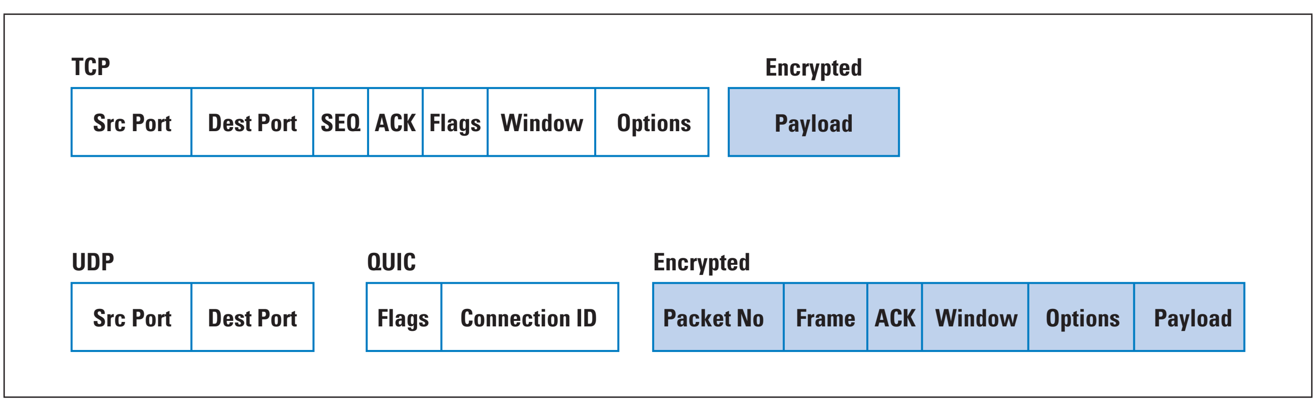
Part of the reason to multiplex multiple data flows between the same two endpoints in HTTP/2 was to reduce the overhead of setting up a TLS security association for each TCP session. This overhead can be quite significant when the individual streams are each sending a small object, and it's possible to encounter a situation where the TCP and TLS handshake component of a compound web object fetch dominates both the total download time and the data volume.

QUIC pushes the security association to the end-to-end state that is implemented as a UDP data flow, so that streams can be started in a very lightweight manner because they essentially reuse the established secure session state.

QUIC Encryption

As is probably clear from the references to TLS already, QUIC uses *end-to-end encryption*. This encryption is performed on the UDP payload, so once the TLS handshake is complete very little of the subsequent QUIC packet exchange is in the clear (Figure 2).

Figure 2: Comparison of TCP and TLS with QUIC



What is exposed in QUIC are the *public flags*. This initial part of a QUIC packet consists of the connection ID, which allows the receiver to associate the packet with an endpoint without decrypting the entire packet. The QUIC version is also part of the public flag set, which is used in the initial QUIC session establishment and can be omitted thereafter.

The remainder of the QUIC packet includes *private flags* and the payload. They are encrypted and are not directly visible to an eavesdropper. This private section includes the packet sequence number. This field is used to detect duplicate and missing packets. It also includes all the flow-control parameters, including window advertisements.

This encryption is one of the critical differences between TCP and QUIC. With TCP the control parts of the protocol are in the clear, so that a network element would be able to inspect the port addresses (and infer the application type), as well as the flow state of the connection. Connection of a sequence of such TCP packets, even if only looking at the packets flowing in one direction within the connection, would allow the network element to infer the round-trip time and the data-transmission rate. And, like a NAT, manipulation of the receive window in the ACK stream would allow a network element to apply a throttle to a connection and reduce the transfer rate in a manner that would be invisible to both endpoints. Placing all of this control information inside the encrypted part of the QUIC packet ensures that no network element has direct visibility to this information, and no network element can manipulate the connection flow.

One could take the view that QUIC enforces a perspective that was assumed in the 1980s: that the end-to-end transport protocol is not shared with the network. All the network "sees" are stateless datagrams, and the endpoints can safely assume that the information contained in the end-to-end transport control fields is carried over the network in a manner that protects it from third-party inspection and alteration.

QUIC and IP Fragmentation

The short answer is "no!" QUIC packets cannot be fragmented[7, 8]. The way this feature is achieved is by having the QUIC HELLO packet be padded out to the maximal packet size, and not completing the initial HELLO exchange if the maximally sized packet is fragmented.

For IPv4 the maximum QUIC packet size is 1,350 bytes. Adding 8 bytes for the UDP header, 20 bytes for IPv4, and 14 bytes for the Ethernet frame means that a QUIC packet on Ethernet totals 1,392 packets. There is no particular rationale for this choice of 1,350 other than the results of empirical testing on the public Internet.

For IPv6 the QUIC maximum packet size is reduced by 20 bytes to 1,330. The resultant Ethernet packet is still 1,392 bytes because of the larger IPv6 IP packet header.

What happens if the network path has a smaller *Maximum Transmission Unit* (MTU) than this value? The answer is in the next section.

QUIC and TCP

QUIC is not intended as a replacement for TCP. Indeed, QUIC relies on the continued availability of TCP.

Whenever QUIC encounters a fatal error—such as fragmentation of the QUIC HELLO packet—the intended response from QUIC is to shut down the connection. Since QUIC itself lies in the application space, not the kernel space, the client-side application can be directly informed of this closure of the QUIC connection and it can re-open a connection to the server using a conventional TCP transport protocol.

The implication is that QUIC does not necessarily have to have a robust response for all forms of behavior, and when QUIC encounters a state where it has no clear definition of the desired behavior, it is always an option to signal a QUIC failure to the application. The failure need not be fatal to the application, because such a signal can trigger the application to repeat the transaction using a conventional TCP session.

I can QUIC, do you?

Unlike all other TCP services that use a dedicated TCP port address to distinguish themselves from all other services, QUIC does not advertise itself in such a manner. That reality leaves numerous ways in which a server could potentially advertise itself as being accessible over QUIC.

One such possible path is the use of *Domain Name System* (DNS) *Service Records* (SRV)[9]. The SRV record can indicate the connection point for a named service using the name of the transport protocol and the protocol-specific service address. This usage may be an option for the future, but no such DNS service record has been defined for QUIC.

Instead, in keeping with the overall QUIC approach of loading up most of the service functionality into the application itself, a server that supports QUIC can signal its capability within HTTP itself. The way it signals is defined in an Internet standard for "Alternative Services"[10], which is a means to list alternative ways to access the same resources.

For example, the Google homepage, `www.google.com`, includes the HTTP header:

```
alt-svc: quic=":443"; ma=2592000; v="44,43,39"
```

This entry indicates that the same material is accessible using QUIC over port 443. The "`ma`" field is the time to keep this information on the local client, which in this case is 30 days, and the "`v`" field indicates that the server will negotiate QUIC versions 39, 43, and 44.

QUIC Lessons

QUIC is a rather forceful assertion that the Internet infrastructure is now heavily ossified and more highly constrained than ever. There is no room left for new transport protocols in today's network. If what you want to do can't be achieved within TCP, then all that's left is UDP.

The IP approach to packet-size adaptation through fragmentation was a powerful concept once upon a time. A sender did not need to be aware of the constraints that may apply on a path. Any network-level packet fragmentation and reassembly was invisible to the end-to-end packet transfer. This invisibility is no longer wise. Senders need to ensure that their packets can reach their intended destinations without any additional requirement for fragmentation handling.

Mutual trust is over. Applications no longer trust other applications. They don't trust the platform that hosts them or the shared libraries that implement essential functions. Applications no longer trust a network to keep their secrets. More and more functions and services are being pulled back into the application and are passed out from an application as much as possible in packets that are cloaked in a privacy shroud.

There is a tension between speed, security, and paranoia. An ideal outcome is one that is faster, private, and secure. Where it is not obvious and the inevitable trade-offs emerge, it seems that we have some minimum security and privacy requirements that simply must be achieved. But once we have achieved these minimum requirements, we are then happy to trade off incremental improvements in privacy and security for better session performance.

The traditional protocol-stack model was a convenient abstraction, not a design rule. Applications do not necessarily need to bind to transport-layer sockets provided by the underlying platform. Applications can implement their own end-to-end transport if necessary.

The infrastructure of the Internet might be heavily ossified, but the application space is seeing a new set of possibilities open up. Applications need not wait for the platform to include support for a particular transport protocol or await the deployment of a support library to support a particular name-resolution function. Applications can solve these issues for themselves directly. The gain in flexibility and agility is considerable.

There is a price to pay for this new-found agility, and that price is broad interoperability. Browsers that support QUIC can open up UDP connections to certain servers and run QUIC, but browsers cannot assume—as they do with TCP—that QUIC is a universal and interoperable lingua franca of the Internet. While QUIC is a fascinating adaptation with some very novel concepts, it is still an optional adaptation. For those clients and servers that do not support QUIC, or for network paths where UDP port 443 is not supported, the common fallback is TCP. The expansion of the Internet is inevitably accompanied by inertial bloat, and as we've seen with the extended saga of IPv6 deployment, it is a formidable expectation to think that the entire Internet will embrace a new technical innovation in a timeframe of months, years, or possibly even decades! That does not mean that we can't think new thoughts, and that we can't realize these new ideas into new services on the Internet. We certainly can, and QUIC is an eloquent demonstration of exactly how to craft innovation into a rather stolid and resistant underlying space.

Further Reading

QUIC has excited considerable interest over the past couple of years, and there are many posts to be found on the 'net. Here's a small sample of this online material that you may find to be of interest:

- A useful consideration of positive and negative aspects of QUIC are in Robin Marx's post "QUIC and HTTP/3: Too big to fail?"
  `https://calendar.perfplanet.com/2018/quic-and-http-3-too-big-to-fail/`
- A slightly older (2014) but useful technical overview of QUIC can be found in Shigeki Ohtsu's presentation to the HTTP/2 Conference Japan.
  `https://www.slideshare.net/shigeki_ohtsu/quic-overview`
- A commentary on Cloudflare's investigations with QUIC can be found in a recent blog post: "The Road to QUIC":
  `https://blog.cloudflare.com/the-road-to-quic/`
- A discussion of QUIC work in the IETF by Mark Nottingham, QUIC Working Group Co-Chair: "What's Happening with QUIC,"
  `https://www.ietf.org/blog/whats-happening-quic/?`

References

[1] Jon Postel, "Transmission Control Protocol," RFC 793, September 1981.

[2] Geoff Huston, "Faster," *The ISP Column*, June 2005.
`https://www.potaroo.net/ispcol/2005-06/faster.html`

[3] Neal Cardwell, Yuchuing Cheng, C. Stephen Gunn, Soheil Hasses Yeganeh, and Van Jacobson, "BBR: congestion-based congestion control," *Communications of the ACM*, Vol. 60, Issue 2, pp 58–66, February 2017.

[4] Eric Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.34," RFC 8446, August 2018.

[5] IANA Protocol Numbers Registry.
https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

[6] Geoff Huston, "Anatomy: A Look Inside Network Address Translators," *The Internet Protocol Journal,* Volume 7, No. 3, September 2004.

[7] Geoff Huston, "Fragmentation," *The Internet Protocol Journal,* Volume 19, No. 2, June 2016.

[8] Geoff Huston, "IPv6 and Packet Fragmentation," *The Internet Protocol Journal,* Volume 21, No. 1, April 2018.

[9] Arnt Gulbrandsen, Paul Vixie, and Levon Esibov, "A DNS RR for specifying the location of services (DNS SRV)," RFC 2782, February 2000.

[10] Mark Nottingham, Patrick McManus, and Julian Reschke, "HTTP Alternative Services," RFC 7838, April 2016.

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**

The Quest for the Missing Checksums

by Roy Arends, ICANN

The *Domain Name System* (DNS) is a hierarchical namespace that provides a method to look up Internet identifiers such as IP addresses using easy-to-remember domain names. This hierarchy starts at the root,[0] where the actual namespace is delegated to several registries. The data at the root is signed with cryptographic keys, using *Domain Name System Security Extensions* (DNSSEC)[1, 2, 3]. These cryptographic keys are replaced over time.

In an effort to change the top cryptographic key for the DNS, the so-called root *Key Signing Key*[4], several testbeds were created to emulate the process in a lab environment. In those testbeds, the actual root DNS keys are not used since the testbed operators do not have control of the private keys; rather keys of the same size using the same cryptographic algorithms and functions are generated. Apart from the fact that the key material is different, this emulated root zone cannot be distinguished from the real root zone.

This effort to generate certain cryptographic keys became an adventure in itself that included beautiful discrete math, flawed functions, carefully crafted primes, multiple cryptographic libraries, and some brilliant people.

The result of this effort shows that using an ancient checksum function to identify cryptographic keys is not optimal.

The problem

DNSSEC protects the DNS. To be precise, it protects validating resolvers' caches. DNSSEC uses cryptographic keys to validate signatures, and these signatures contain a *key-tag* that helps to identify which key to use. This key-tag is merely a hint; it doesn't have to be collision-free, and the function to generate it is similar to an IP header checksum (the difference between the two functions is that the key-tag function does not include a final end-around carry).

Technically, a key-tag is a 16-bit unsigned value. For our testbed, to clearly identify which keys were introduced in what year, the idea was to generate some vanity key-tags with the year in them; that is, "2010" for a key that was introduced in 2010, and "2015" for a key introduced in 2015. One way to generate those key-tags is to simply generate all possible key-tags in order to pick the desired ones. This process can be done by repeatedly generating a single key. Since the key-tag is based on the contents of the key, and since the contents of the key contain a lot of random bits, it was assumed that the resulting key-tag would be as random as the key.

After the process to generate keys ran long enough, the expectation was to have 65,536 keys—one for each tag. Surprisingly, it was possible to generate only 16,387 keys with unique tags, even after generating millions of keys. Specifically, the key-tags "2010" and "2015" were not included. It turns out that key-tag "2015" was excluded for a different reason than why key-tag "2010" was excluded!

Is it the software?

In order to track down this non-intuitive result, suspicion first fell on the software used to generate the keys. The *BIND* software package from *Internet Systems Consortium, Inc.* (ISC) has a command-line tool named *dnssec-keygen*. The convention it uses is to embed the key-tag in the filename. When a new key is generated, *dnssec-keygen* checks to determine if a key with a certain tag already exists to avoid overwriting it.

The *Flags* field in a DNSSEC key influences the value of the key-tag. For instance, if a key is revoked in the future, the "REVOKE" flag is set and that changes the value of the key-tag. To make sure that a new key-tag doesn't collide with any existing key, *dnssec-keygen* checks if a new key-tag (and its revoked equivalent) matches an existing key-tag (and its revoked equivalent as well). Initially, it was thought that this key-tag collision check was the culprit.

Since those vanity key-tags were still desired, and since revoked equivalents of keys with the 2010 and 2015 key-tag would not collide with any existing key-tags, it was decided to try to work around this specific check.

One way to avoid this check is to simply use another tool. The LDNS library from NLNetLabs comes with a set of examples. One of these examples is a utility named *ldns-keygen,* which produces DNSSEC keys and does not have the key-tag collision check to protect against accidentally overwriting an existing key. However, after generating millions of keys again, it too generated about 16,384 keys.

The two software tools used have no authors in common, but they do share a cryptographic library: *OpenSSL*. Both pieces of software independently had the limitation of producing only a subset of all possible key-tags. Both used a well-known, widely used cryptographic library. At this discovery the worrying started. If it is the library, and the tags are not distributed evenly, is the quality of the entropy in question? Does the library have any bugs?

To make sure this anomaly was not user error, different versions of OpenSSL were tested. Additionally, different entropy sources were used, and lastly, different key sizes were tried. Still, the same number of key-tags was generated.

Is it the library?

The folks on DNS-OARC's operations list came to the rescue. Peter van Dijk from PowerDNS used the PowerDNS management tool: *pdnsutil add-zone-key,* and was able to generate 32,769 unique key-tags. More key-tags than before, but still only about 50% of all possibilities. The tools in PowerDNS, BIND, and LDNS do not share any code or any authors. All three tools were written "from scratch." Additionally, PowerDNS does not use OpenSSL at all; rather it uses *mbedTLS,* a different cryptographic library. That means a problem related solely to the cryptographic libraries or the tools can be ruled out. There was still the observation that *pdnsutil* was able to produce twice as many key-tags as the other tools, but we'll get to that later.

Is it the checksum algorithm?

The next step was testing the key-tag function in RFC 4034[5]. The key-tag function is very similar to the radix-minus-one complement function for the *Internet Header Checksum*—a radix-minus-one complement function. Note that it is not exactly the same, but the minor difference could not fundamentally reduce the possible number of key-tags.

To test this possibility, a loop was created that fed random numbers into the key-tag algorithm. When using 2,048-bit random numbers as the input (instead of cryptographic keys), all possible key-tags could be produced in a short amount of time. This experiment ruled out that the limiting part was the key-tag algorithm itself. However, we'll come back to that later as well.

Is it purely a math problem?

Meanwhile, Florian Maury and Jérôme Plût from ANSSI took a good look at the problem and discovered it was none of the possibilities mentioned previously. It turns out that an interesting combination of the properties of the Internet Header Checksum and RSA moduli rules out certain results.

The input to the Internet Header Checksum function is treated as blocks of 16 bits and the output is a 16-bit checksum. Radix-minus-one complement methods are as old as accounting itself. The nine's complement method (where the radix is base 10) was used in Pascal's calculator. The method of complements is a technique used to subtract one number from another using only addition of positive numbers. We're not using the complements part here, only the part where we add, with carry, a bunch of bits.

A description of the Internet Header Checksum function follows: Add the 16-bit values with end-around carry; that is, if adding two 16-bit values results in a carry, then add that carry bit to the result of the addition.

Following is the end-around-carry part of the checksum function:

```
($sum AND 65535) + ($sum >> 16)
```

What Jérôme Plût observed is that this expression can be reduced to:

```
$sum mod 65535
```

Since modular arithmetic has the addition property, we can also deduce:

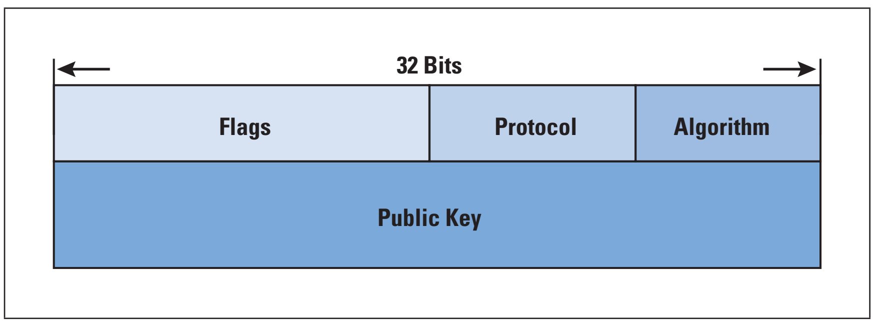
```
($Value1 + $Value2) mod 65535
```

or:

```
($value1 mod 65535) + ($value2 mod 65535)
```

Calculating a key-tag

As said earlier, the Internet Header Checksum is very similar to the key-tag function. The input for this key-tag algorithm is the RDATA part of a DNSKEY record:



For all keys generated in this exercise, all the fields remain the same, except for the modulus in the *Public Key* field.

For a *Key Signing Key*, the value of the *Flags* field is 257, and the *Protocol* field always has the value of 3. The *Algorithm* field has the value 8 (RSASHA256)[7]. With those parameters, the *Public Key* field consists of an *Exponent* and a *Modulus*. For this exercise, the exponent has value 65537 and is preceded with an *Exponent Length* field (value 3).

The constant part of this input can now simply be added up as a series of 16-bit unsigned values:

```
$value1 = Flags + Protocol*256 + Algorithm + ExpLen*256 + Exponent

$value1 = 257 + 3*256 + 8 + 3*256 + 65537

$value1 = 67338
```

Using the deduction from before:

```
keytag = (value1 mod 65535) + (value2 mod 65535)

keytag = (67338 mod 65535) + (value2 mod 65535)

keytag = 1803 + (value2 mod 65535)
```

The part of the checksum that is not constant is the *RSA-modulus*. The RSA-modulus is a composite number with two very large prime factors. In the previous equation, value2 is the RSA-modulus. The last substitution becomes:

```
keytag = 1803 + (RSA-modulus mod 65535)
```

Since the value 1803 is constant, it has no influence on the number of possible key-tags, hence the solution to the reduced set of possible key-tags may be found in the RSA-modulus modulo 65535 part of the equation.

Number theory

What Jérôme Plût observed is that the value 65535 is a composite number with four prime factors: 3, 5, 17, and 257. Since the RSA-modulus and 65535 do not share any factors, the RSA-modulus can't be congruent with 0 modulo 65535.

Therefore, the modulus is not congruent with 0 modulo 3, 0 modulo 5, 0 modulo 17, or 0 modulo 257.

All other congruence values are possible, so the set of possible values is simply a combination of the possible values:

```
2 * 4 * 16 * 256 = 32768.
```

We can now check if we indeed can't have 2010 as a value:

Before, we noted that:

```
keytag = 1803 + (RSA-modulus mod 65535)
```

We can now substitute the key-tag with our desired value:

```
2010 = 1803 + (RSA-modulus mod 65535)
2010 - 1803 = RSA-modulus mod 65535
 207 = RSA-modulus mod 65535
```

However, 207 is congruent with 0 modulo 3, meaning that in order for 207 to be possible, the RSA-modulus must have 3 as a factor. We know this is not the case, so 2010 (that is, 207 + 1803) can't be a key-tag.

The remainder of the problem

Remember that the first exercise led to 16,387 key-tags, not 32,768 as predicted before, or 32,769 as found by Peter van Dijk. Additionally, 32,769 is not 32,768 (and 16,387 is not 16,384, half of the 32,768 space).

32,769 is not 32,768

The key-tag function is similar to the Internet Header Checksum, but not the same. The crucial difference is the last end-around carry.

The last part of the key-tag function is defined in RFC 4034, and reads as follows:

```
ac += (ac >> 16) & 0xFFFF;
return ac & 0xFFFF;
```

The first line adds the carry bits to the accumulator. As a result, the accumulator might be a value larger than fits in a 16-bit value. Instead of again adding the carry bits to the value, it ignores those.

Ignoring the carry bits can, in some cases, result in an off by one value, compared to the Internet Header Checksum. With the Internet Header Checksum, only 32,768 values are possible, as we've seen in the previous section. Since the key-tag function might be off by one, a few more key-tag values are possible.

16,387 is not 32,769

Why was Peter able to produce about twice as many key-tags? Assuming that the values could have been 16,384 and 32,768 (as explained before), the only remaining difference is the library used.

OpenSSL generates primes that are congruent with 2 modulo 3. The resulting modulus is thus always congruent with 1 modulo 3, since:

```
(2 modulo 3) * (2 modulo 3) =
4 modulo 3 =
1 modulo 3
```

This formula reduces the possible key-tag space from 2 * 4 * 16 * 256 to 1 * 4 * 16 * 256, which is 16384.

This reduction is the reason why it was not possible to generate a key-tag with the value 2015. Using the same reduction as before, we can now substitute key-tag with 2015:

```
2015 = 1803 + (RSA-modulus mod 65535)
2015 - 1803 = RSA-modulus mod 65535
 212 = RSA-modulus mod 65535
```

However, 212 is congruent with 2 modulo 3. We now know that RSA moduli from OpenSSL are always congruent with 1 modulo 3, so key-tag 2015 is simply not possible when using OpenSSL.

The library that Peter is using, *mbedTLS*, does generate primes that are congruent with 1 modulo 3.

Conclusion

The limited key-tag space does not present a security issue. The key-tag is merely a hint and it is well known that different cryptographic keys may lead to the same key-tag. However, the decision to use a checksum as an identifier is poor at best. A checksum is designed to check if an error exists in data, and not, in general, designed to be an identifier. Additionally, using a function that is nearly identical to the well-known Internet Header Checksum seems to be an error in the design stage.

Acknowledgements

I cannot begin to thank adequately those who helped me to understand and explain the various compounding issues that resulted in the absence of 75% of all possible key-tags. Florian Maury and Jérôme Plût from ANSSI explained the core issue with the Internet Header Checksum over RSA moduli. Without them, I would still be searching in the dark. Peter van Dijk and Bert Hubert of PowerDNS consumed uncountable electrons and brainwaves to reproduce my findings with different tools and libraries. Google's Ben Laurie held my hand while I was drowning in modular arithmetic and brought me ashore. Finally, it was ICANN's David Conrad who made my broken English and various grammar faux pas readable.

References and Further Reading

[0] Geoff Huston, "The Root of the Domain Name System," *The Internet Protocol Journal,* Volume 20, No. 2, June 2017.

[1] Miek Gieben, "DNSSEC: The Protocol, Deployment, and a Bit of Development," *The Internet Protocol Journal,* Volume 7, No. 2, June 2004.

[2] Donald E. Eastlake 3rd, "Domain Name System Security Extensions," RFC 2535, March 1999.

[3] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends, "DNS Security Introduction and Requirements," RFC 4033, March 2005.

[4] George Michaelson, Patrick Wallström, Roy Arends, and Geoff Huston, "Rolling Over DNS Keys," *The Internet Protocol Journal,* Volume 13, No. 1, March 2010.

[5] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends, "Resource Records for the DNS Security Extensions," RFC 4034, March 2005.

[6] Scott Rose, Matt Larson, Dan Massey, Rob Austein, and Roy Arends, "Protocol Modifications for the DNS Security Extensions," RFC 4035, March 2005.

[7] Wes Hardaker, "Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs)," RFC 4509, May 2006.

[8] Olaf Kolkman and Miek Gieben, "DNSSEC Operational Practices, Version 2," RFC 6781, December 2012.

[9] Samuel Weiler and David Blacka, "Clarifications and Implementation Notes for DNS Security (DNSSEC)," RFC 6840, February 2013.

[10] OpenSSL: `https://www.openssl.org/`

[11] BIND: `https://www.isc.org/downloads/bind/`

[12] PowerDNS: `https://www.powerdns.com/`

[13] LDNS: `https://www.nlnetlabs.nl/projects/ldns/about/`

[14] Paul Hoffman, Andrew Sullivan, and Kazunori Fujiwara, "DNS Terminology," RFC 8499, January 2019.

[15] *L'Agence nationale de la sécurité des systèmes d'information* (ANSSI): `https://www.ssi.gouv.fr/`

[16] *DNS Operations, Analysis, and Research Center* (DNS-OARC): `https://www.dns-oarc.net/`

[17] Michael StJohns, "Automated Updates of DNS Security (DNSSEC) Trust Anchors," RFC 5011, September 2007.

[18] Duane Wessels, Paul Hoffman, and Warren Kumari, "Signaling Trust Anchor Knowledge in DNS Security Extensions (DNSSEC)," RFC 8145, April 2017.

ROY ARENDS serves as a Principal Research Scientist at *The Internet Corporation for Assigned Names and Numbers* (ICANN). Roy is responsible for successfully delivering research projects; undertaking research design, data collection, and analysis; and producing insightful, stimulating reports that expand knowledge related to the system of unique identifiers on the Internet. E-mail: `roy.arends@icann.org`

Fragments

New DNS Terminology RFC

A *Request For Comments* (RFC) updating *Domain Name System* (DNS) terminology was recently published[0], continuing a decades-long IETF practice of publishing documents to help introduce interested readers to protocol topics by going through the most important terms.

The list of topics with terminology documents includes general terminology[1], *Network Address Translators* (NATs)[2], *Diffserv*[3], Internet connectivity[4], internationalization[5], and *Internet of Things* (IoT) networks[6]. Although these documents are not meant to be step-by-step introductions to the topics, they help someone who already has some understanding go deeper into the topic, and often help clarify terms that are often misused in common writing.

There are many dozens of RFCs defining the DNS, so the terminology is often hard to find. Some common terms such as "host name" are not defined in any RFCs; some are defined only by example; worse, some are defined differently in different RFCs. RFC 8499, "DNS Terminology," was published as an update to an earlier work to address these issues.

This document is the result of long discussions in the *Domain Name System Operations* (DNSOPS) Working Group[7], where dozens of DNS operators, software developers, and other experts brought up terms to be covered and argued over the current meaning of terms that are more than 30 years old. A common glossary is necessary to operate the DNS, and to continue to develop the DNS, so that people know what each other mean. The Working Group also hoped that the document would be useful to people who used the DNS tangentially, such as developers of other protocols and non-technical people who interact with the DNS in their work.

RFC 8499 is an update to the first DNS terminology document, RFC 7719[8]. While the first document was being written, the Working Group agreed that some definitions (such as for "domain name") needed more work, and it was so difficult to get consensus on other terms that they were left out. The new document is much more complete, and contains some common terms not covered in the earlier document, such as "recursive query," "lame delegation," and "split DNS."

Another significant addition to the document is the first definition of a standards-track document of "the global DNS" and "private DNS." Many people think they know what "the DNS" is but may not have a specific definition for it; these new terms helps get everyone using the same definitions. Overall, nearly 40 terms that are not defined in other RFCs are defined in this document.

Of course, the DNS will continue to evolve, and new terminology may appear. RFC 8499 is stable, but it might be revised a few years down the road to add these new terms.

[0] Paul Hoffman, Andrew Sullivan, and Kazunori Fujiwara, "DNS Terminology," RFC 8499, January 2019.

[1] Gary Scott Malkin, "Internet Users' Glossary," RFC 1983, August 1996.

[2] Matt Holdrege and Pyda Srisuresh, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663, August 1999.

[3] Dan Grossman, "New Terminology and Clarifications for Diffserv," RFC 3260, April 2002.

[4] John C Klensin, "Terminology for Describing Internet Connectivity," RFC 4084, May 2005.

[5] Paul Hoffman and John C Klensin, "Terminology Used in Internationalization in the IETF," RFC 6365, September 2011.

[6] Carsten Bormann, Ari Keranen, and Mehmet Ersue, "Terminology for Constrained-Node Networks," RFC 7228, May 2014.

[7] DNSOPS Working Group:
`https://datatracker.ietf.org/wg/dnsop/charter/`

[8] Kazunori Fujiwara, Paul Hoffman, and Andrew Sullivan, "DNS Terminology," RFC 7719, December 2015.

(Source: `https://www.ietf.org/blog/`)

DNS-OARC

The *DNS Operations, Analysis, and Research Center* (DNS-OARC) brings together key operators, implementers, and researchers on a trusted platform so they can coordinate responses to attacks and other concerns, share information and learn together. DNS-OARC has five key functions:

Information Sharing: DNS-OARC provides a trusted, shared platform to allow the DNS operations community to share information and data. Stringent confidentiality requirements and secure communications mean that proprietary information can be shared on a bilateral basis.

Operational Characterization: As Internet traffic levels continue to grow, the demand on root and other key name servers will outgrow the current infrastructure: this year's DDoS attack traffic levels will become next year's steady state load. DNS-OARC measures the performance and load of key name servers and publish statistics on both traffic load and traffic type (including error types).

Workshops: DNS-OARC organizes semi-annual workshops where members and the public are invited to give presentations on timely topics relevant to DNS both operations and research.

Analysis: Leading researchers and developers provide long-term analysis of DNS performance and post-mortems of attacks so that institutional learning occurs. A well-provisioned system allows members to upload traces and logs, and to perform their own analysis.

Tools and Services: As vulnerabilities and DNS problems come to light, DNS-OARC develops publicly available tools and services to assist with highlighting, diagnosing, and remedying such problems.

DNS-OARC participants fall into one or more of the following categories:

- Operators of root, TLD, or large commercial name servers who consume DNS technology and produce DNS services.
- Implementers who produce DNS technology including software, appliances, and network elements such as load balancing hardware
- Researchers whose work has a strong DNS emphasis and who need access to trace and log data about the global DNS under both "normal" and "abnormal" conditions.
- Security Providers whose companies offer products and services that utilize DNS information to improve the security of their customers.

For more information, visit: **https://www.dns-oarc.net/**

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
Scott Aitken
Jacobus Akkerhuis
Antonio Cuñat Alario
Matteo D'Ambrosio
Jens Andersson
Danish Ansari
Tim Armstrong
Richard Artes
David Atkins
Jaime Badua
Hidde Beumer
Pier Paolo Biagi
John Bigrow
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
Caner Budakoglu
Stefan Buckmann
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
Steve Corbató
Brian Courtney
Dave Crocker
Kevin Croes
John Curran
André Danthine
Morgan Davis
Jeff Day
Freek Dijkstra
Geert Van Dijk
David Dillow
Richard Dodsworth
Ernesto Doelling
Eugene Doroniuk
Karlheinz Dölger
Joshua Dreier
Lutz Drink
Andrew Dul
Holger Durer
Mark Eanes
Peter Robert Egli
George Ehlers
Peter Eisses
Torbjörn Eklöv
ERNW GmbH
ESdatCo
Steve Esquivel
Jay Etchings
Mikhail Evstiounin
Paul Ferguson
Kent Fichtner
The Flirble Organisation
Gary Ford
Jean-Pierre Forcioli
Christopher Forsyth
Andrew Fox
Craig Fox
Fausto Franceschini
Tomislav Futivic
Edward Gallagher
Andrew Gallo
Chris Gamboni
Xosé Bravo Garcia
Kevin Gee
John Gilbert
Serge Van Ginderachter
Greg Goddard
Octavio Alfageme Gorostiaga
Barry Greene
Martijn Groenleer
Geert Jan de Groot
Christopher Guemez
Gulf Coast Shots
Sheryll de Guzman
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
Alain Van Hoof
Edward Hotard
Bill Huber
Hagen Hultzsch
Kevin Iddles
Mika Ilvesmaki
Karsten Iwen
David Jaffe
Ashford Jaggernauth
Jozef Janitor
John Jarvis
Dennis Jennings
Edward Jennings
Aart Jochem
Richard Johnson
Jim Johnston
Jonatan Jonasson
Daniel Jones
Gary Jones
Jerry Jones
Amar Joshi
Merike Kaeo
Andrew Kaiser
Christos Karayiannis
David Kekar
Jithin Kesavan
Jubal Kessler
Shan Ali Khan
Nabeel Khatri
Anthony Klopp
Henry Kluge
Michael Kluk
Andrew Koch
Ia Kochiashvili
Carsten Koempe
Alexader Kogan
Antonin Kral
Mathias Körber
John Kristoff
Terje Krogdahl
Bobby Krupczak
Murray Kucherawy
Warren Kumari
Darrell Lack
Yan Landriault
Markus Langenmair
Fred Langham
Andrew Lamb
Richard Lamb
Tracy LaQuey Parker

Simon Leinen
Robert Lewis
Martin Lillepuu
Sergio Loreti
Guillermo a Loyola
Hannes Lubich
Dan Lynch
Miroslav Madić
Alexis Madriz
Carl Malamud
Michael Malik
Yogesh Mangar
Bill Manning
Harold March
Vincent Marchand
David Martin
Jim Martin
Timothy Martin
Gabriel Marroquin
Carles Mateu
Juan Jose Marin Martinez
Ioan Maxim
David Mazel
Miles McCredie
Brian McCullough
Joe McEachern
Jay McMaster
Mark Mc Nicholas
Carsten Melberg
Kevin Menezes
Bart Jan Menkveld
William Mills
David Millsom
Desiree Miloshevic
Joost van der Minnen
Thomas Mino
Wijnand Modderman
Mohammad Moghaddas
Charles Monson
Andrea Montefusco
Fernando Montenegro
Joel Moore
Maurizio Moroni
Brian Mort
Soenke Mumm
Tariq Mustafa
Stuart Nadin
Mazdak Rajabi Nasab
Krishna Natarajan
Darryl Newman
Paul Nikolich
Travis Northrup
Marijana Novakovic
David Oates
Ovidiu Obersterescu
Tim O'Brien
Mike O'Connor
Mike O'Dell
Jim Oplotnik
Carlos Astor Araujo Palmeira
Alexis Panagopoulos
Gaurav Panwar
Manuel Uruena Pascual
Ricardo Patara
Dipesh Patel
Alex Parkinson
Craig Partridge
Dan Paynter
Leif Eric Pedersen
Juan Pena
Chris Perkins
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
Paul Rathbone
Bill Reid
Rodrigo Ribeiro
Glenn Ricart
Justin Richards
Mark Risinger
Ron Rockrohr
Carlos Rodrigues
Lex Van Roon
William Ross
Boudhayan Roychowdhury
Carlos Rubio
Timo Ruiter
RustedMusic
Babak Saberi
George Sadowsky
Scott Sandefur
Sachin Sapkal
Arturas Satkovskis
PS Saunders
John Sayer
Phil Scarr
Elizabeth Scheid
Jeroen Van Ingen Schenau
Carsten Scherb
Ernest Schirmer
Dan Schrenk
Richard Schultz
Roger Schwartz
SeenThere
Scott Seifel
Yury Shefer
Yaron Sheffer
Doron Shikmoni
Tj Shumway
Jeffrey Sicuranza
Thorsten Sideboard
Andrew Simmons
Pradeep Singh
Henry Sinnreich
Geoff Sisson
Helge Skrivervik
Darren Sleeth
Bob Smith
Courtney Smith
Mark Smith
Job Snijders
Ronald Solano
Asit Som
Ignacio Soto Campos
Peter Spekreijse
Thayumanavan Sridhar
Paul Stancik
Ralf Stempfer
Matthew Stenberg
Adrian Stevens
Clinton Stevens
John Streck
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
Buddy Venne
Alejandro Vennera
Luca Ventura
Tom Vest
Dario Vitali
Laurence Walker
Randy Watts
Andrew Webster
Tim Weil
Jd Wegner
Rick Wesson
Peter Whimp
Jurrien Wijlhuizen
Pindar Wong
Romeo Zwart
Bernd Zeimetz
廖 明沂.

Follow us on Twitter and Facebook `@protocoljournal` `https://www.facebook.com/newipj`

Letters to the Editor

Ole,

Geoff Huston's most recent article on the last 10 years of the Internet is absolutely brilliant (IPJ Volume 21, No. 2, August 2018). As one of the early implementers of our dear Internet, I am of course amazed at its evolution these past decades, and Geoff has more than "kept up"! His ability to summarize quickly and accurately is without peer. Thank you all.

—Dan Lynch
dan@lynch.com

Geoff,

Thank you very much for your article "Another 10 Years" in *The Internet Protocol Journal*. I enjoyed your perspective and your writing style very much. You have a great skill at explaining a great amount of information.

I subscribed to the early *ConneXions—The Interoperability Report* and later IPJ. I've been glad to see your articles over the many years.

Sincerely,

—Richard Berke
Richard_Berke@troweprice.com

The author responds:

I really appreciate your kind words, and I am glad you liked the article.

—Geoff Huston
gih@apnic.net

Letters may be edited for clarity. We'd love to hear from you. Send us your feedback via e-mail to **ipj@protocoljournal.org**

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

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**

Supporters and Sponsors

Supporters

Diamond Sponsors

Ruby Sponsors

Your logo here!

Sapphire Sponsors

Your logo here!

Emerald Sponsors

Corporate Subscriptions

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

The Internet Protocol Journal
NMS
535 Brennan Street
San Jose, CA 95131

ADDRESS 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