The Internet Protocol Journal

April 2018 Volume 21, 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

We live in a complex and increasingly interconnected world. With this complexity comes a desire by network engineers to design systems that can cope with increasing demands while still offering predictable performance, manageability, and maintainability. In our first article, Russ White discusses ways to analyze network designs from a complexity point of view.

The *Internet Protocol* (IP) was designed to operate over a variety of underlying network technologies, such as Ethernet, X.25, FDDI, Frame Relay, WiFi, and even mobile telephone networks. Applications that use IP must deal with the fact that datagrams may be split into *fragments* as they travel across the network with subsequent *reassembly* at the receiving end. Previous articles in this journal have discussed fragmentation, largely in the context of IPv4. This time Geoff Huston describes fragmentation in IPv6 and the particular challenges that arise with this protocol in conjunction with applications such as the *Domain Name System* (DNS).

We usually provide a section of announcements entitled "Fragments," but this time it has been replaced by a selection of Letters to the Editor—all in response to articles in our November 2017 issue. We are very happy to receive feedback on any aspect of this journal, and we would also point you to our website, which contains additional articles and material as well as all of our back issues in PDF format.

As mentioned in previous issues, 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**

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

Considerations in Network Complexity

by Russ White

Computer networks are complex—and getting more complex by the day. At one time, knowing the *Internet Protocol* (IP) was enough; today there are underlays, overlays, virtualized services, service chains, and a host of other technologies engineers need to plan around and for. With complexity on the rise, maybe it's time to ask some fundamental questions, such as—what does complexity mean? Can complexity be solved? How can engineers manage complexity?

Why So Complex?

While the most obvious place to begin might be with a definition of complexity, it's actually more useful to consider why complexity is required in a more general sense. To put it more succinctly, is it possible to "solve" complexity? Why not just design networks and protocols that are simpler? Why does every attempt to make anything simpler in the networking world end up apparently making things more complex in the long run? For instance, tunneling on top of (or through) IP reduces the complexity of the control plane and makes the network simpler overall. Why is it, then, that tunneled overlays end up containing so much complexity?

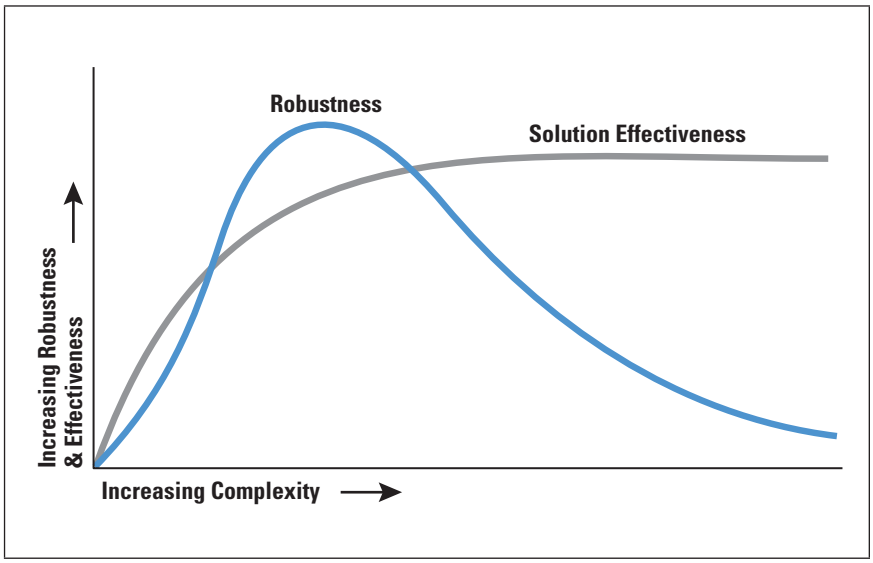
This question has two answers: The first is that human nature being what it is, engineers will always invent 10 different ways to solve the same problem. This reality is especially true in the virtual world, where new solutions are (relatively) easy to deploy, it's (relatively) easy to find a problem with the last set of proposed solutions, and it's (relatively) easy to move some bits around to create a new solution that is "better than the old one." The virtual space, in other words, is partially so messy because it's so easy to build something new there.

- Abstract the complexity away, to build a black box around each part of the system, so each piece and the interactions among these pieces are more immediately understandable.
- Toss the complexity over the cubicle wall—to move the problem out of the networking realm into the realm of applications, or coding, or a protocol. As RFC 1925[1] says, "It is easier to move a problem around (for example, by moving the problem to a different part of the overall network architecture) than it is to solve it."
- Add another layer on top, to treat all the complexity as a black box by putting another protocol or tunnel on top of what's already there. Returning to RFC 1925, "It is always possible to add another level of indirection."
- Become overwhelmed with the complexity, label what exists as "legacy," and chase some new shiny thing that will solve all the problems in what is perceived as a much less complex way.

- Ignore the problem and hope it will go away. Argue for an exception "just this once," to meet a particular business goal, or fix some problem, within a very tight schedule, with the promise that the complexity issue will be dealt with "later," is a good example.

The second answer, however, lies in a more fundamental problem: complexity is necessary to deal with the uncertainty involved in problems that are difficult to solve (Figure 1).

Figure 1: Complexity, Effectiveness, and Robustness



Adding complexity, then, allows a network to handle future requirements and unexpected events more easily, as well as providing more services over a smaller set of base functions. If this condition is the case, why not simply build a single protocol running on a single network that can handle all the requirements potentially thrown at it, and can handle any sequence of events you can imagine? A single network running a single protocol would certainly reduce the number of moving parts network engineers need to deal with, making all our lives simpler, right?

Maybe not. At some point, any complex system becomes brittle—*robust yet fragile* is one phrase you can use to describe this condition. A system is robust yet fragile when it is able to react resiliently to an expected set of circumstances, but an unexpected set of circumstances will cause it to fail. As an example from the real world—knife blades are required to have a somewhat unique combination of characteristics. They must be hard enough to hold an edge and cut, and yet flexible enough to bend slightly in use, returning to their original shape without any evidence of damage, and they must not shatter when dropped. It has taken years of research and experience to find the right metal to make a knife blade, and there are still long and deeply technical discussions about which material is right for specific properties, under what conditions, etc.

Complexity is necessary, then: it cannot be "solved."

Defining Complexity

Given complexity is necessary, engineers are going to need to learn to manage it in some way, by finding or building a model or framework. The best place to begin to build such a model is with the most fundamental question: what does complexity mean in terms of networks? Can you put a network on a scale and have the needle point to "complex?" Is there a mathematical model into which you can plug the configurations and topology of a set of network devices that will, in turn, produce a "complexity index?" How do the concepts of scale, resilience, brittleness, and elegance relate to complexity? The best place to begin in building a model is with an example.

Control-Plane State versus Stretch

What is network *stretch*? In the simplest terms possible, it is the difference between the shortest path in a network and the path traffic between two points actually takes. Figure 2 illustrates this concept.

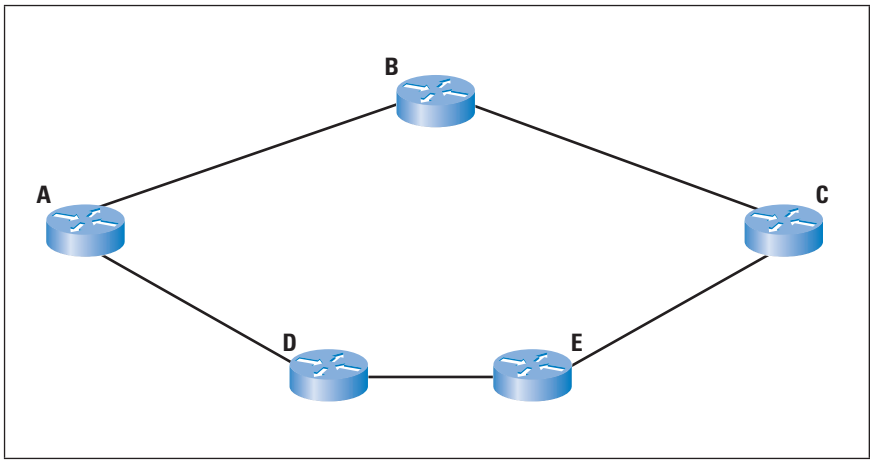


Figure 2: A Small Network to Illustrate State and Stretch

Assuming the cost of each link in this network is the same, the shortest physical path between Routers A and C will also be the shortest logical path: [A,B,C]. What happens, however, if we change the metric on the [A,B] link to 3? The shortest physical path is still [A,B,C], but the shortest logical path is now [A,D,E,C]. The differential between the shortest physical path and the shortest logical path is the distance a packet being forwarded between Routers A and C must travel—in this case, the stretch can be calculated as (4 [A,D,E,C])–(3 [A,B,C]), for a stretch of 1.

How Is Stretch Measured?

In terms of hop count, is stretch measured by the summary of the metrics, the delay through the network, or some other way? It depends on what is most important in any given situation, but the most common way is by comparing hop counts through the network, and this method is used in the examples here for simplicity. In some cases, it might be more important to consider the metric along two paths, the delay along two paths, or some other metric, but the important point is to measure it consistently across every possible path to allow for accurate comparison between paths.

It's sometimes difficult to differentiate between the physical topology and the logical topology. In this case, was the [A,B] link metric increased because the link is actually a slower link? If so, whether this is an example of stretch or an example of simply bringing the logical topology in line with the physical topology is debatable.

In line with this observation, it's much easier to define policy in terms of stretch than almost any other way. Policy is any configuration that increases the stretch of a network. Using *Policy-Based Routing* or *Traffic Engineering* to push traffic off the shortest physical path and onto a longer logical path to reduce congestion on specific links, for instance, is a policy—it increases stretch.

Increasing stretch is not always a bad thing. Understanding the concept of stretch simply helps us understand various other concepts, and put a framework around complexity tradeoffs. The shortest path, physically speaking, isn't always the best path.

Stretch, in this illustration, is very simple—it affects every destination, and every packet flowing through the network. In the real world, things aren't so simple. Stretch is actually per source/destination pair, making it very difficult to measure on a networkwide basis.

With all of this information in mind, let's look at two specific examples of the tradeoff between stretch and optimization.

Aggregation versus Stretch

Aggregation is a technique used to reduce not only the amount of information carried in the control plane, but also the rate of state change in the control plane. Aggregation is built into IP (both IPv4 and IPv6)—even a single subnet contains multiple host addresses. By connecting a single broadcast segment to a set of hosts, the IP routing protocol doesn't need to manage Layer 2 reachability, nor individual host addresses. Aggregation within the control plane can also cause stretch, as Figure 3 shows.

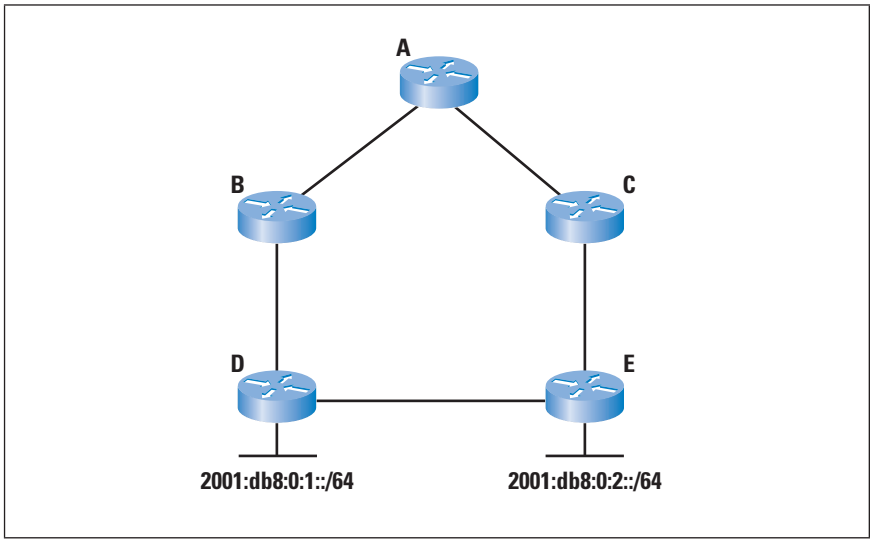


Figure 3: Aggregation and Stretch

Two different situations illustrate increasing stretch through route aggregation:

1. Assume the [A,B] link has a cost of 2, and all the other links in this network have a cost of 1. If Routers B and C both aggregate to `2001:db8::/61`, then the path through [A,C] would be preferred for everything within the aggregate. Traffic destined to `2001:db8:0:1::/64` will pass along the path [A,C,E,D] to reach its destination, even though the shortest (physical) path is [A,B,D]. The stretch for `2001:db8:0:2::/64` isn't changed, but the stretch for `2001:db8:0:1::/64` is increased by 1.
2. Assume all the links in the network have a cost of 1. If Routers B and C both aggregate to `2001:db8::/61`, then Router A will somehow load share traffic toward the two subnets behind Routers D and E across the two equal-cost paths it has available. Given perfect load sharing, 50% of the traffic destined to `2001:db8:0:1::/64` will flow along [A,C,E,D], with a stretch of 1, and 50% of the traffic destined to `2001:db8:0:2::/64` will flow along [A,B,D,E], with a stretch of 1.

Implementing aggregation removes specific reachability information about the two /64 prefixes behind Routers D and E from the state of Router A. Implementation aggregation also disconnects the state of the individual /64's behind Routers D and E from the state at Router A. Aggregation, then, decreases complexity from the perspective of Router A by reducing the *amount* and *speed* of state in the routing table of Router A.

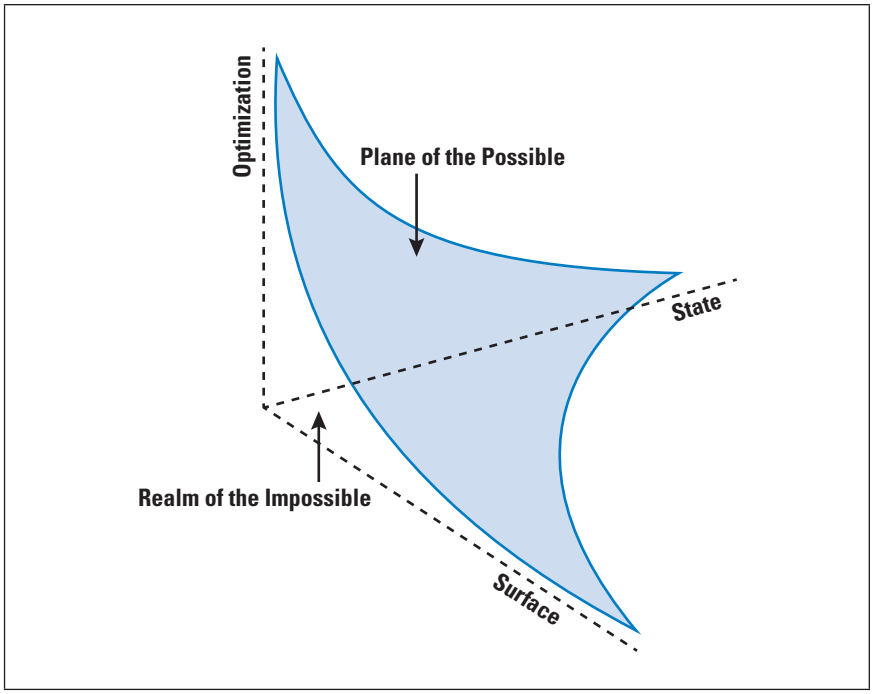
Increasing stretch increases the overall use of the network without any actual increase in the amount of traffic being carried through the network. In the example given in Figure 2, traffic that would normally take a two-hop path is directed along a three-hop path, meaning one more link and router are involved in forwarding and switching the packets in the flow(s) across the network. In purely mathematical terms, increasing stretch decreases the overall efficiency of the network by increasing the number of devices and links used to forward any particular flow.

Finally, to implement aggregation Routers B and C must be configured to summarize the two longer prefixes into a single shorter one. This additional configuration introduces an additional bit of interaction between the human operator (or at least the configuration system) and the control plane. This situation can be described as an increase in *surface* in the network.

Defining Complexity: A Model

These three components—*state*, *optimization*, and *surface*—are common in virtually every network or protocol design decision. They can be seen as a set of tradeoffs, as illustrated in Figure 4.

Figure 4: The Plane of the Possible



Increasing optimization always moves towards more state or more interaction surfaces. Decreasing state always moves towards less optimization or more interaction surfaces. Decreasing interaction surfaces always moves towards less optimization or more state. These rules aren't ironclad, of course; they are contingent on the specific network, protocols, and requirements, but they are generally true often enough to make this model useful for understanding tradeoffs in complexity.

Interaction Surfaces

While state and optimization are fairly intuitive, it's worthwhile to spend just a moment more on interaction surfaces. The concept of interaction surfaces is difficult to grasp primarily because it covers such a wide array of ideas. Perhaps an example would be helpful; assume a function that:

- Accepts two numbers as input
- Adds them
- Multiplies the resulting sum by 100
- Returns the result

This single function can be considered a subsystem in some larger system. Now assume you break this single function into two functions, one of which does the addition, and the other of which does the multiplication. You've created two simpler functions (each one does only one thing), but you've created an interaction surface between the two functions—you've created two interacting subsystems within the system where there used to be only one. This example is really simple, I know, but consider a few more that might help.

The routing information carried in *Open Shortest Path First* (OSPF) is split into *external* routes being carried in *Border Gateway Protocol* (BGP) and *internal* routes being carried in OSPF. You've gone from one system with more state to two systems with less state, but you've created an interaction surface between the two protocols—they must now work together to build a complete forwarding table.

A single set of hosts with different access policies is split onto multiple virtual topologies on the same physical network. You've simplified the amount of state in filtering, but you've created an interaction surface between the two virtual topologies and between the two topologies and the control plane. In addition, you've exposed new shared risk groups where a single physical failure can cause multiple logical ones. Hence you've traded state in one control plane for interaction surfaces between multiple control planes.

Even two routers communicating within a single control plane can be considered an interaction surface. This breadth of definition is what makes it so very difficult to define what an interaction surface is.

Interaction surfaces aren't a bad thing; they help engineers and designers divide and conquer in any given problem space, from modeling to implementation. At the same time, interaction surfaces are all too easy to introduce without thought.

Managing Complexity through the Wasp Waist

There is a simple model that is ubiquitous throughout the natural world, and is widely mimicked in the engineering world. While engineers don't often consciously apply this model, it's actually used all the time. What is this model?

Figure 5 illustrates the hourglass model in the context of the four-layer *Department of Defense* (DoD) model that gave rise to the *Internet Protocol Suite*.

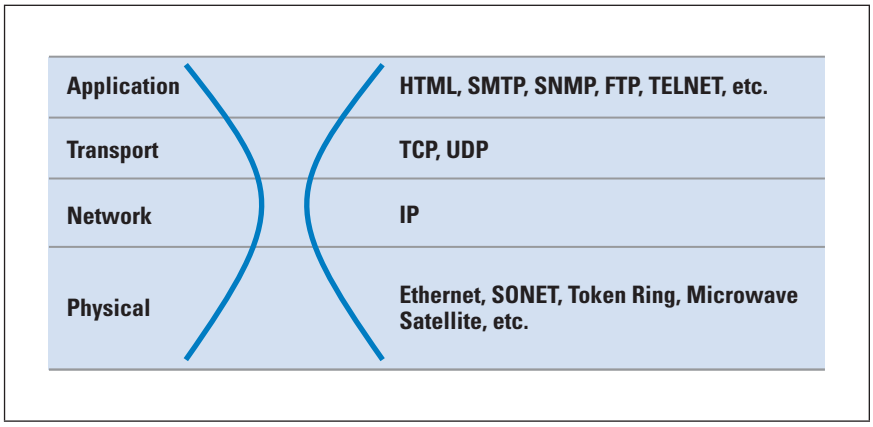


Figure 5: The DoD Model and the "Wasp Waist"

At the bottom layer, the physical transport system, there is a wide array of protocols, from Ethernet to Satellite. At the top layer, where information is marshaled and presented to applications, there is a wide array of protocols, from *Hypertext Transfer Protocol* (HTTP) to TELNET (and thousands of others besides). A funny thing happens when you move towards the middle of the stack, however: the number of protocols decreases, creating an hourglass. Why does this work to control complexity?

Going back through the three components of complexity—*state, optimization,* and *surface*—exposes the relationship between the hourglass and complexity.

- *State* is divided by the hourglass into two distinct types of state: information about the network, and information about the data being transported across the network. While the upper layers are concerned with marshaling and presenting information in a usable way, the lower layers are concerned with discovering what connectivity exists and what the properties of that connectivity actually are. The lower layers don't need to know how to format a *File Transfer Protocol* (FTP) frame, and the upper layers don't need to know how to carry a packet over Ethernet—state is reduced at both ends of the model.
- *Optimization* is traded off by allowing one layer to reach into another layer, and by hiding the state of the network from the applications. For instance, the *Transmission Control Protocol,* (TCP) doesn't really know the state of the network other than what it can gather from local information. TCP could potentially be much more efficient in its use of network resources, but only at the cost of a layer violation, which opens up interaction surfaces that are difficult to control.
- *Surfaces* are controlled by reducing the number of interaction points between the various components to precisely one—IP. This single interaction point can be well defined through a standards process, with changes in the one interaction point closely regulated to prevent massive rapid changes that will reflect up and down the protocol stack.

The layering of a stacked network model is, then, a direct attempt to control the complexity of the various interacting components of a network.

Managing Complexity as an Engineer

Managing complexity in design in a general sense is just one application of the state/optimization/surface model. Another use is in learning how to understand complex systems quickly—a skill every engineer could use in everyday life. Here this three-sided model is used as part of a process, or a way of thinking about systems.

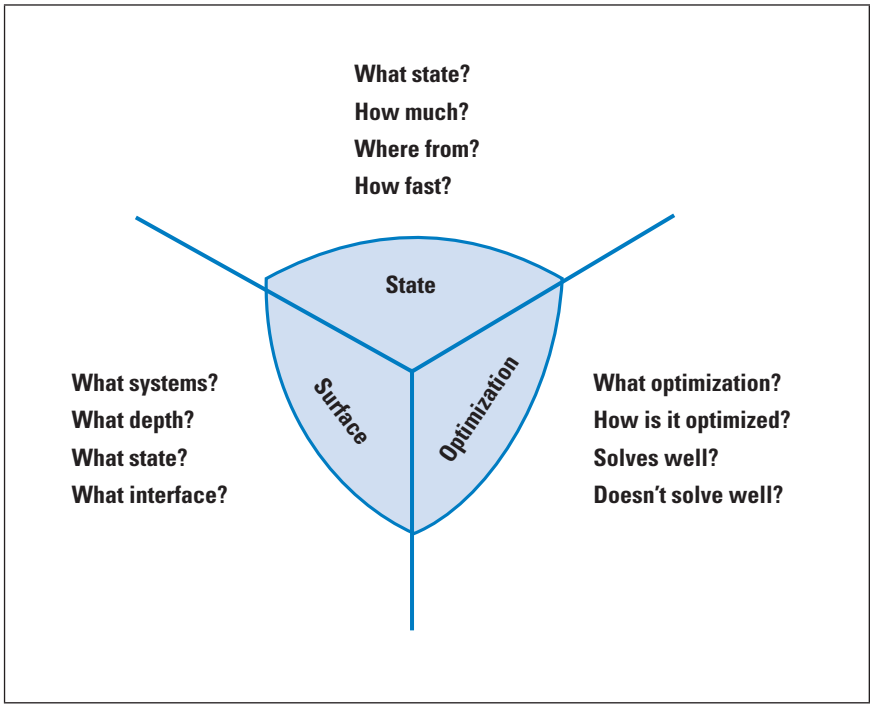
There are three steps in this model, or rather three questions:

- Why is this being done this way?
- What is being accomplished?
- What is this like?

Why acts as an abstractor, focusing on the problem at hand by excluding others. This question uncovers the purpose, or the goal, of the system. Asking why also connects the system to the business driver. As an example: "Why does OSPF elect a designated router?" might be a perfectly valid question in some settings, but not necessarily while you're troubleshooting slow network convergence. Asking why can focus energy and uncover motives that drive configuration, policy, and other choices.

What is the question engineers normally engage with first, but they often do so with little structure. The three-pronged complexity model provides a solid model with which to find the right questions to ask. Figure 6 illustrates this method.

Figure 6: A Structure for Asking What



By focusing on questions about each of the three prongs of the complexity model, you can quickly uncover the tradeoffs made in system design—even if those tradeoffs weren't made intentionally.

Finally, *what is this like* helps relate the problem space, the potential solutions, and even potential problems with potential solutions to the system being considered. Assume a new working group is formed to solve a particular problem in the routing space. The working group quickly decides on using Dijkstra's *Shortest Path First* (SPF) algorithm to find the solution to one particular problem the protocol is set out to solve.

Asking *what is this like* should immediately uncover the relationship of new solution to existing link-state protocols, such as OSPF. From there, given experience with OSPF, the engineer knows what sort of convergence characteristics the newly proposed solution is likely to have, and where to look for potential problems. As link-state protocols are subject to microloops in some situations, so the newly proposed solution is likely to be subject to microloops. As link-state protocols can suffer from overwhelming amounts of state during large-scale convergence events, so the new solution might suffer from the same sort of problem.

Concluding Thoughts

You can't run, and you can't hide from complexity; there's no point in even trying. You're going to encounter it; ignoring it doesn't make the problem go away, it just allows the problem to fester under some "rug" in some corner of your network. The complexity problems you create today will return as bigger, more complex problems in just a few years. To quote someone who's spent years looking at complexity:

> *"Trying to make a network proof against predictable problems tends to make it fragile in dealing with unpredictable problems (through an ossification effect as you mentioned). Giving the same network the strongest possible ability to defend itself against unpredictable problems, it necessarily follows, means that it MUST NOT be too terribly robust against predictable problems—not being too robust against predictable problems is necessary to avoid the ossification issue, but not necessarily sufficient to provide for a robust ability to handle unpredictable network problems."*
>
> *—Tony Przygienda*

That call at 2 a.m. might not be pleasant, but solving it the wrong way might cause a much worse call at 2 a.m. sometime later. Hardening the network against all failures eventually means to make it fail spectacularly when a failure that you didn't predict occurs—there's just no way around this reality.

When dealing with engineering problems, then, a little humility around what can, and cannot, be solved is in order. Don't ignore complexity, but don't think you can solve it, either. Instead, remember to treat every situation as a set of tradeoffs—and if you don't see the tradeoffs, you're not looking hard enough.

References

[1] Ross Callon, "The Twelve Networking Truths," RFC 1925, April 1996.

[2] Russ White and Shawn Zandi, "Cloudy-Eyed: Complexity and Reality with Software-Defined Networks," *The Internet Protocol Journal,* Volume 19, No. 3, September 2016.

[3] Russ White and Jeff Tantsura, *Navigating Network Complexity: Next-Generation Routing with SDN, Service Virtualization, and Service Chaining,* Addison-Wesley Professional, 2015, ISBN-13: 978-0133989359.

[4] Russ White and Ethan Banks, *Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks*, Addison-Wesley Professional, 2018, ISBN-13: 978-1587145049.

RUSS WHITE began working with computers in the mid-1980s, and computer networks in 1990. He has experience in designing, deploying, breaking, and troubleshooting large-scale networks, and is a strong communicator from the white board to the board room. Across that time, he has co-authored more than 40 software patents, participated in the development of several Internet standards, helped develop the *Cisco Certified Design Expert* (CCDE) and the *Cisco Certified Architect* (CCAr) programs, and worked in Internet governance with the Internet Society. Russ has a background covering a broad spectrum of topics, including radio frequency engineering and graphic design, and is an active student of philosophy and culture. Russ is a co-host at the *Network Collective,* serves on the Routing Area Directorate at the IETF, co-chairs the BABEL working group, serves on the *Technical Services Council* as a maintainer on the open source *FR Routing* project, and serves on the *Linux Foundation* (Networking) board. His most recent works are *Computer Networking Problems and Solutions, The Art of Network Architecture, Navigating Network Complexity,* and the Intermediate System-to-Intermediate System LiveLesson. He holds a *Master of Science in Information Technology* (MSIT) from Capella University, and a *Master of Arts in Christian Ministries* (MACM) from Shepherds Theological Seminary and is currently working on a PhD at Southeastern Baptist Theological Seminary. E-mail: **russ@riw.us**

IPv6 and Packet Fragmentation

by Geoff Huston, APNIC

Version 6 of the Internet Protocol (IPv6) introduced very few changes to its Version 4 predecessor (IPv4). The major change was of course the expansion of the size of the IP source and destination address fields in the IP packet header from 32 bits to 128 bits. Some other changes, however, apparently were intended to subtly alter IP behaviour. One of them was the change in treatment of IP *packet fragmentation*.

It appears that rather than effecting a slight improvement from IPv4, the manner of fragmentation handling in IPv6 appears to be significantly more difficult. In light of these problems, it is perhaps unsurprising that calls have been made from time to time to dispense completely with packet fragmentation in IPv6[1], as the current situation with IPv6 appears to be worse than both no fragmentation and the IPv4-style of fragmentation.

Packet Fragmentation

One of the more difficult design exercises in packet-switched network architectures is the design of packet fragmentation.

In time-switched networks, developed to support a common bearer model for telephony, each "unit" of information passed through the network occurred within a fixed timeframe (an analogue voice stream was digitized into 8,000 sample points per second, so the basic time unit for switching these digital samples was 1/8,000 of a second), which resulted in fixed-size packets, all clocked off a common time base.

Packet-switched data networks could dispense with a constant common time base, in turn allowing individual data packets to be sized according to the needs of the application as well as the needs and limitations of the network substrate.

For example, smaller packets have a higher packet header–to–packet payload ratio and are consequently less efficient in data carriage and impose a higher processing load as a function of effective data throughput. On the other hand, within a packet-switching system the smaller packet can be dispatched faster, reducing head-of-line blocking in the internal queues within a packet switch and potentially reducing network-imposed jitter as a result. This reduction can make it easier to use the network for real-time applications such as voice or video. Larger packets allow larger data payloads, in turn allowing greater carriage efficiency. Larger payloads per packet also allows a higher internal switch capacity when measured in terms of data throughput, in turn facilitating higher carriage capacity and higher-speed network systems.

Various network designs adopted various parameters for packet size. The original Ethernet specification, invented in the early 1970s, adopted a variable packet size, with supported packet sizes of between 64 and 1,500 octets. *Fiber Distributed Data Interface* (FDDI), a fibre ring local network, used a variable packet size of up to 4,478 octets. *Frame Relay* used a variable packet size of between 46 and 4,470 octets. The choice of variable-size packets allows applications to refine their behaviour. Jitter and delay-sensitive applications, such as digitised voice, may prefer to use a stream of smaller packets in an attempt to minimise jitter, while reliable bulk data transfer may choose a larger packet size to increase the carriage efficiency. The nature of the medium may also have a bearing on this choice. If there is a high *Bit Error Rate* (BER) probability, then reducing the packet size minimises the impact of sporadic errors within the data stream, possibly increasing throughput in such environments.

In designing a network protocol that is intended to operate over a wide variety of substrate networking media and support as wide a variety of applications as possible, the designers of IP could not rely on a single packet size for all transmissions. Instead, the designers of IPv4 provided a *packet length field* in the packet header. This field was a 16-bit octet count, allowing for an IP packet to be anywhere from the minimum size of 20 octets (corresponding to an IP header without any payload) to a maximum of 65,535 octets.

Obviously not all packets can fit into all underlying media. If the packet is too small for the minimum payload size, then it can be readily padded. But if it's too big for the maximum packet size of the media, then the problem is a little more challenging. IPv4 solved this problem using "forward fragmentation." The basic approach is that any IPv4 router that is unable to forward an IPv4 packet into the next hop because the packet is too large for the next-hop network may split the packet into a set of smaller "fragments," copying the original IP header fields into each of these fragments, and then forwarding each of these fragments instead. The fragments continue along the network path as autonomous IP packets, and the destination host is responsible for re-assembling them back into the original IP packet and pass the result, namely the packet as it was originally sent, back up to the local instance of the end-to-end transport protocol.

It is a clever approach, as it hides the entire network-level fragmentation issue from the upper-level protocols, including the *Transmission Control Protocol* (TCP) and *User Datagram Protocol* (UDP). The transport protocols and the upper-level application protocols can, in theory, treat the underlying network as capable of supporting any IP packet size, and the IP layer performs the necessary adaptation to allow the IP packet to traverse any media layer. However, even with this intended transparency of operation, this approach has managed to earn a very poor reputation.

Packet fragmentation was seen as being a source of performance inefficiency, a security vulnerability, and it even posed a cap on maximal delay-bandwidth product on data flows across networks. IP fragmentation was considered harmful[2,3].

The IPv6 designers removed the fragmentation controls from the common IPv4 packet header and placed them into an 8-octet IPv6 *Extension Header*. This additional packet header, placed between the IPv6 packet header and the end-to-end transport packet header, was present *only* in fragmented packets (whereas the IPv4 fragmentation control fields are present in *all* IPv4 packet headers). Secondly, IPv6 did not permit fragmentation to be performed when the packet was in transit within the network: all fragmentation was to be performed by the packet source prior to transmission. This stipulation, too, has resulted in an uncomfortable compromise, where an unforeseen need for fragmentation relies on *Internet Control Message Protocol for IPv6* (ICMPv6) signalling from the interior of the network back to the original packet source and retransmission.

In the case of TCP, a small amount of layer violation goes a long way. If the sending host is permitted to pass an IPv6 *Packet Too Big* (PTB) ICMPv6 diagnostic message up to the TCP session that generated the original packet, then it's possible for the TCP driver to adjust its sending *Maximum Segment Size* (MSS) to the new, smaller value and carry on. In this case, no fragmentation is required.

UDP is different. In UDP a functional response to path message size issues inevitably relies on interaction with the upper-level application protocol.

It appears that when we consider fragmentation in IPv6 we have to consider the treatment of IPv6 Extension Headers and UDP. And that story is not a robust one[4].

The DNS and IPv6 Packet Fragmentation

The *Domain Name System* (DNS) is the major user of UDP. As a consequence of the increasing use of *Domain Name System Security Extensions* (DNSSEC) as a security mechanism, coupled with the increasing use of IPv6 as the IP protocol transition gathers momentum, it is time to look once more at the interaction of larger DNS payloads over IPv6.

To illustrate this situation, here are two DNS queries, both made by a recursive resolver to an authoritative name server, both using UDP over IPv6.

Query 1:

```
$ dig +bufsize=4096 +dnssec 000-4a4-000a-000a-
0000-b9ec853b-241-1498607999-2a72134a.ap2.
dotnxdomain.net. @8.8.8.8
```
139.162.21.135

```
(MSG SIZE rcvd: 1190)
```

Query 2:

```
$ dig +bufsize=4096 +dnssec 000-510-000a-000a-0000-
b9ec853b-241-1498607999-2a72134a.ap2.
dotnxdomain.net. @8.8.8.8
status: SERVFAIL

(MSG SIZE rcvd: 104)
```

What we see here are two almost identical DNS queries that have been passed to Google's Public DNS service to resolve.

In the first case, the DNS response is 1,190 octets long, and in the second case the response is 1,346 octets long. The DNS server is an IPv6-only server, and the underlying host of this name server is configured with a local maximum packet size of 1,280 octets. Therefore, in the first case the response being sent to the Google resolver is a single, unfragmented IPv6 UDP packet, and in the second case the response is broken into two fragmented IPv6 UDP packets. And it is this single change that triggers the Google Public DNS Server to provide the intended answer in the first case, but to return a SERVFAIL failure notice in response to the fragmented IPv6 response. When the local *Maximum Transmission Unit* (MTU) on the server is lifted from 1,280 octets to 1,500 octets, the Google resolver returns the server DNS response in both cases.

The only difference in these two responses is IPv6 fragmentation, but there is perhaps more to it than that.

IP fragmentation in both IPv4 and IPv6 "raises the eyebrows" of firewalls. Firewalls typically use the information provided in the transport protocol header of the IP packet to decide whether to admit or deny the packet. For example, you may see firewall rules admitting packets using TCP ports 80 and 443 as a way of allowing web traffic through the firewall filter. For this process to work, the inspected packet needs to contain a TCP header and use the fields in the header to match against the filter set. Fragmentation in IP duplicates the IP portion of the packet header, but the inner IP payload, including the transport protocol header, is not duplicated in every ensuing packet fragment. Thus trailing fragments pose a conundrum to the firewall. Either all trailing fragments are admitted, a situation that has its own set of consequent risks, or all trailing fragments are discarded, a situation that also poses connection issues[5].

IPv6 adds a further factor to the picture. In IPv4 every IP packet, fragmented or not, contains IP fragmentation control fields. In IPv6 these same fragmentation *control fields* are included in an IPv6 *Extension Header* that is attached only to packets that are fragmented.

This 8-octet Extension Header is placed immediately after the IPv6 packet header in all fragmented packets, meaning that a fragmented IPv6 packet does not contain the *Upper Level Protocol Header* starting at octet offset 40 from the start of the IP packet header. But in the first packet of this set of fragmented packets, the Upper Level Protocol Header is chained off the fragmentation header, at byte offset 48, assuming that there is only a *Fragmentation Extension Header* in the packet. The implications of this fact are quite significant. Instead of always looking at a fixed point in a packet to determine its upper-level protocol, the packet-handling device needs to unravel the Extension Header chain, raising two rather tough questions. First, how long is the device prepared to spend unravelling this chain? And second, would the device be prepared to pass on a packet with an Extension Header that it did not recognise?

In some cases, implementers of IPv6 equipment have found it simpler to just drop all IPv6 packets that contain Extension Headers. Some measurements of this behaviour are reported in RFC 7872[6]. This document reports a 38% packet-drop rate when sending fragmented IPv6 query packets to DNS Name servers. But the example provided previously is in fact the opposite case to that reported in RFC 7872, and the example illustrates a more conventional case. It's not the queries in the DNS that can readily grow to sizes that require packet fragmentation, but the responses. The relevant question concerns the anticipated probability of packet drop when sending fragmented UDP IPv6 packets as responses to DNS queries. To rephrase the question slightly, how do DNS recursive resolvers fare when the IPv6 response from the server is fragmented?

For a start, it appears from the example cited here that Google's Public DNS resolvers experienced some packet-drop problem when they passed a fragmented IPv6 response (this problem was noted in mid-2017, and Google has subsequently corrected it). But was this problem limited to just one or two DNS resolvers, or do many other DNS resolvers experience a similar packet-drop issue? How widespread is this problem?

We used an experiment that tested resolver capabilities in handling DNS responses that entailed the use of fragmented UDP IPv6 packets. The experiment used a measurement script embedded in an online ad to enlist a large number of endpoints to perform resolution of a domain name[7]. For this measurement, we altered the DNS resolution system to fragment certain DNS responses.

The approach we took in this experiment was to use a user-level packet-processing system that listens on UDP port 53 and passes all incoming DNS queries to a back-end DNS server. When it receives a response from this back-end server it generates a sequence of IPv6 packets that fragments the DNS payload and uses a raw device socket to pass these packets directly to the device interface.

We are relying on the observation that IPv6 packet fragmentation occurs at the IP level in the protocol stack, so the IPv6 driver at the remote end will reassemble the fragments and pass the UDP payload to the DNS application, and if the resolver receives the payload packets , there will be no trace that the IPv6 packets were fragmented.

The results of this experiment follow:

- 10,851,323 experiments used IPv6 queries for the name server address.
- 6,786,967 experiments queried for the terminal DNS name.
- Fragmented response: 6,786,967/10,851,323 = 62.54% = 37.45% drop.

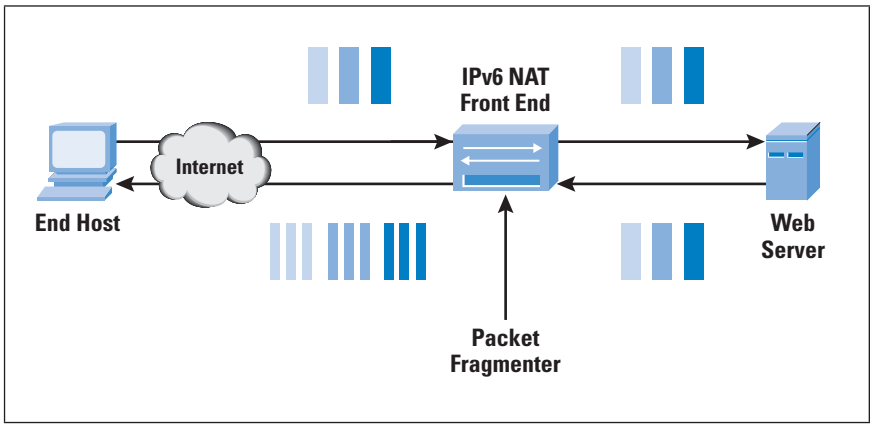
Some 37% of client endpoints used IPv6-capable DNS resolvers that were incapable of receiving a fragmented IPv6 response.

TCP and IPv6 Packet Fragmentation

The use of IPv6 Extension Headers implies that any transport protocol-sensitive functions within network equipment must follow the Extension Header chain of the packet header. This process takes a variable number of cycles for the device. It also requires that the device should recognise all the Extension Headers encountered on the header chain as passing through Extension Headers that the device either does not understand or is not prepared to check to determine whether or not it is a security risk. It's easier to drop all packets with Extension Headers! And that is what a lot of deployed equipment evidently does.

To measure the extent to which equipment drops fragmented IPv6 packets in TCP, we used a front-end unit to a web server and configured this front end to perform packet fragmentation on outbound packets as required. All TCP packets passed across the unit from the back end towards the Internet that contain a TCP payload larger than 15 octets are fragmented (Figure 1).

Figure 1: Experiment Configuration



The subsequent data-analysis phase can detect if the end host has received and successfully reassembled the set of fragments by looking at a log of packets. If an incoming TCP *Acknowledgement* (ACK) has a sequence number that encompasses the sending sequence number of outbound fragments within the same TCP session, that is evidence that the remote end has successfully reassembled the fragmented packet.

How "Real" Is This Experiment?

Before looking at the results, it may be useful to ask whether this experiment represents a "real" scenario that is commonly encountered on the Internet.

It's certainly the case that in TCP over IPv6 we do not expect to see packet fragmentation in the normal course of events.

A TCP sender should ensure that all outbound TCP segments fit within the local interface MSS size, so in the absence of network path MTU issues, a sender should not be fragmenting outbound TCP packets before sending them.

What about the case where the path MTU is smaller than the local interface MTU? When a packet encounters a network path next hop where the packet is larger than the next-hop MTU, the IPv6 router constructs an ICMPv6 PTB message, noting the size of the next hop, and also including the original packet headers as the payload of this ICMPv6 message. It sends this ICMPv6 diagnostic message back to the original sender and discards the original packet.

When a sending host receives this ICMPv6 PTB message, it also has the TCP packet header as part of the inner payload. This information can be used to find the local TCP control entry for this session, and the outbound MSS value of this TCP session can be updated with the new value. In addition to the updated size information, the TCP header in the ICMPv6 PTB message payload also contains the sequence number of the lost packet. The sending TCP process can interpret the ICMPv6 PTB message as an implicit *Negative Acknowledgement* (NACK) of the lost data, and resend the discarded data, using the updated MSS size. Again, no packet fragmentation is required.

All this sounds like a blatant case of "layer violation" and we should call in the Protocol Police. But before we do so, maybe we should think about the hypothetical situation where the host did not pass the ICMPv6 PTB message to the TCP control block. This situation is analogous to the case where the ICMPv6 PTB message is not passed to the host at all, where, for example, some unhelpful piece of network filtering middleware is filtering out all ICMPv6 messages.

In this case, the sending TCP session has sent a TCP segment and is waiting to receive an ACK. The receiver will not get this packet, so it cannot ACK it. The sender might have a retransmission timer and it might try to resend the offending large packet, but that too will get lost, so it will never get the ACK.

This situation results in a wedged TCP state, or a *Path MTU Black Hole* condition. Hiding ICMPv6 PTB messages from the TCP controller, either because of local processing rules within the host or because some network element has decided to drop them, is invariably harmful.

In that sense, we have constructed a somewhat "unreal" experiment, and we should not expect to see applications that critically depend on the correct working of packet fragmentation in TCP experiencing the same network conditions as those we've set up here.

On the other hand, fragmentation is an IP function, not a function performed by an end-to-end transport protocol. Therefore, the question of whether a host can receive a fragmented UDP packet is essentially the same question as whether a host can receive a fragmented TCP packet—at least from the perspective of the host itself. In both cases the real question is whether the IPv6 process on the host can receive fragmented IPv6 packets.

While the experiment itself uses conditions that are essentially an artifice, the result, namely the extent to which IPv6 Extension Header drop occurs when passing fragmented IPv6 packets towards end hosts, is nevertheless a useful and informative result.

Results

Over a period in August 2017, this experiment presented fragmented TCP packets to 1,702,949 unique IPv6 addresses. The results are summarized in Table 1.

Table 1: Results of Fragmentation Test

	Count	% of Total
Sent Fragmented TCP Packets	1,675,898	
Acknowledged Fragmented TCP Packets	1,324,834	79.03%
Failed to Acknowledge Fragmented TCP Packets	351,514	20.97%

Compared to the earlier DNS packet fragmentation result, namely that some 37% of endpoints who used IPv6-capable DNS resolvers used resolvers that were incapable of receiving IPv6 Fragmentation Extension Headers, the overall failure rate observed here of some 21% looks somewhat better. However, "better" is a relative term, as it is still the case that one-fifth of IPv6-capable endpoints are unable to receive a fragmented IPv6 packet.

Having one-fifth of the end-user population incapable of receiving fragmented large responses over IPv6 is indeed a serious problem.

Let's look briefly at the IPv6-over-IPv4 auto-tunnelling techniques Teredo and 6to4 (Table 2), as these two auto-tunnelled IPv6 bridging technologies just don't seem to want to die!

Table 2: Results of IPv6 Fragmentation Test for Teredo and 6to4 Prefixes

	Teredo	%	6to4	%
Sent Fragmented TCP Packets	53,780		24,384	
Acknowledged Fragmented TCP Packets	263	0.5%	1,486	6.1%
Failed to Acknowledge Fragmented TCP Packets	53,517	98.5%	22,898	93.9%

Both of these auto-tunnelling services are atrocious in this respect! Almost no Teredo endpoints can handle IPv6 fragmentation, and the 6to4 failure rate is not much better. Having no IPv6 at all is far better than having such a terrible service, and I can think of few better justifications for turning off the remaining Teredo and 6to4 gateways than these figures! What is even more depressing is that these two auto-tunnelling technologies represent one-quarter of the count of unique /64 prefixes seen in this experiment.

There is a considerable level of variation in the extent to which networks support the delivery of IPv6 Fragmentation Extension Headers to hosts. In some cases, it appears that the choice of customer premises equipment, or the configuration of IPv6 firewalls, may be a factor. Where the failure rate is very high it would point to the drop point being part of the behaviour of the provider network rather than the behaviour of the customer premises equipment.

Conclusions

Whatever the reasons, the conclusion here is unavoidable: IPv6 fragmentation is just not a viable component of the IPv6 Internet.

We need to adjust our protocols to avoid fragmentation.

For TCP, this adjustment should not be a major issue. Of course, this assertion relies on ICMPv6 PTB messages getting back to the sender's TCP process, but that is a major topic in its own right, so we won't delve deeper into this subject right now.

However, for UDP, this conclusion should be cause for some major rethinking of the way the DNS works, as the combination of DNSSEC, UDP, and IPv6 is really not going to work very well. It has implications for other UDP-based protocols as well, particularly where the protocol can generate large payloads.

The *Quick UDP Internet Connections* (QUIC) protocol, which uses a TCP-like control protocol embedded within a UDP encapsulation[8], has taken the pragmatic position of using a maximum packet size of 1,350 octets as a universal base and does not expect to encounter fragmentation issues given this somewhat conservative choice of packet size. If the DNS over IPv6 used a similar upper limit of UDP packet size and always sent back truncated responses for larger answers, we could probably avoid many of the packet-loss problems that we encounter today. Of course, the consequent larger use of TCP has its own implications in terms of query processing capacity for DNS resolvers and servers, so there are no free points here.

However, one conclusion looks starkly clear to me from these results. We can't just assume that the DNS as we know it today will just work in an all IPv6 future Internet. We must make some changes in some parts of the protocol design to get around this current widespread problem of IPv6 Extension Header packet loss in the DNS, assuming that we want to have a DNS at all in this all-IPv6 future Internet.

References and Further Reading

[0] J. Postel, "Internet Protocol," RFC 791, September 1981.

[1] Ron Bonica, Warren Kumari, Randy Bush, and Hagen Pfeifer, "IPv6 Fragment Header Deprecated," July 2013, Internet Draft, Work in Progress, **`draft-bonica-6man-frag-deprecate-02`**.

[2] Christopher A. Kent and Jeffrey C. Mogul, "Fragmentation Considered Harmful," Proceedings of Frontiers in Computer Communications Technology, ACM SIGCOMM '87, August 1987.

[3] Matt Mathis, Ben Chandler, and John W. Heffner, "IPv4 Reassembly Errors at High Data Rates," RFC 4963, July 2007.

[4] Ron Bonica, Fred Baker, Geoff Huston, Robert Hinden, Ole Troan, and Fernando Gont, "IP Fragmentation Considered Fragile," March 2018, Internet Draft, Work in Progress, **`draft-bonica-intarea-frag-fragile-01`**.

[5] Joel Jaeggli, Lorenzo Colitti, Warren Kumari, Eric Vyncke, Merike Kaeo, and Tom Taylor, "Why Operators Filter Fragments and What It Implies," December 2013, Internet Draft, Work in Progress, **`draft-taylor-v6ops-fragdrop-02`**.

[6] Fernando Gont, J. Linkova, and Tim Chown, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World," RFC 7872, June 2016.

[7] Geoff Huston, Joao Damas, and George Michaelson, "How we Measure IPv6," Presentation to APNIC 44 Conference, September 2017.

[8] Jana Iyengar and Martin Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," April 2018, Internet Draft, Work in Progress, **draft-ietf-quic-transport-11**.

[9] Pekka Savola, "MTU and Fragmentation Issues with In-the-Network Tunneling," RFC 4459, April 2006.

[10] G. Ziemba, D. Reed, and P. Traina, "Security Considerations for IP Fragment Filtering," RFC 1858, October 1995.

[11] Geoff Huston, "Fragmentation," *The Internet Protocol Journal,* Volume 19, No. 2, June 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. He is an active contributor to the Internet Engineering Task Force. At various times Geoff has worked as an Internet researcher, an ISP systems architect, and a network operator. E-mail: **gih@apnic.net**

Letters to the Editor

Ole and William,

As a loyal IPJ reader for decades, I think you did a terrific job on your blockchain piece in the November 2017 issue of IPJ. I don't think I have seen anything yet as comprehensive, understandable, and interesting to read on this subject. Thanks for writing and publishing the piece. There is far too much fluff and misinformation on blockchain and cryptocurrencies, so it is nice to see a solid treatment that provides helpful information to further everyone's understanding of this important technology.

— *David Strom,* **david@strom.com**

The author responds:

David,

Thank you for the kind words. A lot of credit goes to the reviewers, who did an awesome job of providing detailed feedback on the first draft.

—*Bill Stallings,* **ws@shore.net**

Ole,

I just re-read Geoff Huston's article on *Network Address Translation* (NAT) in the November 2017 issue (Volume 20, No. 3), and needed to applaud—possibly for the second time. Geoff is in a league of his own—the clear thinking, the pragmatism, the ability to communicate clearly and understandably—and not the least, the perspective on history. How we got here and why. Enlightening. NAT changed the Internet and continues to do so in ways I had never thought of until I read the article. Understanding that the 2018 Internet is vastly different in almost every possible way from the 1995 and 2005 Internet is extremely important. Otherwise we continue to plan for an historic model rather than the future. IPJ rocks!

—*Helge Skrivervik,* **helge@mymayday.com**

Ole,

As a data-networking engineer and architect of 20 years on the front line of fortune 100 network projects, I would like to offer a counter perspective from the recent article "In Defence of NATs" (Volume 20, No. 3).

The networking world seems to be losing sight that NAT is a "crutch" of sorts, a way of dealing with the primary problem in a lack of IPv4 address space.

By trying to justify NAT as a way to scale up IPv4 future potential scalability by "stealing" port/socket bits for something other than their originally intended purpose is nonsensical. One correction I would like to make on the article is the claim that NAT provides a firewall function. NAT and firewall functionality are mutually exclusive mechanisms, even if they are most often found on the same network device. NAT provides an obscured view of a host from elsewhere via address/port translation but does not by itself provide natural protection from the host on the other side of the translation point. Firewall protection encompasses the scrutinizing and controlling of the traffic that is allowed to traverse the NAT point. Firewalls provide this function with or without NAT, and NAT can function with or without a firewall mechanism.

Slow adoption of IPv6 has nothing to do with any perceived brilliant nature of NAT, and NAT does present real-world problems for several types of very important software; Microsoft's *Active Directory Replication* and IBM's *Virtual Tape Library* (VTL)/*Virtual Tape Server* (VTS) are two examples off the top of my head. When traversing a NAT point, many applications that share their host IP address in the data field with other hosts require active "swapping" of this payload-imbedded IP address or another compensating mechanism. The communication would "break" the application's intended communication model without the addition of a compensating mechanism.

The question in the article "should I deploy IPv6 now?" is the wrong question. The Internet was first "born" in a practical sense at the point it became commercially available to the world to use. In the first 25 years it grew at an amazing rate, doubling its size several times over in a very natural and organic way. It has been almost a quarter century since IPv6 was released, and its resistance has been *significant* for good reasons beyond the scope of this letter. IPv6 is "The Emperor's New Clothes," otherwise IPv4 would have been replaced by now if IPv6 had been a natural and organic progression of IPv4, and there would be no need to "defend" NAT or speak up against its forced ubiquitous overuse. Someone once told me that IPv6 was here to stay. To my way of thinking it has not yet arrived after almost a quarter century.

While IPv4 port/socket numbers can be seen as "borrowed address bits" for NAT, I believe it is a distorted view of port/socket intended use. Fields and protocols are defined and delineated for a reason. If you wish to repurpose bits—for example, *Type of Service* (ToS) to *Differentiated Services Code Point* (DSCP) use—then repurpose them officially. Until then, fields and protocols should be respected as originally intended and not subject to implied de-facto depreciation by NAT's liberal theft. Field definitions and structure have purpose.

The sirens' song that I believe we collectively are starting to fall for is that NAT is a "one-size-fits-all" solution for all forms of network scalability in a ubiquitous way. This is not the real-world case. Resource hosts need (in a practical sense) globally unique Layer 3 identifiers. Consider corporate mergers/acquisitions as well as divestitures leading to merging of a divested entity. Trying to merge two significant company networks together that both use NAT RFC 1918 on the "inside" for resource hosts is overly complex in a way that it would not have to be if a viable replacement for IPv4 had been rolled out in a manner that world corporations could embrace commercially. That protocol does not exist. While I agree with the notion that the Internet cannot be "stateless," this does not uniquely justify NAT as "middleware." End hosts ideally should be ultimate keepers of their stateful connections; justifying NAT just for the sake of IPv4 continued life-support is nonsensical.

NAT has a proper use; middleware that uses NAT as a complementary protocol along with others, such as "load balancing" [for example, the *Local Traffic Manager* (LTM) product by F5] is justifiable, but in this case application scalability and fault tolerance encompass the direct purpose of this middleware, not compensating for world IPv4 address depletion. Other forms of network middleware are perfectly justifiable, even if NAT did not need to exist; firewalls and *Intrusion Prevention Systems* (IPS) are examples. We should appreciate NAT for its role as a "tactical" compensating mechanism for IPv4 address space depletion, not as a "strategic future-proofing" scalability mechanism for IPv4. People with broken legs appreciate a crutch, but would not appreciate needing to use a crutch for the rest of their lives if their body decided not to heal itself because the body viewed the crutch as "good enough." NAT seen as a long-term way of extending IPv4 scalability and, therefore, lifespan is just putting lipstick on the IPv4 pig.

NAT is a mechanism to be used (like any other protocol) where it makes sense to use it and no further. If IPv6 or some other more sensible replacement for IPv4 were completely rolled out with IPv4 relegated to the history books, then the practical use of NAT in such a future environment would be a single-digit fraction of its current existence. That existence would be primarily in terms of resiliency mechanisms such as load balancing, as previously mentioned, and certain (client) mobility cases. One of the most memorable pieces of wisdom I have ever heard about IT applies here very well: "There is nothing more permanent than a temporary solution." Let us not fall victim to this easy psychological trap only because we seem to have collectively painted ourselves into a corner of sorts.

—*Leroy Harvey,* `leroy.harvey@hotmail.com`

The author responds:

The purpose of any opinion piece is to provoke the reader into thinking about the issue, and perhaps looking at it from a set of different perspectives. For more than two decades the *Internet Engineering Task Force* (IETF) viewed NATs as a somewhat ugly hack, and from time to time attempted to discourage its use in various ways. However, the undeniable observation is that NATs keep today's Internet running. Perhaps there is more to NATs than a rather ugly short-term hack that should disappear. What if they are here to stay? The opinion piece was intended to look at NATs from a perspective that shared little with the orthodox view of NATs, and provoke readers to think about this unanticipated direction that the Internet has taken and wonder where it may lead. I'm pleased to see that this provocation has motivated one reader to provide a thoughtful response.

—Geoff Huston, APNIC
gih@apnic.net

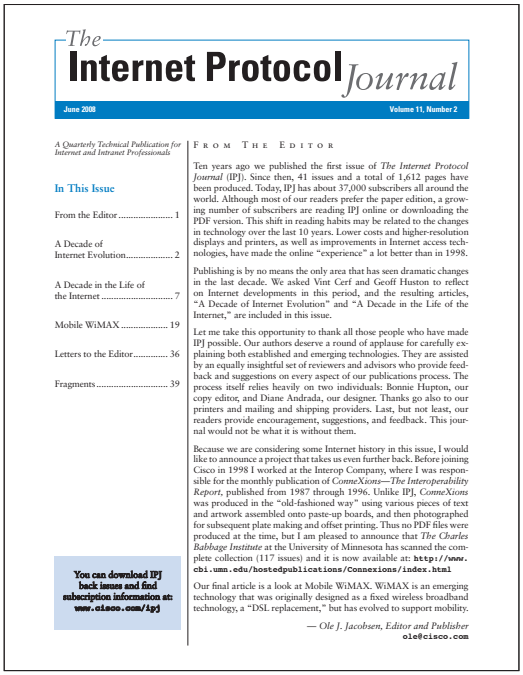
Letters may be edited for clarity. We'd love to hear from you. Send us your feedback via e-mail to **ipj@protocoljournal.org**

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

Coming Soon: Our 20th Anniversary Issue

It is difficult to believe, but another decade has passed and in June we will celebrate 20 years of *The Internet Protocol Journal*. Make sure your subscription is up-to-date so you don't miss this issue!

Ten years ago:



The Internet Protocol Journal

June 2008 — Volume 11, Number 2

A Quarterly Technical Publication for Internet and Intranet Professionals

In This Issue

You can download IPJ back issues and find subscription information at: www.cisco.com/ipj

FROM THE EDITOR

Ten years ago we published the first issue of *The Internet Protocol Journal* (IPJ). Since then, 41 issues and a total of 1,612 pages have been produced. Today, IPJ has about 37,000 subscribers all around the world. Although most of our readers prefer the paper edition, a growing number of subscribers are reading IPJ online or downloading the PDF version. This shift in reading habits may be related to the changes in technology over the last 10 years. Lower costs and higher-resolution displays and printers, as well as improvements in Internet access technologies, have made the online "experience" a lot better than in 1998.

Publishing is by no means the only area that has seen dramatic changes in the last decade. We asked Vint Cerf and Geoff Huston to reflect on Internet developments in this period, and the resulting articles, "A Decade of Internet Evolution" and "A Decade in the Life of the Internet," are included in this issue.

Let me take this opportunity to thank all those people who have made IPJ possible. Our authors deserve a round of applause for carefully explaining both established and emerging technologies. They are assisted by an equally insightful set of reviewers and advisors who provide feedback and suggestions on every aspect of our publications process. The process itself relies heavily on two individuals: Bonnie Hupton, our copy editor, and Diane Andrada, our designer. Thanks go also to our printers and mailing and shipping providers. Last, but not least, our readers provide encouragement, suggestions, and feedback. This journal would not be what it is without them.

Because we are considering some Internet history in this issue, I would like to announce a project that takes us even further back. Before joining Cisco in 1998 I worked at the Interop Company, where I was responsible for the monthly publication of *ConneXions—The Interoperability Report*, published from 1987 through 1996. Unlike IPJ, *ConneXions* was produced in the "old-fashioned way" using various pieces of text and artwork assembled onto paste-up boards, and then photographed for subsequent plate making and offset printing. Thus no PDF files were produced at the time, but I am pleased to announce that *The Charles Babbage Institute* at the University of Minnesota has scanned the complete collection (117 issues) and it is now available at: **http://www.cbi.umn.edu/hostedpublications/Connexions/index.html**

Our final article is a look at Mobile WiMAX. WiMAX is an emerging technology that was originally designed as a fixed wireless broadband technology, a "DSL replacement," but has evolved to support mobility.

— Ole J. Jacobsen, Editor and Publisher
ole@cisco.com

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
Scott Aitken
Antonio Cuñat Alario
Matteo D'Ambrosio
Jens Andersson
Danish Ansari
Tim Armstrong
Richard Artes
David Atkins
Jaime Badua
John Bigrow
Axel Boeger
Gerry Boudreaux
Kevin Breit
Ilia Bromberg
Christophe Brun
Gareth Bryan
Stefan Buckmann
Scott Burleigh
Jon Harald Bøvre
Olivier Cahagne
Tracy Camp
Fabio Caneparo
Roberto Canonico
John Cavanaugh
Lj Cemeras
Dave Chapman
Stefanos Charchalakis
Greg Chisholm
Brad Clark
Narelle Clark
Steve Corbató
Brian Courtney
Dave Crocker
Kevin Croes
John Curran
André Danthine
Morgan Davis
Freek Dijkstra
Geert Van Dijk
Richard Dodsworth
Ernesto Doelling
Eugene Doroniuk
Karlheinz Dölger
Andrew Dul
Holger Durer
Peter Robert Egli
George Ehlers
Peter Eisses
Torbjörn Eklöv
ERNW GmbH
ESdatCo
Steve Esquivel
Mikhail Evstiounin
Paul Ferguson
Kent Fichtner
Gary Ford
Christopher Forsyth
Craig Fox
Fausto Franceschini
Tomislav Futivic
Edward Gallagher
Andrew Gallo
Chris Gamboni
Xosé Bravo Garcia
Kevin Gee
Serge Van Ginderachter
Greg Goddard
Octavio Alfageme Gorostiaga
Barry Greene
Martijn Groenleer
Geert Jan de Groot
Gulf Coast Shots
Sheryll de Guzman
James Hamilton
Stephen Hanna
Martin Hannigan
John Hardin
David Harper
Edward Hauser
David Hauweele
Headcrafts SRLS
Johan Helsingius
Robert Hinden
Alain Van Hoof
Edward Hotard
Bill Huber
Hagen Hultzsch
Mika Ilvesmaki
Karsten Iwen
Ashford Jaggernauth
David Jaffe
John Jarvis
Dennis Jennings
Edward Jennings
Aart Jochem
Richard Johnson
Jim Johnston
Jonatan Jonasson
Daniel Jones
Gary Jones
Amar Joshi
Merike Kaeo
Andrew Kaiser
David Kekar
Shan Ali Khan
Nabeel Khatri
Anthony Klopp
Henry Kluge
Andrew Koch
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
Richard Lamb
Tracy LaQuey Parker
Simon Leinen
Robert Lewis
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
Timothy Martin
Gabriel Marroquin
Carles Mateu
Juan Jose Marin Martinez
Ioan Maxim
Miles McCredie
Brian McCullough
Joe McEachern
Jay McMaster
Carsten Melberg
Kevin Menezes
Bart Jan Menkveld
William Mills
Desiree Miloshevic
Thomas Mino
Mohammad Moghaddas
Charles Monson
Andrea Montefusco
Fernando Montenegro
Soenke Mumm
Tariq Mustafa
Stuart Nadin
Mazdak Rajabi Nasab
Krishna Natarajan
Darryl Newman
Marijana Novakovic
Ovidiu Obersterescu
Mike O'Connor
Mike O'Dell
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
Jorge Ivan Pincay Ponce
Blahoslav Popela
Tim Pozar
David Raistrick
Priyan R Rajeevan
Paul Rathbone
Bill Reid
Rodrigo Ribeiro
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
Phil Scarr
Jeroen Van Ingen Schenau
Carsten Scherb
Roger Schwartz
SeenThere
Scott Seifel
Yury Shefer
Yaron Sheffer
Tj Shumway
Jeffrey Sicuranza
Thorsten Sideboard
Andrew Simmons
Henry Sinnreich
Geoff Sisson
Helge Skrivervik
Darren Sleeth
Bob Smith
Mark Smith
Job Snijders
Asit Som
Ignacio Soto Campos
Peter Spekreijse
Thayumanavan Sridhar
Matthew Stenberg
Adrian Stevens
Clinton Stevens
John Streck
Viktor Sudakov
Edward-W. Suor
Vincent Surillo
Roman Tarasov
David Theese
Sandro Tumini
Phil Tweedie
Steve Ulrich
Unitek Engineering AG
John Urbanek
Martin Urwaleck
Betsy Vanderpool
Surendran Vangadasalam
Alejandro Vennera
Luca Ventura
Tom Vest
Dario Vitali
Randy Watts
Andrew Webster
Tim Weil
Jd Wegner
Rick Wesson
Peter Whimp
Jurrien Wijlhuizen
Pindar Wong
Bernd Zeimetz
廖 明沂.

Follow us on Twitter and Facebook

@protocoljournal

https://www.facebook.com/newipj

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**

The Internet Protocol Journal is published under the "CC BY-NC-ND" Creative Commons Licence. Quotation with attribution encouraged.

This publication is distributed on an "as-is" basis, without warranty of any kind either express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. This publication could contain technical inaccuracies or typographical errors. Later issues may modify or update information provided in this issue. Neither the publisher nor any contributor shall have any liability to any person for any loss or damage caused directly or indirectly by the information contained herein.

Supporters and Sponsors

Supporters

Internet Society

cisco

Diamond Sponsors

Google

ICANN

Ruby Sponsor

RIPE NCC
RIPE NETWORK COORDINATION CENTRE

Sapphire Sponsors

Your logo here!

Emerald Sponsors

Afilias

APNIC

APRICOT
Asia Pacific Regional Internet Conference on Operational Technologies

cisco

CNNIC
中国互联网络信息中心
CHINA INTERNET NETWORK INFORMATION CENTER

COMCAST

DE-CIX

EQUINIX

JPRS
JAPAN REGISTRY SERVICES

JUNIPER NETWORKS

lacnic

LinkedIn

linx

NSRC
Network Startup Resource Center

NTT Communications

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

netnod

qa|cafe

SIDN

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
Ole J. Jacobsen, Editor and Publisher

Editorial Advisory Board
Dr. Vint Cerf, VP and Chief Internet Evangelist
Google Inc, USA

David Conrad, Chief Technology Officer
Internet Corporation for Assigned Names and Numbers

Dr. Steve Crocker, CEO and Co-Founder
Shinkuro, Inc.

Dr. Jon Crowcroft, Marconi Professor of Communications Systems
University of Cambridge, England

Geoff Huston, Chief Scientist
Asia Pacific Network Information Centre, Australia

Dr. Cullen Jennings, Cisco Fellow
Cisco Systems, Inc.

Olaf Kolkman, Chief Internet Technology Officer
The Internet Society

Dr. Jun Murai, Founder, WIDE Project, Dean and Professor
Faculty of Environmental and Information Studies,
Keio University, Japan

Pindar Wong, Chairman and President
Verifi Limited, Hong Kong

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

The title "The Internet Protocol Journal" is a trademark of Cisco Systems, Inc. and/or its affiliates ("Cisco"), used under license. All other trademarks mentioned in this document or website are the property of their respective owners.

Printed in the USA on recycled paper.