The Internet Protocol Journal

June 2015 Volume 18, 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.protocoljournal.org

ISSN 1944-1134

From The Editor

The *Transmission Control Protocol* (TCP) is one of the core protocols used in today's Internet. This issue of IPJ is almost entirely devoted to discussions about TCP. Anyone who has studied TCP/IP will have marveled at the "ASCII Art" state diagram for TCP on page 23 of RFC 793, published in 1981. This diagram is a good illustration of both the power and the limitations of using only text characters to draw a "picture." I am happy to report that efforts to define a new format for the RFC series of documents are nearing completion. We will report further on this new RFC format in a future issue.

Your mobile device contains several interfaces, such as USB, WiFi, Cellular Data, and Bluetooth. Most, if not all, of these interfaces can be used for Internet communications, specifically to carry TCP/IP datagrams. In our first article, Geoff Huston looks at an emerging standard, *Multipath TCP* (MPTCP), which allows TCP to operate several simultaneous connections using *different* interfaces.

Although TCP has not fundamentally changed since its introduction in 1981, much work has gone into improving TCP performance in the presence of network congestion and variations in network throughput. Our second article, entitled "TCP Protocol Wars," recalls a term from the late 1980s that referred to the battle between TCP/IP and the ISO/OSI Protocol Suite. This time, the term is used more humorously to compare the many special implementations and refinements to TCP.

If you received a printed copy of this journal in the mail, you should also have received a subscription activation e-mail with information about how to update and renew your subscription. If you didn't receive such a message, it may be because we do not have your correct e-mail address on file. To update and renew your subscription, just send a message to **ipj@protocoljournal.org** and include your subscription ID. Your subscription ID and expiration date are printed on the back of your journal.

Let me once again remind you that IPJ relies on the support of numerous individuals and organizations. If you or your company would like to sponsor IPJ, please contact us for further details. Our website at **protocoljournal.org** contains all back issues, subscription information, a list of current sponsors, and much more.

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

IP Multi-Adressing and Multipath TCP

by Geoff Huston, APNIC

The *Transmission Control Protocol* (TCP) is a core protocol of the Internet networking protocol suite. This protocol transforms the underlying unreliable datagram delivery service provided by the *Internet Protocol* (IP) into a reliable data stream protocol. For me this protocol was the single greatest transformative moment in the evolution of computer networks.

Prior to TCP, computer network protocols assumed that computers wanted a lossless reliable service from the network, and worked hard to provide it. The *Digital Data Communications Message Protocol* (DDCMP) in Digital Equipment Corporation's *DECnet* was a lossless data link control protocol. X.25 in the telecommunications world provided reliable stream services to the attached computers. Indeed, I recall that Ethernet was criticized when it was introduced to the world because of its lack of a reliable acknowledgement mechanism. TCP changed all of that. TCP pushed all of the critical functionality supporting reliable data transmission right out of the network and into the shared state of the computers at each end of the TCP conversation. TCP embodies the *End-to-End Principle* of the Internet architecture, where there is no benefit in replicating within the network functionality that can be provided by the end points of a conversation. What TCP required of the network was a far simpler service where packets were allowed to be delivered out of order, but packets could be dropped and TCP would detect and repair the problem and deliver to the far-end application precisely the same bit stream that was passed into the TCP socket in the first place.

The TCP protocol is now some 40 years old, but that doesn't mean that it has been frozen over all these years.

TCP is not only a reliable data stream protocol, but also a protocol that uses *Adaptive Rate Control*. TCP can operate in a mode that allows the protocol to push as much data through the network as it can. A common mode of operation is for an individual TCP session to constantly probe into the network to see what the highest sustainable data rate is, interpreting packet loss as the signal to drop the sending rate and resume the probing. This aspect of TCP has been a constant field of study, and much work has been done in the area of flow control. We now have many variants of TCP that attempt to optimize the flow rates across various forms of networks.

Other work has looked at the TCP data acknowledgement process, attempting to improve the efficiency of the algorithm under a broad diversity of conditions. *Selective Acknowledgments* (SACK) allowed a receiver to send back more information to the sender in response to missing data. *Forward Acknowledgment* (FACK) addresses data-loss issues during TCP *Slow Start*.

One approach to trying to improve the relative outcome of a data transfer, as compared to other simultaneously open TCP sessions, is to split the data into multiple parts and send each part in its own TCP session. This splitting effectively opens up numerous parallel TCP sessions. A variant of TCP, *MulTCP,* emulates the behavior of multiple parallel TCP sessions in a single TCP session. These behaviors assume the same endpoints for the parallel TCP sessions and assume the same end-to-end path through the network. An evolution of TCP that uses multiple parallel sessions but tries to spread these sessions across multiple *paths* through the network is *Multipath TCP* (MPTCP).

Multipath TCP had a brief moment of prominence when it was revealed that Apple's release of iOS 7 contained an implementation of Multipath TCP for the company's *Siri* application, but it has the potential to play a bigger role in the mobile Internet. In this article, I will explore this TCP option in a little more detail, and see how it works and how it may prove to be useful in today's mobile networks.

Multi-Addressing in IP

First we need to return to one of the basic concepts of networking, that of addressing and addresses. Addresses in the Internet Protocol were subtly different from many other computer communications protocols that were commonly used in the 1970s and 1980s. While many other protocols used the communications protocol-level address as the address of the host computer, the Internet Protocol was careful to associate an IP address with the *interface* to a network. This distinction was a relatively unimportant one in most cases because computers usually had only a single network attachment interface. But it was a critical distinction when the computer had two or more interfaces to two or more networks. An IP host with two network interfaces has two IP addresses—one for each interface. In IP it is the interface between the device and the network that is the addressed endpoint in a communication. An IP host accepts an IP packet as being addressed to itself if the IP address in the packet matches the IP addresses of the network interface that received the packet, and when sending a packet, the source address in the outgoing packet is the IP address of the network interface that was used to pass the packet from the host into the network.

As simple as this model of network addressing may be, it does present some operational problems. One implication of this form of addressing is that when a host has multiple interfaces, the application-level conversations using TCP are "sticky." If, for example, a TCP session was opened on one network interface, the network stack in the host cannot quietly migrate this active session to another interface while maintaining the common session state. An attempt by one "end" of a TCP conversation to change the IP address for an active session would not normally be recognized at the other end of the conversation as being part of the original session. So having multiple interfaces and multiple addresses does not create additional resiliency of TCP connections.

The simplicity of giving each network interface a unique IP address does not suit every possible use case, and it was not all that long before the concept of *secondary addresses* came into use. The use of secondary addresses was a way of using multiple addresses to refer to a host by allowing a network interface to be configured with multiple IP addresses. In this scenario, an interface receives packets addressed to any of the IP addresses associated with the interface. Outgoing packet handling allows the transport layer to specify the source IP address, and this action overrides the default action of using the primary address of the interface on outgoing packets. Secondary addresses have their uses, particularly when you are trying to achieve the appearance of multiple application-level "personas" on a single common platform, but in IPv4 they were perhaps more of a specialized solution to a particular family of requirements, rather than a commonly used approach. Applications using TCP were still "sticky" with IP addresses that were used in the initial TCP handshake and could not switch the session between secondary IP addresses on the same interface.

IPv6 addressing is somewhat different. The protocol allows from the outset for an individual interface to be assigned multiple IPv6 unicast addresses without the notion of "primary" and "secondary" addresses. The IPv6 protocol introduces the concept of an address *scope,* so an address may be assuredly unique in the context of the local link-layer network, or it may have a global scope, for example. Privacy considerations have also introduced the concept of permanent and temporary addresses, and the efforts to support a certain form of mobility have introduced the concepts of *home addresses* and *care-of addresses*.

However, to some extent these IPv6 changes are cosmetic modifications to the original IPv4 address model. If an IPv6 host has multiple interfaces, each of these interfaces has its own set of IPv6 addresses, and when a TCP session is started using one address pair TCP does not admit the ability to shift to a different address pair in the life of the TCP session. A TCP conversation that started over one network interface is stuck with that network interface for the life of the conversation, whether it's IPv4 or IPv6.

The Internet has changed significantly with the introduction of the mobile Internet, and the topic of multi-addresses is central to many of the problems with mobility. Mobile devices are adorned with many IP addresses. The cellular radio interface has its collection of IP addresses. Most of these "smart" devices also have a WiFi interface that also has its set of IPv4 and possibly IPv6 addresses. And there may be a Bluetooth network interface with IP addresses, and perhaps some USB network interface as well. When active, each of these network interfaces requires its own local IP address. We now are in an Internet where devices with multiple active interfaces and multiple usable IP addresses are relatively commonplace. But how can we use these multiple addresses?

For many scenarios there is little value in being able to use multiple addresses. The conventional behavior is where each new session is directed to a particular interface, and the session is given an outbound address as determined by local policies. However, when we start to consider applications in which the binding of location and identity is more fluid, network connections are transient, and the cost and capacity of connections differ (as is often the case in today's mobile cellular radio services and in WiFi roaming services), then having a session that has a certain amount of agility to switch across networks can be a significant factor.

If individual end-to-end sessions could use multiple addresses, and by inference could use multiple interfaces, then an application could perform a seamless handoff between cellular data and WiFi, or even use both at the same time. Given that the TCP interface to IPv4 and IPv6 is identical, it is even quite feasible to contemplate a seamless handoff between the two IP protocols. The decision as to which carriage service to use at any time would no longer be a decision of the mobile carrier or that of the WiFi carrier, or that of the device, or that of its host operating system. If applications could use multiple addresses, multiple protocols, and multiple interfaces, then the decision could be left to the application itself to determine how best to meet its needs as connections options become available or as they shut down. At the same time as the debate between traditional mobile operators in the licensed spectrum space and the WiFi operators in the unlicensed spectrum space heats up over access to the unlicensed spectrum, the very nature of how devices and applications implement "WiFi handoff" is changing. Who is in control of this handoff function is changing as a result. Multi-Addressing and Multipath TCP is an interesting response to this situation; it allows individual applications to determine how they want to operate in a multi-connected environment.

SHIM6

One of the first attempts to use multiple addresses in IP was the *Site Multihoming by IPv6 Intermediation* (SHIM6) effort in IPv6.

In this case the motivation was end-site resilience in an environment of multiple external connections, and the constraint was to avoid the use of an independently routed IPv6 address prefix for the site. So SHIM6 was an effort to support site multi-homing without routing fragmentation. To understand the SHIM6 model, we need to start with an end site that does not have its own provider-independent IPv6 address prefix, yet is connected to two or more upstream transit providers that each provide addresses to the end site. In IPv4 it's common to see this scenario approached with *Network Address Translators* (NATs). In IPv4 the site is internally addressed using a private address prefix, and the interface to each upstream provider is provisioned with a NAT. Outbound packets have their source address rewritten to use an address that is part of the provider's prefix as it transits the NAT.

Which provider is used is a case of internal routing policies toward each of the NATs. Although it is possible to configure a similar setup in IPv6 using an IPv6 *Unique Local Address* (ULA) prefix as the internal address and NAT IPv6-to-IPv6 devices connected to each upstream service provider, one of the concepts behind IPv6 and its massive increase in address space was the elimination of NATs. So how can an IPv6 end site be homed into multiple upstream service providers without needing to advertise a more specific routing entry in the interdomain routing tables and avoiding the use of any form of network address translation?

The conventional IPv6 architecture has the site receiving an end-site prefix delegation from each of its upstream service providers, and the interface routers each advertising its end-site prefix into the site. Hosts within the site see both router advertisements, and they configure their interface with multiple IPv6 addresses, one for each site prefix. Presumably, the end site chooses to multi-home in order to benefit from the additional resiliency that such a configuration should offer. When the link to one provider is down, there is a good chance that the other link will remain up, particularly if the site has been careful to engineer the multi-homed configuration using discrete components at every level. It would be even better if even when the link to the upstream provider is up and that provider can't reach a specific destination, another of the site's upstream providers could continue to support all active end-to-end conversations without interruption, in exactly the same manner as when this functionality is implemented in the routing system.

What SHIM6 attempted was a host-based approach to use the additional local IPv6 addresses in the host as indicators of potential backup paths to a destination. If a communication with a remote counterpart were to fail (that is, the flow of incoming packets from the remote host stopped), then the IP-level shim in the local host would switch to use a different source/destination address pair. To prevent the upper-level transport protocol from being fatally confused by these address changes in the middle of one or more active sessions, the local SHIM module also included a network address translation function. This function helped ensure that although the address pair on the wire may have changed, the address pair presented to the upper layer by the shim would remain constant, and the path change would not be directly visible at the transport layer of the protocol stack.

This approach essentially folds the NAT function into the host IP protocol stack. In terms of design it avoided altering either TCP or *User Datagram Protocol* (UDP), and endeavoured to preserve the IP addresses used by active transport sessions. What this approach implied was that if you wanted to change the routing path but not change the IP addresses used by transport, then address translation was an inevitable consequence.

Network-based NATs was the response in IPv4, and to avoid this problem in IPv6 the SHIM6 effort attempted to push the NAT functionality further "back," implementing a NAT in each host.

SHIM6 was an approach that was less than entirely satisfactory.

Network operators expressed deep distrust of pushing decision-making functionality back into individual hosts (a distrust that network operators continue to hold when the same issue arises with WiFi handoff). The network operators wanted to control the connectivity structure for the hosts in their network, in precisely the same manner as the routing system provided network-level control over traffic flows. So although these network operators had some sympathy with the SHIM6 objective of avoiding further bloat in the routing table, which reduced the "independence" of attached end sites by using IPv6 address prefixes drawn from the upstream address block, they were unsupportive of an approach that pushed connectivity choice and control back to individual end host systems.

Outside of this issue of control over the end host was another multi-homing problem that SHIM6 did not address. Although the provision of backup paths in the case of failure of the primary path is useful, what is even more useful is the ability to use the backup paths in some form of load-sharing configuration. However, at this point the SHIM6 approach runs into problems. Because SHIM6 operates at the IP layer, it is not directly aware of packet sequencing. When a SHIM unit at one end of a conversation splays a sequence of packets across multiple paths, the corresponding SHIM unit at the remote end passes the packets into the upper transport layer in the order of their arrival, not in the original order. This out-of-order delivery can be a significant problem for TCP if SHIM6 leaves multiple paths open. The best SHIM6 can provide is a primary/backup model for individual sessions, where at any time all data traffic for a session is passed along the primary path.

Inexorably, we are drawn to the conclusion that the most effective place to insert functionality that allows a data flow to use multiple potential paths across the network is in the transport layer itself, and ne weed to jack ourselves further up the protocol stack from the IP level approach of SHIM6 and re-examine the space from the perspective of TCP.

Multipath TCP

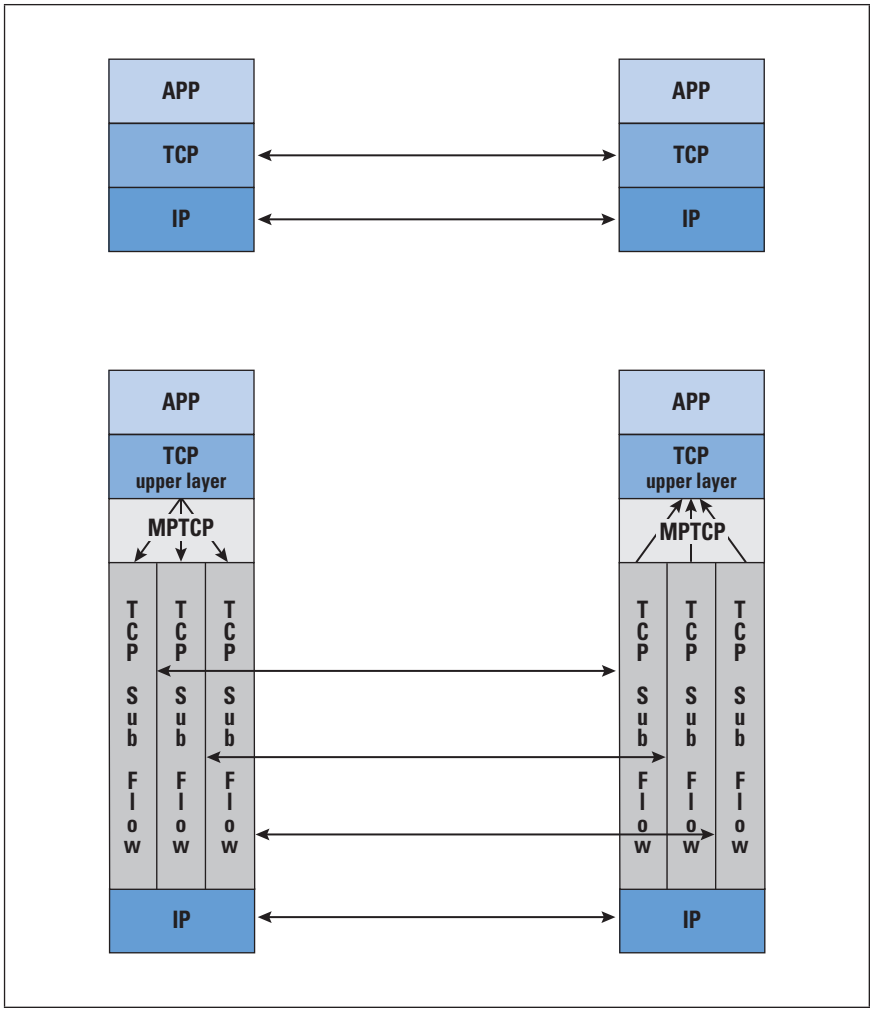
The approach of incorporating multiple IP addresses in the transport protocol is comparable to the efforts of SHIM6 one level further down in the protocol stack, in so far as this approach is an end-to-end mechanism with a shared multiplex state maintained in the two end hosts, and no state whatsoever in the network.

The basic mechanisms for MPTCP are also similar to that of SHIM6, with an initial capability exchange to confirm that both parties support the mechanism, allowing the parties to then open up additional paths, or channels. But at this point the functionality diverges. In SHIM6 these alternate paths are provisioned as backup paths if the primary path fails, whereas in the case of MPTCP these additional paths can be used immediately to spread the load of the communication across these paths, if the application so desires.

One of the most critical assumptions of MPTCP was drawn from SHIM6, in that the existence of multiple addresses in a host is sufficient to indicate the existence of multiple diverse paths within the network. Whether or not this assumption is, in fact, the case is perhaps not that critical, in that even in the case where the addresses are on the same path from end to end, the result is roughly equivalent to running multiple parallel sessions of TCP.

The basic approach to MPTCP is the division of the single outbound flow of the application into multiple *subflows*, each operating its own end-to-end TCP session, and the rejoining of multiple input subflows into a single flow to present to the remote counterpart application. This approach is shown in Figure 1.

Figure 1: Comparison of Standard TCP and MPTCP Protocol Stacks



This solution is essentially a "shim" inserted in the TCP module. To the upper-level application, MPTCP can operate in a manner that is entirely consistent with TCP, so that the opening up of subflows and the manner in which data is assigned to particular subflows is intentionally opaque to the upper-level application. The envisaged *Application Programming Interface* (API) allows the application to add and remove addresses from the local multipath pool, but the remainder of the operation of the MPTCP shim is not envisaged to be managed directly by the application. MPTCP also leaves the lower-level components of TCP essentially untouched, in so far as each MPCTP subflow is a conventional TCP flow. On the data sender's side, the MPTCP shim essentially splits the received stream from the application into blocks and directs individual blocks into separate TCP subflows. On the receiver's side, the MPTCP shim assembles the blocks from each TCP subflow and reassembles the original data stream to pass to the local application.

Operation of MPTCP

TCP has the ability to include 40 bytes of TCP *options* in the TCP header, indicated by the *Data Offset* value. If the Data Offset value is greater than 5, then the space between the final 32-bit word of the TCP header (*Checksum* and *Urgent Pointer*) and the first octet of the data can be used for options. MPTCP uses the *Option Kind* value of 30 to denote MPTCP options. All MPTCP signalling is contained in this TCP header options field.

The MPTCP operation starts when the initiating host passes a MP_CAPABLE capability message in the MPTCP options field to the remote host as part of the initial TCP SYN message when opening the TCP session. The SYN+ACK response contains a MP_CAPABLE flag in its MPTCP options field of the SYN+ACK response if the other end is also MPTCP-capable. The combined TCP and MPTCP handshake concludes with the ACK and MP_CAPABLE flag, confirming that both ends now have each other's MPTCP session data. This capability negotiation exchanges 64-bit keys for the session, and each party generates a 32-bit hash of the session keys, which are subsequently used as a shared secret between the two hosts for this particular session to identify subsequent subjoin connection attempts.

Further TCP subflows can be added to the MPTCP session by a conventional TCP SYN exchange with the MPTCP option included. In this case the exchange contains the MP_JOIN values in the MPTCP options field. The values in the MP_JOIN exchange include the hash of the original receiver's session key and the token value from the initial session, so that both ends can associate the new TCP session with the existing session, as well as a random value intended to prevent replay attacks.

The MP_JOIN option also includes the sender's address index value to allow both ends of the conversation to reference a particular address even when NATs on the path perform address transforms. MPTCP allows these MP_JOINs to be established on any port number, and by either end of the connection. Therefore, although a MPTCP web session may start using a port 80 service on the server, subsequent subflows may be established on any port pair, and it is not necessary for the server to have a LISTEN open on the new port. The MPTCP session token allows the 5-tuple of the new subflow (protocol number, source and destination addresses, and source and destination port numbers) to be associated with the originally established MPTCP flow. Two hosts can also inform each other of new local addresses without opening a new session by sending ADD_ADDR messages, and remove them with the complementary REMOVE_ADDR message.

Individual subflows use conventional TCP signalling. However, MPTCP adds a *Data Sequence Signal* (DSS) to the connection that describes the overall state of the data flow across the aggregate of all of the TCP subflows that are part of this MPTCP session. The sender sequence numbers include the overall data sequence number and the subflow sequence number that is used for the mapping of this data segment into a particular subflow. The DSS Data ACK sequence number is the aggregate acknowledgement of the highest in-order data that the receiver receives. MPTCP does not use SACK, because this acknowledgement is left to the individual subflows.

To prevent data loss that causes blockage on an individual subflow, a sender can retransmit data on additional subflows. Each subflow uses a conventional TCP sequencing algorithm, so an unreliable connection will cause that subflow to stall. In this case MPTCP can use a different subflow to resend the data, and if the stalled condition is persistent it can reset the stalled subflow with a TCP RST within the context of the subflow.

Individual subflows are stopped by a conventional TCP exchange of FIN messages, or through the TCP RST message. The shutting down of the MPTCP session is indicated by a data FIN message that is part of the data sequencing signalling within the MPTCP option space.

Congestion control appears still to be an open issue for MPTCP. An experimental approach is to couple the congestion windows of each of the subflows, increasing the sum of the total window sizes at a linear rate per *Round-Trip Time* (RTT) interval, and applying the greatest increase to the subflows with the largest existing window. In this way the aggregate flow is no worse than a single TCP session on the best available path, and the individual subflows take up a fair share of each of the paths they use. Other approaches are being considered that may reduce the level of coupling of the individual subflows.

MPTCP and Middleware

Today's Internet is not the Internet of old. It is replete with various forms of middleware that include NATs, load balancers, traffic shapers, proxies, filters, and firewalls. The implication of this reality is that any deviation from the most basic forms of the use of IP will run into various issues with various forms of middleware.

For MPTCP, the most obvious problem is that of middleware that strips out unknown TCP options.

However, more insidious issues come with the ADD_ADDR messages and NATs on the path. Sending IP addresses within the data payload of a NATed connection is always a failure-prone option, and MPTCP is no exception here. MPTCP contains no inbuilt NAT detection functions, and there is no way to determine the direction of the NAT. A host can communicate to the remote end its own IP address or additional available addresses, but if there is a NAT translating the local-host outbound connections, then the actual address will be unavailable for use until the host actually starts a TCP session using this local address as the source.

A simple approach that is effective where NATs are in place is to leave the role of initiation of new subflows to the host that started the connection in the first place. In a client–server environment this solution would imply that the role of setting up new subflows is best left to the client in such cases. However, no such constraints exist when there are no NATs, and in that case either end can initiate new subflows, and the ADD_ADDR messages can keep the other end informed about potential new parallel paths between the two hosts. Logically it makes little sense for MPTCP itself to define a NAT-sensing probe behavior, but it makes a lot of sense for the application using MPTCP to undertake such a test.

The Implications of MPTCP

MPTCP admits considerable flexibility in the way an application can operate when many connection options are available.

All TCP subflows carry the MPTCP option, so that the MPTCP shared state is shared across all active TCP subflows. No single subflow is the "master" in the MCTCP sense. Subflows can be created when interfaces come up, and removed when they go down. Subflows are also IP protocol agnostic: they can use a collection of IPv4 and IPv6 connections simultaneously. Subflows can be used to load-share across multiple network paths, or operate in a primary/backup configuration depending on the application and the flexibility offered in the API in particular implementations of MPTCP.

When applied to mobile devices, this behavior can lead to unexpected results. I always assumed that my device was incapable of "active handoff." Any connections that were initiated across the cellular radio interface had to stay on that interface, and any connections established over the WiFi interface would also stay on that WiFi network. I always understood that active sessions could not be handed off to a different network. Although it was never an explicitly documented feature (or if it was I have never seen it), I had also assumed that when my mobile device was in an area with an active WiFi connection, then the WiFi would take precedence over its *fourth-generation* (4G) connection for all new connections. This assumption matched the factor of typical data tariffs, where the marginal cost of data over 4G is typically somewhere between 10 and 1,000 times higher than the marginal cost of the same data volume over the WiFi connection. But if applications use MPTCP instead of TCP, then how will they balance their network use across the various networks? The way MPTCP is defined it appears that applications simply open subflows on all available local interfaces, and then the fastest network, rather than the cheapest, will take on the greatest volume of traffic.

But, as usual, it can always get more complicated. What if the WiFi network is a corporate service, with NATs, split-horizon *Virtual Private Networks* (VPNs) and various secure servers? If my device starts to perform MPTCP in such contexts, then to what extent are the properties of my WiFi connection preserved in the cellular data connection? Have I exposed new vulnerabilities by doing this? How can a virtual interface, such as a VPN, inform an MPTCP-aware application that other interfaces are not in the same security domain as the VPN interface?

However, it does appear that MPTCP has a role to play in the area of seamless WiFi handoff. With MPTCP is it possible for a mobile handset to enter a WiFi-serviced area and include a WiFi subflow into the existing data transfer without stopping and restarting the data flow? The application may even shut down the cellular radio subflow when the WiFi subflow is active. This functionality is under the control of the application using MPTCP, rather than being under the control of the host operating system of the carrier.

Going Up the Stack

Of course it does not stop at the transport layer and with the use of MPTCP. Customized applications can perform handoffs themselves.

For example, the "mosh" application is an example of a serial form of address agility, where the session state is a shared secret, and the server will accept a reconnection from any client's IP address, as long as the client can demonstrate its knowledge of the shared secret.

Extending the TCP data-transfer model to enlist multiple active TCP sessions at the application level in a load-balancing configuration is also possible, in a manner not all that different from MPTCP.

Of course one could take this further. Rather than use multiple TCP sessions between the same two endpoints, you could instead share the data from the same server across multiple endpoints, and use multiple TCP sessions to these multiple servers. At this point you have something that looks remarkably like the peer-to-peer data-distribution architecture.

Another approach is to format the data stream into "messages" and permit multiple messages to be sent across diverse paths between the two communicating systems. This approach, the *Stream Control Transmission Protocol* (SCTP), is similar to MPTCP in that it can take advantage of multiple addresses to support multiple paths. It combines the message transaction qualities of UDP with the reliable in-sequenced transport services of TCP. The problem of course in today's network is that because it is neither TCP nor UDP, many forms of middleware, including NATs, are often hostile to SCTP and they drop SCTP packets. One additional cost of the escalation of middleware in today's Internet. These days innovation in protocol models is limited by the rather narrow rules applied by network middleware, and the approximate general rule in today's Internet is that it's TCP, UDP, or middleware fodder!

It has been observed numerous times that the abstraction of a network protocol stack is somewhat arbitrary, and it's possible to address exactly the same set of requirements at many different levels in the reference stack. In the work on multipath support in the Internet, we've seen approaches that exploit parallel data streams at the data link layer, at the IP layer, within routing, in the transport layer, and in the application layer. Each has its respective strengths and weaknesses. But what worries me is what happens if you inadvertently encounter a situation where you have all of these approaches active at the same time? Is the outcome one of amazing efficiency, or paralyzing complexity?

For Further Reading

[1] Erik Nordmark and Marcelo Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6," RFC 5533, June 2009.

[2] Janardhan Iyengar, Costin Raiciu, Sebastien Barre, Mark Handley, and Alan Ford, "Architectural Guidelines for Multipath TCP Development," RFC 6182, March 2011.

[3] Mark Handley, Alan Ford, Costin Raiciu, and Olivier Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, January 2013.

[4] Bit Torrent:
https://wiki.theory.org/BitTorrentSpecification

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

[6] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, October 1996.

[7] M. Mathis and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control," Proceedings of SIGCOMM, August 1996.

[8] Randall Stewart, "Stream Control Transmission Protocol," RFC 4960, September 2007.

[9] Ethan Blanton and Mark Allman, "TCP Congestion Control," RFC 5681, September 2009.

[10] Olivier Bonaventure, "Apple seems to also believe in Multipath TCP," Blog post,
http://perso.uclouvain.be/olivier.bonaventure/blog/html/2013/09/18/mptcp.html

[11] Jonathan B. Postel, "Transmission Control Protocol," RFC 793, September 1981.

[12] Geoff Huston, "TCP Performance," *The Internet Protocol Journal*, Volume 3, No. 2, June 2000.

[13] Geoff Huston, "The Future for TCP," *The Internet Protocol Journal*, Volume 3, No. 3, September 2000.

[14] Wesley M. Eddy, "Defenses Against TCP SYN Flooding Attacks," *The Internet Protocol Journal*, Volume 9, No. 4, December 2006.

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**

TCP Protocol Wars

by Geoff Huston, APNIC

There are two end-to-end transport protocols in common use in today's Internet: the *User Datagram Protocol* (UDP) and the *Transmission Control Protocol* (TCP).

UDP is an abstraction of the basic IP datagram, in that UDP is an unreliable medium. Packets sent using UDP may or may not go to their intended destination. UDP packets may be reordered, duplicated, or lost. UDP has no flow control or throttling. The packet quantization in UDP is explicit: if the sender splits data into two UDP packets, then the receiver will collect the data using two distinct read operations.

TCP is a reliable end-to-end flow-controlled stream protocol. A stream of data passed into a TCP socket at one end will be read as a stream of data at the other end. The packet quantization is hidden from the application, as are the mechanics of flow control, loss detection, and retransmission and session establishment and teardown. TCP will not preserve any inherent timing within the data stream, but will preserve the integrity of the stream.

Critically, the Internet assumes that most of the network resources are devoted to passing TCP traffic, and it also assumes that the flow-control algorithms used by these TCP sessions all behave in approximately similar ways. If the switching and transmission resources of the network are seen as a common resource, then the assumption about the uniform behavior of TCP sessions implies that these end-to-end transport sessions will behave similarly under contention. The result is that, to a reasonable level of approximation, a set of concurrent TCP sessions will self-equilibrate to give each TCP session an equal share of the common resource. In other words, the network itself does not have to impose "fairness" on the TCP flows that pass across it—as long as all the flows are controlled by a uniform flow-control algorithm, the flows will interact with each other in a manner that is likely to allocate an equal proportion of the network resources to each active TCP flow. At least that's the theory.

This theory raises numerous questions of whether these assumptions are true in today's Internet and what may be changing with these assumptions.

Other Protocols?

Is it still a choice between UDP and TCP? Despite many technical efforts to specify new end-to-end transport protocols, there is little chance that any new protocol will gain acceptance in today's Internet. The network contains large numbers of intercepting "middleware," and these units function as security firewalls by using rules that are very limited in the protocols that they admit.

The most common filters in middleware are configured to admit only IP protocols 6 and 17 (TCP and UDP, respectively), and drop all others. This setup has implied that more recent end-to-end transport protocols, such as the *Stream Control Transmission Protocol* (SCTP)[10] or the *Datagram Congestion Control Protocol* (DCCP)[11], for example, have very limited applicability in the public Internet, because they can be used only in environments where there is no such intercepting middleware.

TCP or UDP?

In this world where choice is limited to TCP or UDP, the conventional view was that the bulk of the traffic was carried in TCP, whereas UDP was used in limited contexts for *Domain Name System* (DNS) name resolution, running time, and network management. This view raises the question as to whether TCP still carries the bulk of the Internet traffic load.

It is not necessarily true that TCP still carries the bulk of the Internet traffic load, although reliable data sources that provide visibility into the actual traffic profile seen on end user-facing networks is not easily forthcoming. A recent study of traffic profiles between 2002 and 2009 by the *Center for Applied Internet Data Analysis* (CAIDA)[1] points to a UDP/TCP ratio value of 0.11 when looking at the volume of data being transported by the two protocols. In other words, some 90% of the traffic was carried inside TCP sessions and 10% inside UDP sessions. For UDP this value is considerably higher than would be conventionally expected from the combination of only DNS and *Network Time Protocol* (NTP) payloads in UDP. The study points out: "A port-based analysis suggests that the recent increase in UDP flows on the traces analyzed stems mainly from *Peer-to-Peer* (P2P) applications using UDP for their overlay signalling traffic," a result that corresponds to reports of the use of the *Low Extra Delay Background Transport* (LEDBAT) protocol for *BitTorrent*[2]. More recently, video streaming applications have also turned to TCP, using local buffer management in the playback device to overcome TCP-induced signal jitter.

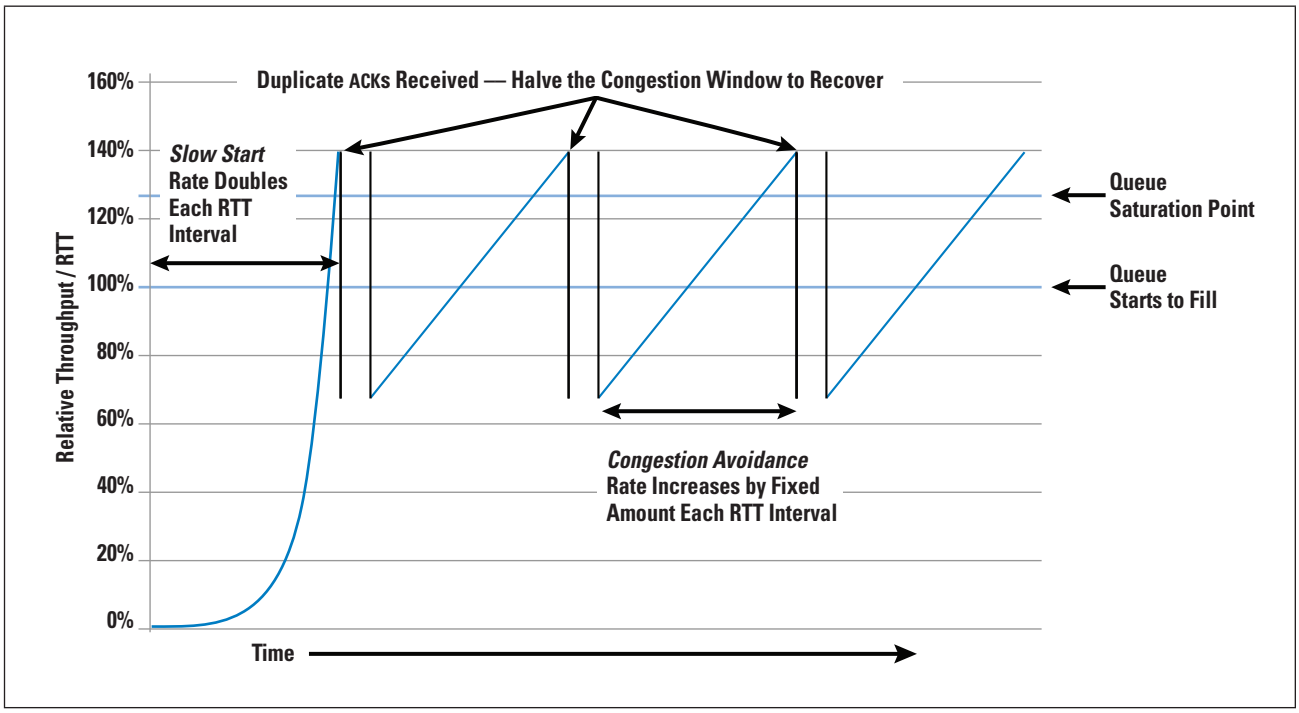
It is reasonable to assume that the overall majority of the Internet traffic load is carried in TCP, and therefore the behavior of the TCP flow-control algorithm is a matter of interest.

TCP Flow Control – TCP Reno

TCP does not have a single flow-control algorithm. Although the common TCP protocol specification defines how to establish and shut down a session, and defines the way in which received data is acknowledged back to the sender, the core protocol specification does not specify how the two ends negotiate the speed at which data is passed between them. This negotiation has been left to the various implementations of the TCP flow-control algorithm.

"Conventional" flow control in TCP is typified by the behavior of the TCP *Reno* algorithm (Figure 1).

Figure 1: Idealised TCP Reno Flow Control



There are two distinct phases of behavior: the *Slow Start* phase, where the sending rate is doubled every *Round-Trip Time* (RTT) interval, and a *Congestion Avoidance* phase, where the sending rate is increased by a fixed amount—one *Message Segment Size* (MSS)—in each RTT interval. When the sender is notified of packet loss—by receiving a duplicate *Acknowledgment* (ACK) message from the receiver—the actions of the sender vary according to its current phase. In Slow Start phase a duplicate ACK will shift the sender to Congestion Avoidance mode. In Congestion Avoidance mode a duplicate ACK will cause the sender to halve its sending rate and continue in this mode. Three duplicate ACKs in succession will cause the session to restart from scratch in Slow Start mode, because three duplicate ACKs signals a higher rate of congestion which means that the two ends of the TCP stream have lost their shared flow state assumption.

In steady state the TCP Reno flow-control algorithm increases the flow rate by a constant amount each round-trip time interval, and when a packet is dropped, because of buffer overflow in a switch, the algorithm halves the flow rate. The result is an *Additive Increase Multiplicative Decrease* (AIMD) algorithm, which tends to place high levels of pressure on the buffers in the network while there is still available buffer space, and react dramatically when the buffers eventually overfill and reach the packet drop point. Crudely, this process is a "boom and bust" form of feedback control.

Better than Reno

There have been strong motivations by application families to break out of this form of TCP flow-control behavior. One motivation is to use a more even packet flow across the network, and remove some of the "jerkiness" inherent in TCP Reno. There is also the motivation that a more sensitive flow-control application could achieve a superior outcome compared to TCP Reno. In other words, a different TCP flow-control algorithm could achieve better than its "fair share" when competing against a set of concurrent TCP Reno flows.

The first of these motivations is a simple change. In an attempt to double the pressure on other concurrent TCP sessions, the AIMD algorithm can be adjusted by increasing the sending speed a larger constant amount, and decreasing it by less following packet loss (*MulTCP* uses this model). For example, if the speed was increased by 2 MSS units each RTT interval and the sending rate was reduced by one-quarter rather than one-half upon receipt of a duplicate ACK, then the resultant behavior would, in an approximate sense, behave like two concurrent TCP sessions, and in a fair sharing scenario this form of flow control would attempt to secure double the network resources of an equivalent TCP Reno session.

Another variant of this approach is *Highspeed TCP* which increases its frequency of probing into potentially claimable capacity by increasing its sending rate by a larger volume while keeping its reduction rate at a constant value. This protocol probes for the packet-loss onset at a far higher frequency than either TCP Reno or MulTCP, and is capable of accelerating to much higher flow speeds in a much shorter time interval.

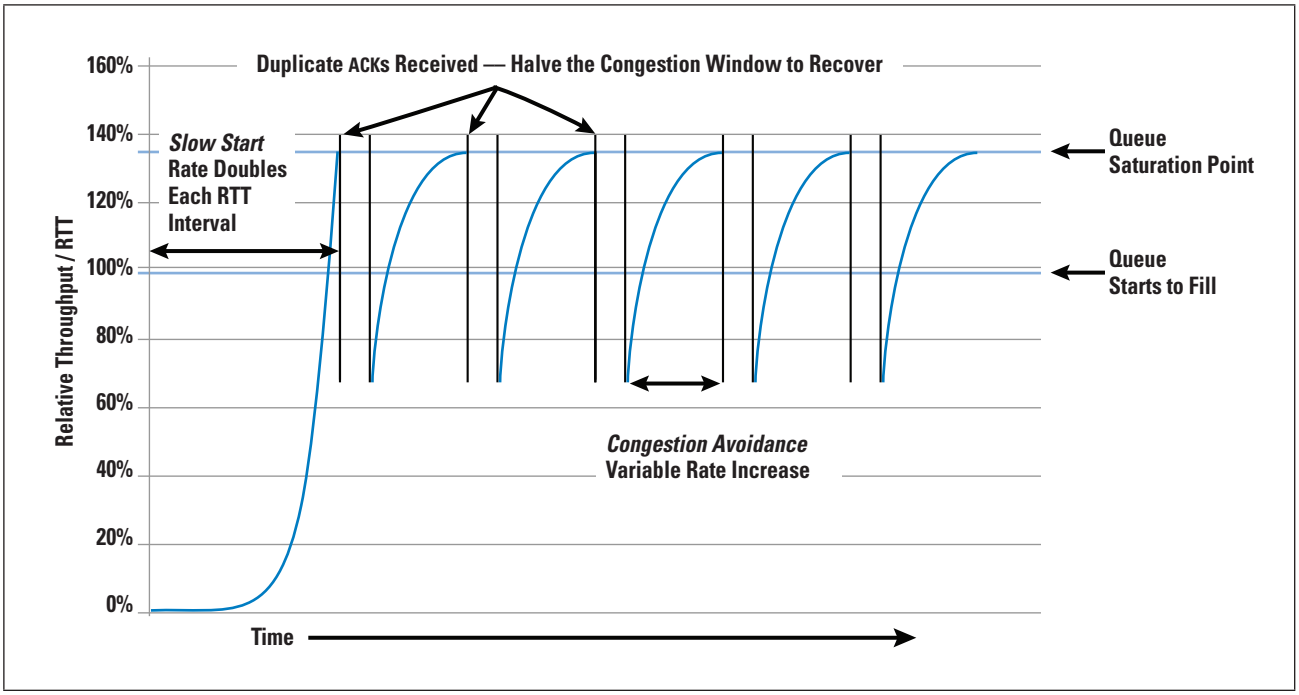
Binary Increase Congestion Control (BIC) and its variant CUBIC use a nonlinear increase function rather than a constant rate increase function (Figure 2). Instead of increasing the speed by a fixed amount each RTT in Congestion Avoidance mode, BIC remembers the sending rate at the onset of packet drop, and each RTT interval increases its speed by one-half of the difference between the current sending rate and the assumed bottleneck rate.

BIC quickly drives the session towards the bottleneck capacity, and then probes more cautiously when the sending speed is close to the bottleneck capacity. Again, compared to a Reno flow session, CUBIC should produce a superior outcome.

Other flow-control algorithms move away from using packet loss as the control indication and tend to oscillate more frequently around the point of the onset of queuing in the routers in the network path. This form of feedback control is sensitive to the relative time differences between sent packets and received ACKs.

An example is "packet-pair" flow-controlled TCP, where the sending rate is increased as long as the time interval between two packets being sent is equal to the time interval of the received ACKs. If the ACK interval becomes larger, then this increase is interpreted as the onset of queuing in the sending path, and the sending rate is decreased until the ACK timing interval once again equals the send timing interval.

Figure 2: Idealized TCP BIC Flow Control



Recent Microsoft systems use *Compound TCP*, which combines TCP Reno and delay-based flow control. The algorithm attempts to measure the amount of in-flight data held in queues (higher delay traffic), and upon packet loss the algorithm reduces its sending rate to below the onset of growth in queuing.

Apple's Macintosh systems use *New Reno*, a variant of the Reno flow-control algorithm that improves the Reno loss recovery procedure, but is otherwise the same AIMD control algorithm. Linux kernels have switched to use CUBIC, a variant of the BIC algorithm that uses a cubic function rather than an exponential function to govern window inflation.

Crossing the Beams: TCP implemented in UDP

Other approaches have headed further away from conventional TCP and change both the server and the client. One way to change both the server and the client is to avoid the use of the operating system-provided implementation of TCP completely, place a TCP-styled reliable flow-control streaming protocol into the application itself, and use the UDP interface of the operating system to pass packets to and from the network.

This approach has been used in the widely deployed *BitTorrent* application (LEDBAT), and more recently by Google in its experiments with *Quick UDP Internet Connections* (QUIC)[8] and *SPDY* (pronounced "speedy").

Google's QUIC uses a TCP emulation in UDP that has a data encoding that includes *Forward Error Correcting Codes* (FEC) as a way of performing a limited amount of repair of the data stream in the face of packet loss without retransmission. QUIC performs bandwidth estimation as a means of rapidly reaching an efficient sending rate. SPDY further assists QUIC by multiplexing application sessions within a single end-to-end transport protocol session. This approach avoids the startup overhead of each TCP session, and leverages the observation that TCP takes some time to establish the bottleneck capacity of the network. The use of UDP also avoids intercepting middleware that performs deep packet inspection on TCP flows and modifies their advertised window size to perform external moderation on TCP flow rate.

There is, however, one issue with the use of UDP as a substitute for TCP, and although public reports from Google on this topic have not been published, it is a source of concern. The problem relates to the use of UDP through *Network Address Translators* (NATs)[12] and the issue of address binding times within the NAT. In TCP a NAT takes its directions from TCP. When the NAT sees an opening TCP handshake packet from the "inside," it creates a temporary address binding and sends the packet to its intended destination (with the translated source address of course). The reception of the response part of the handshake at the NAT causes the NAT to confirm its binding entry and apply it to subsequent packets in this TCP flow. The NAT holds state until it sees a closing exchange or a reset signal that closes the TCP session, or until an idle timer expires. For TCP the NAT is attempting to hold the binding for as long as the TCP session is active. For NATs, UDP is different. Unlike TCP, there is no flow-status information in UDP. So when the NAT creates a UDP binding, it has to hold it for a certain amount of time. There is no clear technical standard here, so implementations vary. Some NATs use very short timers and release the binding quickly, matching the expectation of the use of UDP as a simple query/response protocol. The use of UDP as an ersatz packet-framing protocol for user-level TCP implementation requires the NAT to hold the UDP address binding for longer intervals, corresponding to the hidden TCP session. Some NATs will do so, while others will destroy the binding even though there are still UDP packets active, thus disturbing the hidden TCP session.

This example illustrates the level of compromise in today's environment between end-to-end protocols and network middleware. TCP sessions are being modified by active middleware that attempts to govern the TCP flow rate by active modification of window sizes within the TCP session, negating some of the efforts of the TCP session to optimize its flow speed.

TCP in UDP passes control of the TCP flow management to the application, and hides the TCP flow parameters from the network. However, UDP sessions are susceptible to interruption by NAT intervention, because some NATs assume that UDP is used only for micro-sessions, and long-held UDP sessions are some form of anomalous behavior that should be filtered by removing the UDP port binding in the NAT.

The Transport Protocol Ecosystem

The Internet is somewhat unique in so far as there is no intrinsic network-level functionality that can allocate a certain amount of network resources to each active flow being carried across the network. The network is not actively "managed." Network resources are allocated to traffic flows in a manner similar to fluid-flow equilibrium. Each active flow exerts pressure on all other concurrent flows. The higher the relative imbalance, the more the largest flows are pressured to reduce their flow rate by the smaller flows. The system reaches a meta-equilibrium point when all concurrent flows receive approximately equal amounts of network resource.

The underlying assumption here is that a fair result is achieved if all the concurrent flows are operating in a similar manner. What is happening in the network today is a fragmentation of the TCP flow-control algorithm as operating systems, and even applications, prefer to use a customized flow-control algorithm that attempts to optimize their position by exerting slightly more pressure on other TCP sessions, causing them to drop their flow rates in response. These techniques do not create additional network transmission capacity, they bias the way in which network capacity is available to individual traffic flows in their favor. So if a TCP session is able to secure better than its "fair share" of a laden network, then other sessions are necessarily affected and receive less than their "fair share."

There is some relationship between these protocol-level efforts and the *Net Neutrality* policy debates. The proponents of a Net Neutrality position argue that the network should be a largely passive entity, and that the interaction of the various traffic flows produces a fair and efficient outcome. The network resources will be fully allocated to carrying traffic with relatively small levels of retransmission (efficiency), and the concurrent flows will interact with each other to produce an outcome where each flow gathers approximately equal network resource (fair). With the increasing level of diversity in approaches to packet-flow management, and the options of whether to use the flow-control services provided by the operating system platform or go the path of using UDP as the transport protocol and passing the flow-control algorithm to the application, what is being witnessed is some amount of escalation in competitive pressure between applications to secure network resources.

For Further Reading

[0] Geoff Huston, "IP Multi-Addressing and Multipath TCP," *The Internet Protocol Journal,* Volume 18, No. 2, June 2015.

[1] CAIDA Traffic Analysis,
http://www.caida.org/research/traffic-analysis/tcpudpratio/

[2] **http://en.wikipedia.org/wiki/LEDBAT**

[3] Van Jacobson and Mike Karels, "Congestion Avoidance and Control," 1988, **http://ee.lbl.gov/papers/congavoid.pdf**

[4] W. Richard Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC 2001, January 1997.

[5] Ethan Blanton and Mark Allman, "TCP Congestion Control," RFC 5681, September 2009.

[6] Peter Dodcal, "15 Newer TCP Implementations,"
http://intronetworks.cs.luc.edu/current/html/newtcps.html

[7] FAST: **https://en.wikipedia.org/wiki/FAST_TCP**

[8] Google's QUIC:
https://www.chromium.org/quic
http://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.htm

[9] IETF activity on TCP flow control: TCP Maintenance and Minor Extensions (tcpm)
https://datatracker.ietf.org/wg/tcpm/charter/

[10] Randall Stewart, "Stream Control Transmission Protocol," RFC 4960, September 2007.

[11] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340, March 2006.

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

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**

Fragments

IAB Statement on the Trade in Security Technologies

The *Internet Architecture Board* (IAB) published the following statement on June 15, 2015:

"The Internet Architecture Board is deeply sympathetic with the desire to enhance the security of Internet protocols, infrastructure, and Internet-connected systems. We believe, however, that efforts to enhance Internet security must proceed from a thorough knowledge of the threats against the network, its protocols, and the systems attached to it. Efforts to limit the export or transfer of Internet security technologies seem likely to limit that knowledge in ways that ultimately will frustrate the general goal of a secure and stable Internet.

The identification of vulnerabilities is a fundamental part of security practice. Restrictions on systems which perform that function will make it substantially more difficult for those performing that function to design and deploy secure systems.

Traffic analysis systems, though they may be used in other ways, are a similarly crucial part of the methods used to identify attacks and to analyze the success of remediations put in place. The Internet is a deeply interconnected set of networks that spans international borders, and attacks may occur in one part of the Internet that have extensive ramifications for the operation of the whole. Limiting traffic analysis technologies to specific territories seems likely to hinder efforts to detect and thwart both active threats and other network issues.

We note that in 1996 the IAB and *Internet Engineering Steering Group* (IESG) jointly published RFC 1984[1], with the following comments on a similar matter, the export of encryption technology:

Export controls on encryption place companies in that country at a competitive disadvantage. Their competitors from countries without export restrictions can sell systems whose only design constraint is being secure, and easy to use.

Usage controls on encryption will also place companies in that country at a competitive disadvantage because these companies cannot securely and easily engage in electronic commerce.

Export controls and usage controls are slowing the deployment of security at the same time as the Internet is exponentially increasing in size and attackers are increasing in sophistication. This puts users in a dangerous position as they are forced to rely on insecure electronic communication.

We believe the same points to be fundamentally true for the export of traffic analysis, penetration testing, and similar security technologies.

While it may appear possible to narrowly circumscribe restrictions so that they target technologies that serve no possible purpose but attack, any modular system, including those intended solely for research, will like have some elements that, divorced from the system, would serve no other purpose. Efforts to target such systems will thus likely sweep up many other security technologies. We therefore recommend that export restrictions on security technologies be generally avoided."

[1] IAB and IESG, "IAB and IESG Statement on Cryptographic Technology and the Internet," RFC 1984, August 1996.

A Primer on IPv4 Scarcity

The April 2015 Issue of the ACM SIGCOMM *Computer Communication Review* contained an excellent summary of the rise and fall of the IPv4 address space[1]. The authors have managed to be wonderfully concise, packing into just a little over 8 pages a history of the initial address allocation practices, the evolution of needs-based address provisioning through the *Regional Internet Registry* (RIR) framework, and the onset of depletion and exhaustion in the last five years. The paper also reviews the routed address space, and explains the differences between *occupied, routed*, and *allocated* address space. It also explains the concept of efficiency of utilization of addresses. The authors consider IPv4 addresses as a resource and the long standing debate over whether addresses can be considered as conventional "property," as well as the tension between the policies of the various registries and the perspectives of the holders of address space. The paper outlines recent efforts to augment the registry functions with a form of certification allowing third parties to use a *Public Key Infrastructure* (PKI) to validate the authenticity of attestations about addresses and their use, particularly in the context of the Internet's routing system. The paper details current efforts in coping with an environment where the traditional source of IPv4 addresses has been exhausted, considers address *markets,* and the interplay between efforts to increase the address utilization efficiency in IPv4 and incentives to adopt IPv6.

[1] Philipp Richter, Mark Allman, Randy Bush, Vern Paxson, "A Primer on IPv4 Scarcity," ACM SIGCOMM *Computer Communication Review,* Volume 45, Number 2, April 2015.
`http://www.sigcomm.org/sites/default/files/ccr/papers/2015/April/0000000-0000002.pdf`

Corrections

While we are all looking forward to *Terabit* (1000G) Ethernet, the article in IPJ Volume 18, No.1 entitled "Gigabit Ethernet," contained errors in the table on page 27. Thanks to reader Marcin Cieślak for pointing this out. Here is the corrected version:

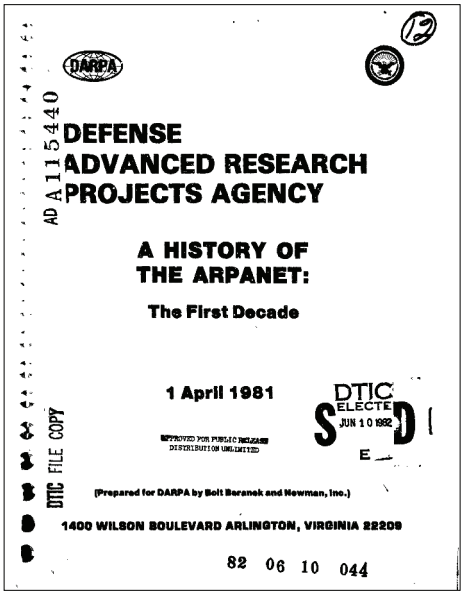
Table 2: Media Options for 40- and 100-Gbps Ethernet

	40 Gbps	100 Gbps
1-m backplane	40GBASE-KR4	
10-m copper	40GBASE-CR4	100GBASE-CR10
100-m multimode fiber	40GBASE-SR4	100GBASE-SR10
10-km single-mode fiber	40GBASE-LR4	100GBASE-LR4
40-km single-mode fiber		100GBASE-ER4

Naming nomenclature:

Copper: K = Backplane; C = Cable assembly
Optical: S = Short reach (100 m); L - Long reach (10 km);
E = Extended long reach (40 km)
Coding scheme: R = 64B/66B block coding
Final number: Number of lanes (copper wires or fiber wavelengths)

Also in Volume 18, No. 1, we told you about Bolt Beranek and Newman Report 4799 entitled "A History of the ARPANET: The First Decade." It appears that this document is no longer available from the link we gave, so we have placed a copy in the "Downloads" section of our website at **protocoljournal.org**.



Call for Papers

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

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

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

Supporters and Sponsors

Publication of this journal is made possible by:

Supporters

Diamond Sponsors

Ruby Sponsor

Sapphire Sponsors

Emerald Sponsors

Corporate Subscriptions

Individual Sponsors

Lyman Chapin, Steve Corbató, Dave Crocker, Jay Etchings, Martin Hannigan, Hagen Hultzsch, Dennis Jennings, Jim Johnston, Merike Kaeo, Bobby Krupczak, Richard Lamb, Tracy LaQuey Parker, Bill Manning, Andrea Montefusco, Tariq Mustafa, Mike O'Connor, Tim Pozar, George Sadowsky, Helge Skrivervik, Rob Thomas, Tom Vest, Rick Wesson.

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

Printed in the USA on recycled paper.