

The Internet Protocol Journal

December 2006

Volume 9, Number 4

A Quarterly Technical Publication for
Internet and Intranet Professionals

FROM THE EDITOR

In This Issue

From the Editor	1
SYN Flooding Attacks	2
XML Networking	17
Letters to the Editor	33
Book Review	37
Fragments	40
Call for Papers	43

Internet security and stability are topics we keep returning to in this journal. So far we have mainly focused on technologies that protect systems from unauthorized access and ensure that data in transit over wired or wireless networks cannot be intercepted. We have discussed security-enhanced versions of many of the Internet core protocols, including the *Border Gateway Protocol* (BGP), *Simple Network Management Protocol* (SNMP), and the *Domain Name System* (DNS). You can find all these articles by visiting our Website and referring to our index files. All back issues continue to be available in both HTML and PDF formats. In this issue, Wesley Eddy explains a vulnerability in the *Transmission Control Protocol* (TCP) in which a sender can overwhelm a receiver by sending a large number of SYN protocol exchanges. This form of *Denial of Service* attack, known as *SYN Flooding*, was first reported in 1996, and researchers have developed several solutions to combat the problem.

Speaking of Internet stability, at 12:26 GMT on December 26, 2006, an earthquake of magnitude 6.7 struck off Taiwan's southern coast. Six submarine cables were damaged, resulting in widespread disruption of Internet service in parts of Asia. We hope to bring you more details and analysis of this event in a future issue of IPJ. The topic will also be discussed at the next *Asia Pacific Regional Internet Conference on Operational Technologies* (APRICOT), which will take place in Bali, Indonesia, February 21 through March 2, 2007. For details see: <http://www.apricot2007.net>

The design and operation of systems that use Internet protocols for communication in conjunction with advanced applications—such as an e-commerce system—require the use of a certain amount of “middleware.” This software, largely hidden from the end user, has been the subject of a great deal of development and standardization work for several decades. An important component of today's Web systems is the *Extensible Markup Language* (XML). Silvano Da Ros explains how XML networking can be used as a critical building block for network application interoperability.

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

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

Defenses Against TCP SYN Flooding Attacks

by Wesley M. Eddy, Verizon Federal Network Systems

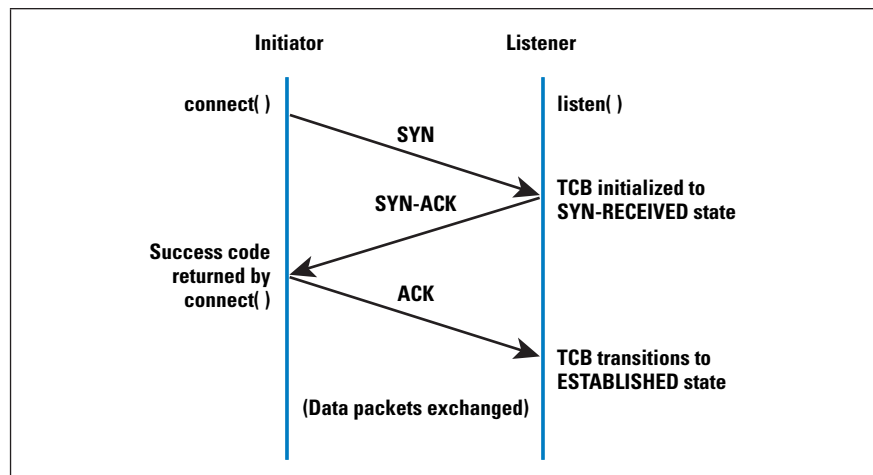
This article discusses a specific *Denial of Service* (DoS) attack known as *TCP SYN Flooding*. The attack exploits an implementation characteristic of the *Transmission Control Protocol* (TCP), and can be used to make server processes incapable of answering a legitimate client application's requests for new TCP connections. Any service that binds to and listens on a TCP socket is potentially vulnerable to TCP SYN flooding attacks. Because this includes popular server applications for e-mail, Web, and file storage services, understanding and knowing how to protect against these attacks is a critical part of practical network engineering.

The attack has been well-known for a decade, and variations of it are still seen. Although effective techniques exist to combat SYN flooding, no single standard remedy for TCP implementations has emerged. Varied solutions can be found among current operating systems and equipment, with differing implications for both the applications and networks under defense. This article describes the attack and why it works, and follows with an overview and assessment of the current tactics that are used in both end hosts and network devices to combat SYN flooding attacks.

Basic Vulnerability

The SYN flooding attack became well-known in 1996, when the magazines *2600* and *Phrack* published descriptions of the attack along with source code to perform it^[1]. This information was quickly used in attacks on an Internet service provider's (ISP's) mail and Telnet servers, causing outages that were widely publicized in *The Washington Post* and *The Wall Street Journal* (among other venues). CERT quickly released an advisory on the attack technique^[2].

Figure 1: Normal TCP 3-Way Handshake



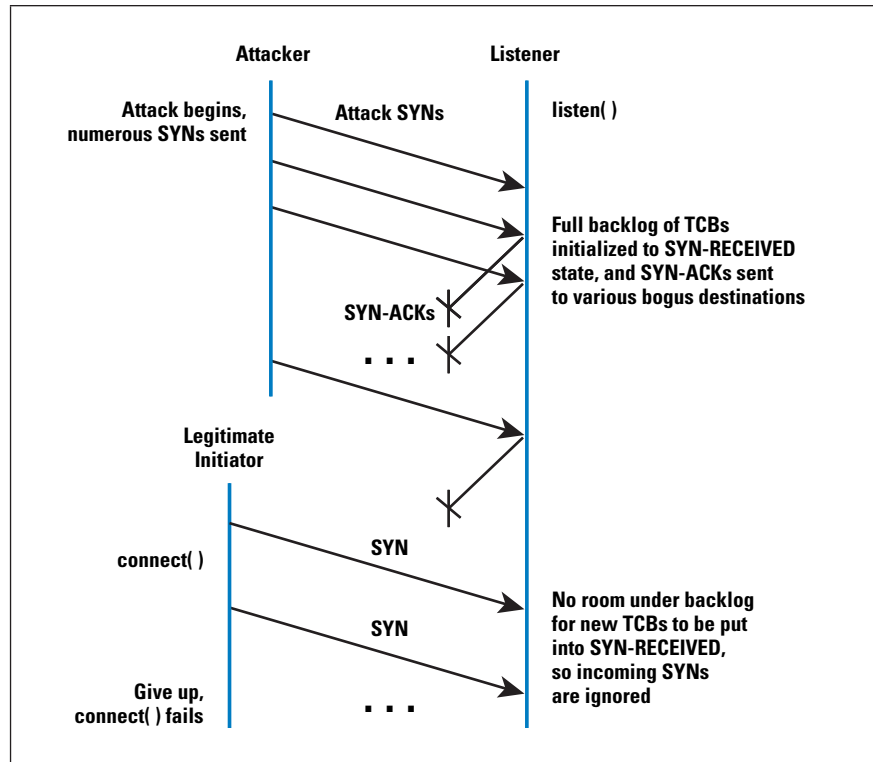
The basis of the SYN flooding attack lies in the design of the 3-way handshake that begins a TCP connection. In this handshake, the third packet verifies the initiator's ability to receive packets at the IP address it used as the source in its initial request, or its return reachability. Figure 1 shows the sequence of packets exchanged at the beginning of a normal TCP connection (refer to RFC 793 for a detailed description of this process).

The *Transmission Control Block* (TCB) is a transport protocol data structure (actually a set of structures in many operations systems) that holds all the information about a connection. The memory footprint of a single TCB depends on what TCP options and other features an implementation provides and has enabled for a connection. Usually, each TCB exceeds at least 280 bytes, and in some operating systems currently takes more than 1300 bytes. The TCP SYN-RECEIVED state is used to indicate that the connection is only half open, and that the legitimacy of the request is still in question. The important aspect to note is that the TCB is allocated based on reception of the SYN packet—before the connection is fully established or the initiator's return reachability has been verified.

This situation leads to a clear potential DoS attack where incoming SYNs cause the allocation of so many TCBs that a host's kernel memory is exhausted. In order to avoid this memory exhaustion, operating systems generally associate a "backlog" parameter with a listening socket that sets a cap on the number of TCBs simultaneously in the SYN-RECEIVED state. Although this action protects a host's available memory resource from attack, the backlog *itself* represents another (smaller) resource vulnerable to attack. With no room left in the backlog, it is impossible to service new connection requests until some TCBs can be reaped or otherwise removed from the SYN-RECEIVED state.

Depleting the backlog is the goal of the TCP SYN flooding attack, which attempts to send enough SYN segments to fill the entire backlog. The attacker uses source IP addresses in the SYNs that are not likely to trigger any response that would free the TCBs from the SYN-RECEIVED state. Because TCP attempts to be reliable, the target host keeps its TCBs stuck in SYN-RECEIVED for a relatively long time before giving up on the half connection and reaping them. In the meantime, service is denied to the application process on the listener for legitimate new TCP connection initiation requests. Figure 2 presents a simplification of the sequence of events involved in a TCP SYN flooding attack.

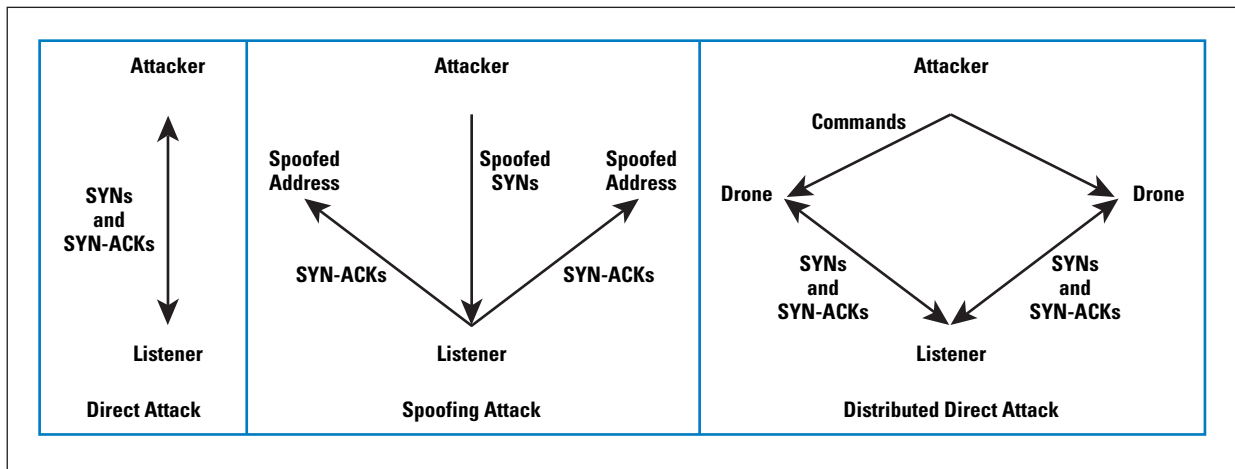
Figure 2: Attack Demonstration: Enough illegitimate TCBS are in SYN-RECEIVED that a legitimate connection cannot be initiated.



Attack Methods

The scenario pictured in Figure 2 is a simplification of how SYN flooding attacks are carried out in the real world, and is intended only to give an understanding of the basic idea behind these types of attacks. Figure 3 presents some variations that have been observed on the Internet.

Figure 3: Some Variants of the Basic Attack



Direct Attack

If attackers rapidly send SYN segments without spoofing their IP source address, we call this a *direct attack*. This method of attack is very easy to perform because it does not involve directly injecting or spoofing packets below the user level of the attacker's operating system. It can be performed by simply using many TCP *connect()* calls, for instance. To be effective, however, attackers must prevent their operating system from responding to the SYN-ACKS in any way, because any ACKs, RSTs, or *Internet Control Message Protocol* (ICMP) messages will allow the listener to move the TCB out of SYN-RECEIVED. This scenario can be accomplished through firewall rules that either filter outgoing packets to the listener (allowing only SYNs out), or filter incoming packets so that any SYN-ACKS are discarded before reaching the local TCP processing code.

When detected, this type of attack is very easy to defend against, because a simple firewall rule to block packets with the attacker's source IP address is all that is needed. This defense behavior can be automated, and such functions are available in off-the-shelf reactive firewalls.

Spoofing-Based Attacks

Another form of SYN flooding attacks uses IP address spoofing, which might be considered more complex than the method used in a direct attack, in that instead of merely manipulating local firewall rules, the attacker also needs to be able to form and inject raw IP packets with valid IP and TCP headers. Today, popular libraries exist to aid with raw packet formation and injection, so attacks based on spoofing are actually fairly easy.

For spoofing attacks, a primary consideration is address selection. If the attack is to succeed, the machines at the spoofed source addresses must not respond to the SYN-ACKS that are sent to them in any way. A very simple attacker might spoof only a single source address that it knows will not respond to the SYN-ACKS, either because no machine physically exists at the address presently, or because of some other property of the address or network configuration. Another option is to spoof many different source addresses, under the assumption that some percentage of the spoofed addresses will be unresponsive to the SYN-ACKS. This option is accomplished either by cycling through a list of source addresses that are known to be desirable for the purpose, or by generating addresses inside a subnet with similar properties.

If only a single source address is repetitively spoofed, this address is easy for the listener to detect and filter. In most cases a larger list of source addresses is used to make defense more difficult. In this case, the best defense is to block the spoofed packets as close to their source as possible.

Assuming the attacker is based in a “stub” location in the network (rather than within a transit *Autonomous System* (AS), for instance), restrictive network ingress filtering^[7] by stub ISPs and egress filtering within the attacker’s network will shut down spoofing attacks—if these mechanisms can be deployed in the right places. Because these ingress/egress filtering defenses may interfere with some legitimate traffic, such as the Mobile IP triangle routing mode of operation, they might be seen as undesirable, and are not universally deployed. *IP Security* (IPsec) also provides an excellent defense against spoofed packets, but this protocol generally cannot be required because its deployment is currently limited. Because it is usually impossible for the listener to ask the initiator’s ISPs to perform address filtering or to ask the initiator to use IPsec, defending against spoofing attacks that use multiple addresses requires more complex solutions that are discussed later in this article.

Distributed Attacks

The real limitation of single-attacker spoofing-based attacks is that if the packets can somehow be traced back to their true source, the attacker can be easily shut down. Although the tracing process typically involves some amount of time and coordination between ISPs, it is not impossible. A distributed version of the SYN flooding attack, in which the attacker takes advantage of numerous drone machines throughout the Internet, is much more difficult to stop. In the case shown in Figure 3, the drones use direct attacks, but to increase the effectiveness even further, each drone could use a spoofing attack and multiple spoofed addresses.

Currently, distributed attacks are feasible because there are several “botnets” or “drone armies” of thousands of compromised machines that are used by criminals for DoS attacks. Because drone machines are constantly added or removed from the armies and can change their IP addresses or connectivity, it is quite challenging to block these attacks.

Attack Parameters

Regardless of the method of attack, SYN flooding can be tuned to use fewer packets than a brute-force DoS attack that simply clogs the target network by sending a high volume of packets. This tuning is accomplished with some knowledge of the listener’s operating system, such as the size of the backlog that is used, and how long it keeps TCBS in SYN-RECEIVED before timing out and reaping them. For instance, the attacker can minimally send a quick flight of some number of SYNs exactly equal to the backlog, and repeat this process periodically as TCBS are reclaimed in order to keep a listener unavailable perpetually.

Default backlogs of 1024 are configured on some recent operating systems, but many machines on the Internet are configured with backlogs of 128 or fewer. A common threshold for retransmission of the SYN-ACK is 5, with the timeout between successive attempts doubled, and an initial timeout of 3 seconds, yielding 189 seconds between the time when the first SYN-ACK is sent and the time when the TCB can be reclaimed.

Assuming a backlog of 128 and that an attacker generates 40-byte SYN segments (with a 20-byte TCP header plus a 20-byte IP header), the attacker has to send only 5.12 kilobytes (at the IP layer) in order to fill the backlog. Repeated every 189 seconds, this process gives an average data rate of only 27 bytes per second (easily achievable even over dialup links). This data rate is in stark contrast to DoS attacks that rely on sending many megabits per second of attack traffic. Even if a backlog of 2048 is used, the required data rate is only 433 bytes per second, so it is clear that the ease of attack scales along with increases to the backlog—and more sophisticated defenses are needed.

Lessons Learned

The protocol flaw in TCP that makes SYN flooding effective is that for the small cost of sending a packet, an initiator causes a relatively greater expense to the listener by forcing the listener to reserve state in a TCB. An excellent technique for designing protocols that are robust to this type of attack is to make the listener side operate statelessly^[3] until the initiator can demonstrate its legitimacy. This principle has been used in more recent transport protocols, such as the *Stream Control Transmission Protocol (SCTP)*^[4], which has a 4-way handshake, with listener TCB state being created only after the initiator echoes back some “cookie” bytes sent to it by the listener. This echo proves to some extent that the initiator side is at the address it appears to be (that is, it has return reachability) and is not attempting a SYN flooding style of attack.

Outside of transport protocols and TCBS, security protocols also commonly use this defense technique. For instance, the *Internet Key Exchange Version 2 (IKEv2)*^[5] component of IPsec does not create state for a new *Security Association* until it can verify that initiators are capable of responding to packets sent to the address they claims to be using. There are other security protocols in which the listener sends out “puzzles” in response to initiation attempts and grants services or state only when puzzle solutions are returned^[6]. This tactic not only verifies the addresses of initiators but also implies a computational burden that causes them to further demonstrate their genuine willingness to communicate productively.

Countermeasures

During the initial Panix attack, random spoofed source addresses were being used, but it was noted that the attack TCP SYNs all used the same source port number. A filter that denied incoming packets from this port was temporarily effective, but easy for the attacker to adapt to, and the attack segments began using random ports. Panix was able to isolate which of its ingress routers the attack was coming from and null-route packets destined for its servers coming through that router, but this solution was obviously a heavy-handed one, and seems to have also been overcome when the attacker started sending packets that were routed through a different upstream provider. Panix had mixed success in getting its providers to assist in tracing and blocking the attack, and the networking community was spurred into devising other solutions.

Two broad classes of solutions to SYN flooding attacks have evolved, corresponding to where the defenses are implemented. The first class of solutions involves hardening the end-host TCP implementation itself, including altering the algorithms and data structures used for connection lookup and establishment, as well as some solutions that diverge from the TCP state machine behavior during connection establishment, as described in RFC 793.

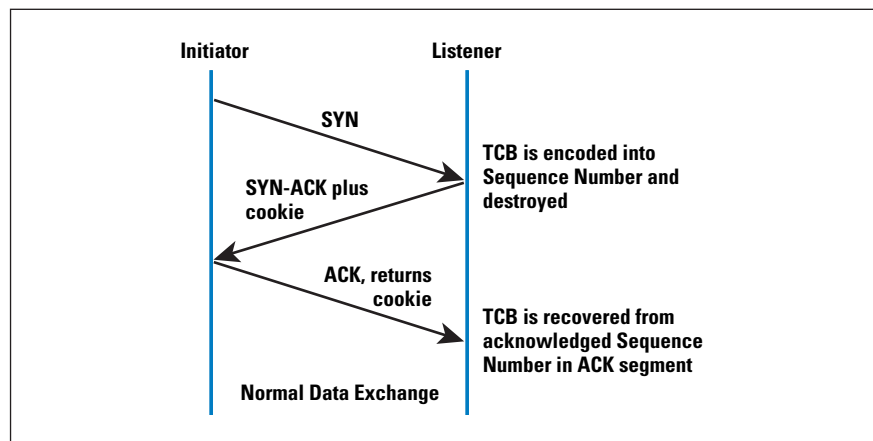
The second class involves hardening the network, either to lessen the likelihood of the attack preconditions (an army of controlled hosts or the propagation of IP packets with spoofed source addresses), or to insert middleboxes that can isolate servers on the networks behind them from illegitimate SYNs.

End-Host Countermeasures

Increasing TCP Backlog: Because the basic attack mechanism relies on overflowing a host's backlog of connecting sockets, an obvious end host-based solution is to simply increase the backlog, as is already done for very popular server applications. In at least some popular TCP implementations, this solution is known to be a poor one because of the use of linear list traversal in the functions that attempt to free state associated with stale connection attempts. Increasing the backlog is typically possible through altering the *listen()* call of an application and setting an operating system kernel parameter named SOMAXCONN, which sets an upper bound on the size of the backlog that an application can request. This step by itself should not be seriously considered as a means to defend against SYN flooding attacks—even in operating systems that can efficiently support large backlogs—because an attacker who can generate attack segments will most likely be able to scale to larger orders than the backlog supportable by a host.

Reducing the SYN-RECEIVED Timer: Another simple end host-based mechanism is to put a tighter limit on the amount of time between when a TCB enters the SYN-RECEIVED state and when it may be reaped for not advancing. The obvious disadvantage to this mechanism is that in cases of aggressive attacks that impose some amount of congestion loss in either the SYN-ACK or handshake-completing ACK packets, legitimate connection TCBs may be reaped as hosts are in the process of retransmitting these segments. Furthermore, there is only a linear relationship between the reduction that an administrator makes in the SYN-RECEIVED timer and the corresponding increase in packet rate that the adversary must make in order to continue attacking the server. Other alternative end-host solutions make it much more difficult for an attack to remain viable. For these reasons, a reduction in the SYN-RECEIVED timer is not an advisable defense against SYN flooding attacks.

Figure 4: Connection Establishment with SYN Cookies



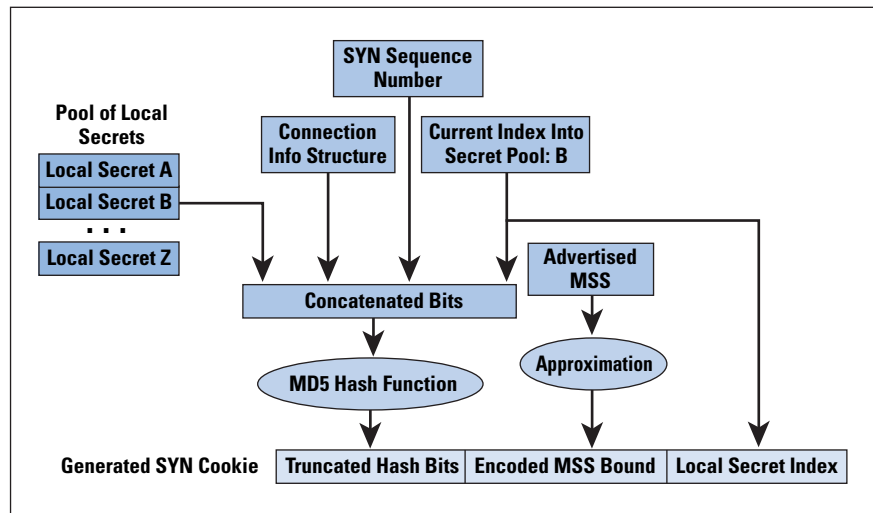
SYN Caches: Two end-host defenses, called SYN caches and SYN cookies (described later), operate by reducing the amount of state allocated initially for a TCB generated by a received SYN, and putting off instantiating the full state^[8]. In a host that uses a SYN cache, a hash table with a limited amount of space in each hash bucket is used to store a subset of the data that would normally go into an allocated TCB. If and when a handshake completing ACK is received, this data can be moved into a full TCB; otherwise the oldest bucket at a particular hash value can be reaped when needed. In Lemon's FreeBSD example^[8], the SYN cache entry for a half connection is 160 bytes, versus 736 bytes for a full TCB, and 15359 entries in the SYN cache are supported.

The SYN cache data structure is robust to attackers attempting to overflow its buckets because it uses the initiator's local port number and some secret bits in the hash value. Because stacks are a more effective data structure to search than a simple linked list, stacks that use a SYN cache can have improved speed, even when not under attack. Under Lemon's tests, during an active attack a host using a SYN cache was able to establish legitimate connections with only about a 15-percent increase in latency.

SYN Cookies: In contrast to the SYN cache approach, the SYN cookies technique causes absolutely zero state to be generated by a received SYN. Instead, the most basic data comprising the connection state is compressed into the bits of the sequence number used in the SYN-ACK. Since for a legitimate connection, an ACK segment will be received that echoes this sequence number (actually the sequence number plus one), the basic TCB data can be regenerated and a full TCB can safely be instantiated by decompressing the Acknowledgement field. This decompression can be effective even under heavy attack because there is no storage load whatsoever on the listener, only a computational load to encode data into the SYN-ACK sequence numbers. The downside is that not all TCB data can fit into the 32-bit Sequence Number field, so some TCP options required for high performance might be disabled. Another problem is that SYN-ACKs are not retransmitted (because retransmission would require state), altering the TCP synchronization procedures from RFC 793.

Recent work by Andre Oppermann uses the TCP Timestamp option in conjunction with the Sequence Number field to encode more state information and preserve the use of high-performance options such as TCP Window Scaling, and TCP *Selective Acknowledgment Options* (SACK), and can also be used to preserve *TCP-Message Digest 5* (MD5) support with SYN cookies. This option is a step forward, in that it removes the major negative effect of previous SYN cookie implementations that disabled these features.

Figure 5: Process for Generation and Validation of TCP SYN Cookies.



The exact format of TCP SYN cookies is not an interoperability issue, because they are only locally interpreted, and the format and procedures for generation and validation can vary slightly among implementations. Figure 5 depicts the general process of SYN cookie generation and validation used by multiple implementations.

To compute the SYN-ACK sequence number (that is, the TCP cookie) when using TCP cookies, a host first concatenates some local secret bits, a data structure that contains the IP addresses and TCP ports, the initial SYN sequence number, and some index data identifying the secret bits. An MD5 digest is computed over all these bytes, and some bits are truncated from the hash value to be placed in the SYN-ACK sequence number. Because the sequence number is about a fourth the size of the full hash value, this truncation is necessary, but generally at least 3 bytes worth of the hash bits are used, meaning that there should still be close to a 2^{24} effort required to guess a valid cookie without knowing the local secret bits. In addition to the hash output, some of the cookie bits indicate a lower bound on the *Maximum Segment Size* (MSS) that the SYN contained, and the index bits identifying the local secret used within the hash.

To validate a SYN cookie, first the acknowledgement number in an incoming ACK segment is decremented by 1 to retrieve the generated SYN cookie. The valid value for the set of truncated hash bits is computed based on the IP address pair, TCP port numbers, segment sequence number minus one, and the value from the secret pool corresponding to the index bits inside the cookie. If these computed hash bits match those within the ACK segment, then a TCB is initialized and the connection proceeds. The encoded MSS bound is used to set a reasonable-sized MSS that is no larger than what was originally advertised. This MSS is usually implemented as three bits whose code points correspond to eight “commonly advertised” MSS values based on typical link *Maximum Transmission Units* (MTUs) and header overheads.

Hybrid Approaches: A hybrid approach combines two or more of the single defense techniques described previously. For instance, some end-host operating systems implement both a large backlog and SYN cookies, but enable SYN cookies only when the amount of the backlog that is occupied exceeds some threshold, allowing them to normally operate without the disadvantages of SYN cookies, but also allowing them to fail over to the SYN-cookie behavior and be strongly protected when an attack occurs.

Network-Based Countermeasures

Filtering: The most basic network-level defense is application of the filtering techniques described in RFC 2827^[7]. Using ingress filtering, an ISP refuses to further route packets coming from an end site with IP source addresses that do not belong to that end site. Ingress filtering would be highly effective at preventing SYN flooding attacks that rely on spoofed IP packets. However, it is not currently reliable because ingress filtering policies are not universally deployed. Ingress filtering is also wholly ineffective against SYN flooding attacks that use a distributed army of controlled hosts that each directly attack. Ingress filtering is also a mechanism that an end site wishing to defend itself most often has no control over, because it has no influence upon the policies employed by ISPs around the world.

Firewalls and Proxies: A firewall or proxy machine inside the network can buffer end hosts from SYN flooding attacks through two methods, by either spoofing SYN-ACKs to the initiators or spoofing ACKs to the listener^[9].

Figure 6 shows the basic operation of a firewall/proxy that spoofs SYN-ACKs to the initiator. If the initiator is legitimate, the firewall/proxy sees an ACK and then sets up a connection between itself and the listener, spoofing the initiator's address. The firewall/proxy splits the end-to-end connection into two connections to and from itself. This splitting works as a defense against SYN flooding attacks, because the listener never sees SYNs from an attacker. As long as the firewall/proxy implements some TCP-based defense mechanism such as SYN cookies or a SYN cache, it can protect all the servers on the network behind it from SYN flooding attacks.

Figure 6: Packet Exchanges through a SYN-ACK spoofing Firewall/Proxy.

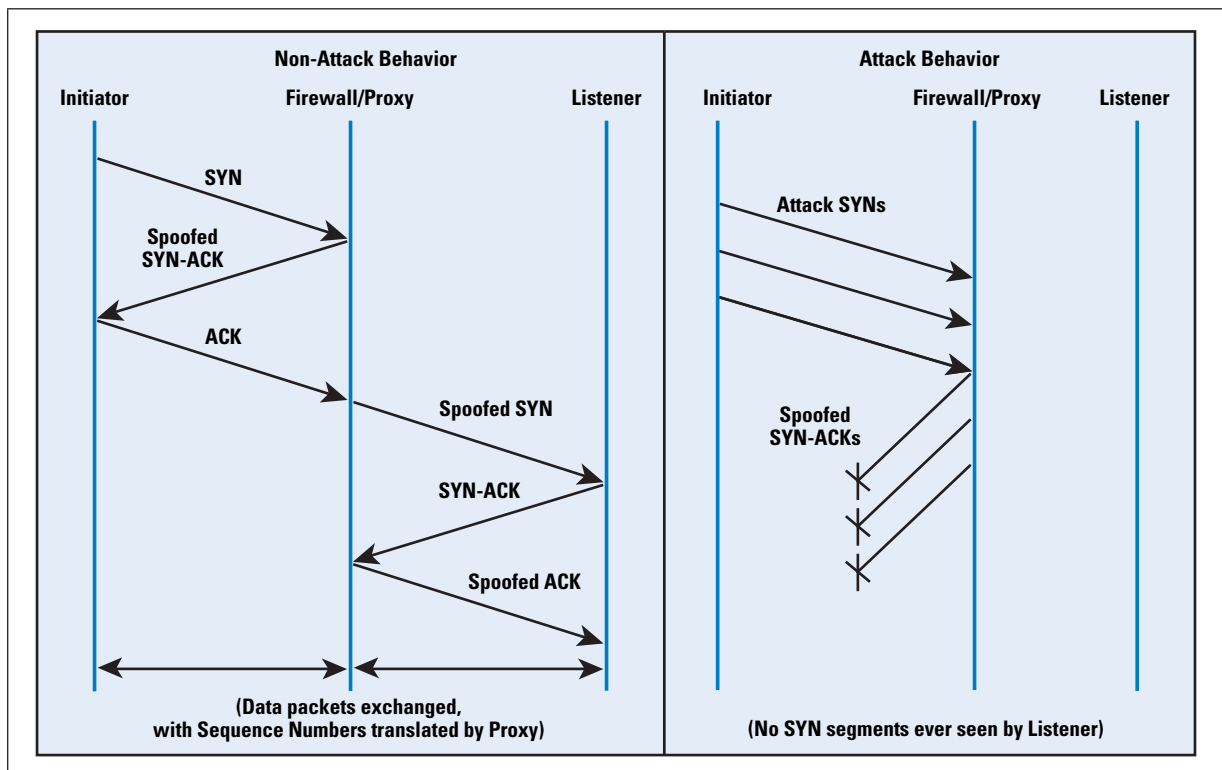
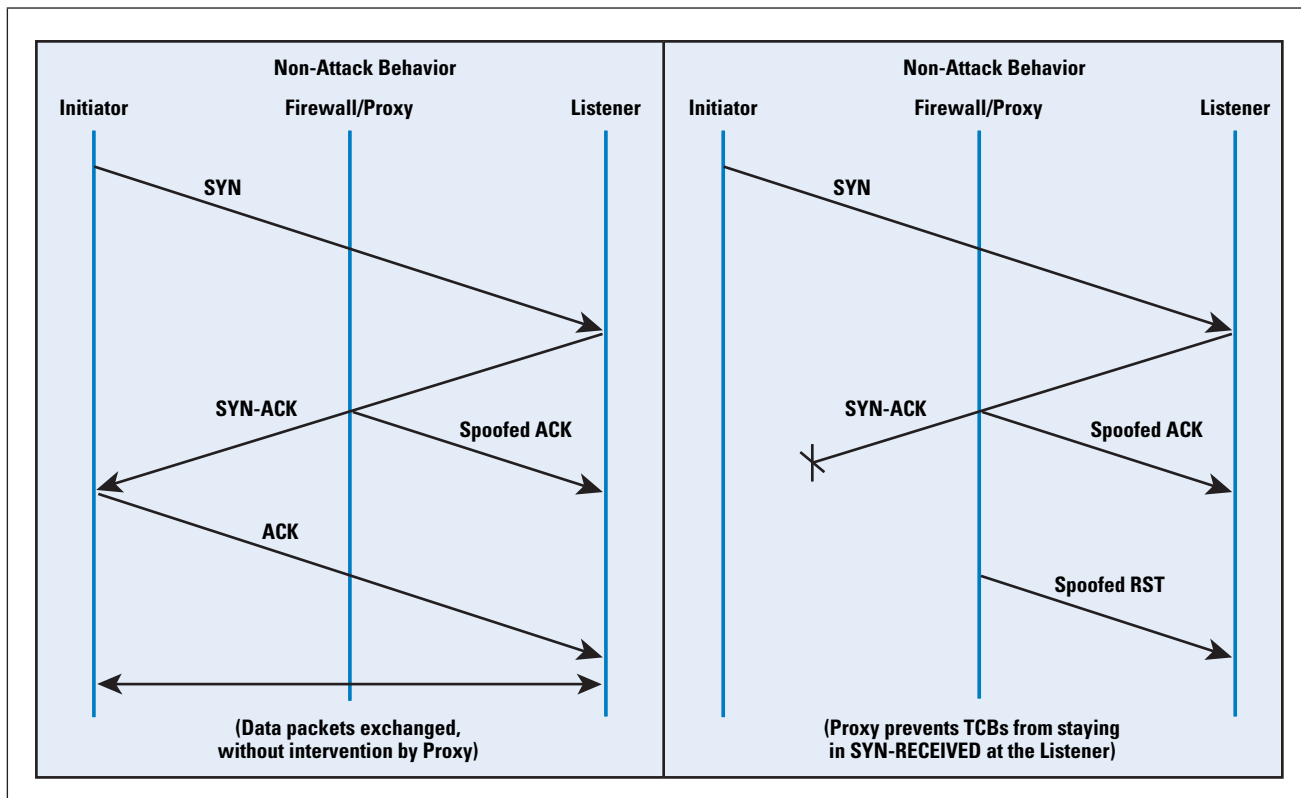


Figure 7 illustrates the packet exchanges through a firewall/proxy that spoofs ACKs to the listener in response to observed SYN-ACKs. This spoofing prevents the listeners TCBS from staying in the SYN-RECEIVED state, and thus maintains free space in the backlog. The firewall/proxy then waits for some time, and if a legitimate ACK from the initiator is not observed, then it can signal the listener to free the TCB using a spoofed TCP RST segment. For legitimate connections, packet flow can continue, with no interference from the firewall/proxy. This solution is more desirable than the mode of operation in Figure 5, where the firewall/proxy spoofs SYN-ACKs, because it does not require the firewall/proxy to actively participate in legitimate connections after they are established.

Figure 7: Packet Exchanges through an ACK-spoofing Firewall/Proxy.



Active Monitor: An active monitor is a device that can observe and inject traffic to the listener, but is not necessarily within the routing path itself, like a firewall is. One type of active monitor acts like the ACK-spoofing firewall/proxy of Figure 6, with the added capability of spoofing RSTs immediately if it sees SYNs from source addresses that it knows to be used by attackers^[9]. Active monitors are useful because they may be cheaper or easier to deploy than firewall-based or filtering solutions, and can still protect entire networks of listeners without requiring every listener’s operating system to implement an end-host solution.

Defenses in Practice

Both end-host and network-based solutions to the SYN flooding attack have merits. Both types of defense are frequently employed, and they generally do not interfere when used in combination. Because SYN flooding targets end hosts rather than attempting to exhaust the network capacity, it seems logical that all end hosts should implement defenses, and that network-based techniques are an optional second line of defense that a site can employ.

End-host mechanisms are present in current versions of most common operating systems. Some implement SYN caches, others use SYN cookies after a threshold of backlog usage is crossed, and still others adapt the SYN-RECEIVED timer and number of retransmission attempts for SYN-ACKs.

Because some techniques are known to be ineffective (increasing backlogs and reducing the SYN-RECEIVED timer), these techniques should definitely not be relied upon. Based on experimentation and analysis (and the author's opinion), SYN caches seem like the best end-host mechanism available.

This choice is motivated by the facts that they are capable of withstanding heavy attacks, they are free from the negative effects of SYN cookies, and they do not need any heuristics for threshold setting as in many hybrid approaches.

Among network-based solutions, there does not seem to be any strong argument for SYN-ACK spoofing firewall/proxies. Because these spoofing proxies split the TCP connection, they may disable some high-performance or other TCP options, and there seems to be little advantage to this approach over ACK-spoofing firewall/proxies. Active monitors should be used when a firewall/proxy solution is administratively impossible or too expensive to deploy. Ingress and egress filtering is frequently done today (but not ubiquitous), and is a commonly accepted practice as part of being a good neighbor on the Internet. Because filtering does not cope with distributed networks of drones that use direct attacks, it needs to be supplemented with other mechanisms, and must not be relied upon by an end host.

Related Attacks

In addition to SYN flooding, several other attacks on TCP connections are possible by spoofing the IP source address and connection parameters for in-progress TCP connections^[10]. If an attacker can guess the two IP addresses, TCP port numbers, and a valid sequence number within the window, then a connection can be disrupted either through resetting it or injecting corrupt data. In addition to spoofed TCP segments, spoofed ICMP datagrams have the capability to terminate victim TCP connections.

Both these other attacks and SYN floods target a victim's TCP application and can potentially deny service to the victim using an attack rate less than that of brute-force packet flooding. However, SYN flooding and other TCP spoofing attacks have significant differences. SYN flooding denies service to new connections, without affecting in-progress connections, whereas other spoofing attacks disrupt in-progress connections, but do not prevent new connections from starting. SYN flooding attacks can be defended against by altering only the initial handshaking procedure, whereas other spoofing attacks require additional per-segment checks throughout the lifetime of a connection. The commonality between SYN flooding and other TCP spoofing attacks is that they are predicated on an attacker's ability to send IP packets with spoofed source addresses, and a similar defense against these attacks would be to remove this capability through more universal deployment of address filtering or IPsec.

Conclusion

At the time of this writing, the TCP SYN flooding vulnerability has been well-known for a decade. This article discussed several solutions aimed at making these attacks ineffective, some of which are readily available in commercial off-the-shelf products or free software, but no solution has been standardized as a part of TCP or middlebox function at the IETF level. The IETF's *TCP Maintenance and Minor Extensions* (TCPM) working group is in the process of producing an informational document that explains the positive and negative aspects of each of the common mitigation techniques^[10], and readers are encouraged to consult this document for further information.

In this author's opinion, some variant of the SYN cache technique should be a mandatory feature to look for in a server operating system, and the variant can be deployed in combination with other network-based methods (address-based filtering, ACK-spoofing firewalls, IPsec, etc.) in appropriate situations. It is encouraging to see that protocol designers have learned a lesson from the SYN flooding vulnerability in TCP and have made more recent protocols inherently robust to such attacks.

Acknowledgements

Several individual participants in the IETF's TCPM working group have contributed bits of data found in the group's informational document on SYN flooding^[11], some of which is replicated in spirit here.

References

- [1] daemon9, route, and infinity, "Project Neptune," *Phrack Magazine*, Volume 7, Issue 48, File 13 of 18, July 1996.
- [2] CERT, "CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks," September 1996.
- [3] Aura, T. and P. Nikander, "Stateless Connections," Proceedings of the First International Conference on Information and Communication Security, 1997.
- [4] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol," RFC 2960, October 2000.
- [5] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol," RFC 4306, December 2005.
- [6] Aura, T., Nikander, P., and J. Leiwo, "DOS-resistant Authentication with Client Puzzles," *Lecture Notes in Computer Science*, Volume 2133, revised from the 8th International Workshop on Security Protocols, 2000.

- [7] Ferguson, P. and D. Senie, “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing,” BCP 38, RFC 2827, May 2000.
- [8] Lemon, J., “Resisting SYN Flood DoS Attacks with a SYN Cache,” BSDCON 2002, February 2002.
- [9] Schuba, C., Krsul, I., Kuhn, M., Spafford, E., Sundaram, A., and D. Zamboni, “Analysis of a Denial of Service Attack on TCP,” Proceedings of the 1997 IEEE Symposium on Security and Privacy, 1997.
- [10] Touch, J., “Defending TCP Against Spoofing Attacks,” Internet-Draft (work in progress), **draft-ietf-tcpm-tcp-antispoof-05**, October 2006.
- [11] Eddy, W., “TCP SYN Flooding Attacks and Common Mitigations,” Internet-Draft (work in progress), **draft-ietf-tcpm-syn-flood-00**, July 2006.

WESLEY M. EDDY works for Verizon Federal Network Systems as an onsite contractor at NASA’s Glenn Research Center, where he performs research, analysis, and development of network protocols and architectures for use in space exploration and aeronautical communications. E-mail: **weddy@grc.nasa.gov**

Boosting the SOA with XML Networking

by Silvano Da Ros

In the 1990s, the widespread adoption of *object-oriented programming* (OOP) and advancing network technologies fostered the development of distributed object technologies, including *Object Management Group's* (OMG's) *Common Object Request Broker Architecture* (CORBA) and Microsoft's *Distributed Common Object Model* (DCOM). Both CORBA and DCOM follow the OOP consumer-producer service model, where applications locally instantiate any number of objects and execute methods for the objects to obtain a service. However, with *distributed* object technologies, a local application can request a service from a remote application by instantiating a remote object and executing the methods of the object using *Remote Procedure Call* (RPC) over the network. The local application executes the methods of the remote object as if the object were an inherent part of the local application.

To push toward a simpler consumer-producer service model than distributed objects, the *Service-Oriented Architecture* (SOA) was created as a worldwide standards-based application interoperability initiative^[1]. SOA differs from distributed object technologies, because you no longer deal with object instantiation and method invocation to provide services between your applications^[2]. Instead, you can create *Extensible Markup Language* (XML)-based standard Web services to exchange XML documents between your applications using Internet-based application layer protocols, such as *Hyper Text Transfer Protocol* (HTTP) and the *Simple Mail Transfer Protocol* (SMTP).

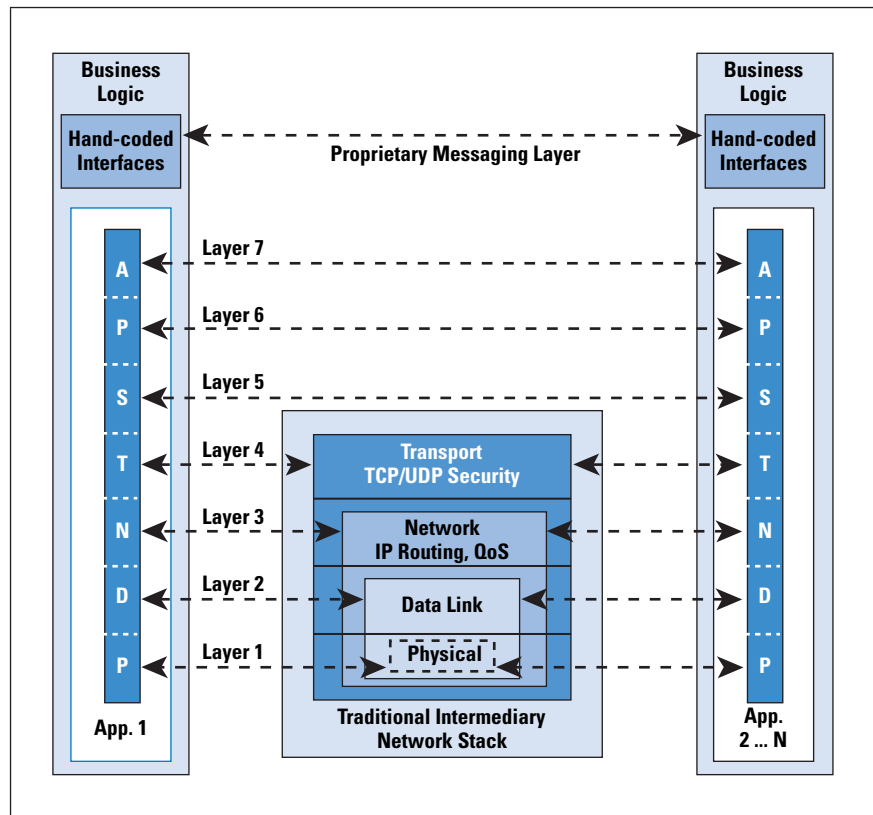
This is where XML networking comes into the picture. This article shows how to use SOA at the network edge, in conjunction with XML within the network, to help with the work required for enabling interoperability between your applications. The problem with SOA on its own is that to scale applications, hardware and software upgrades are required on the servers where your business logic resides. Because application integration using XML is CPU-intensive, it benefits from XML hardware built specifically for XML computations. However, the applications servers that run your business logic are effectively independent of the underlying XML processing. Therefore, to accelerate the SOA at the network level transparently to the application, XML networking technologies can be used. XML networking can provide SOA acceleration using a special middleware-enabled network layer, which this article explains. This special network layer also provides additional benefits to your applications that SOA alone cannot provide at the edge, such as dynamic message routing and transformation.

To help in the understanding of SOA acceleration with XML networking, the following section discusses SOA and its constituent technologies. Further sections explore the specifics of XML and XML-based network processing.

A Brief History of SOA

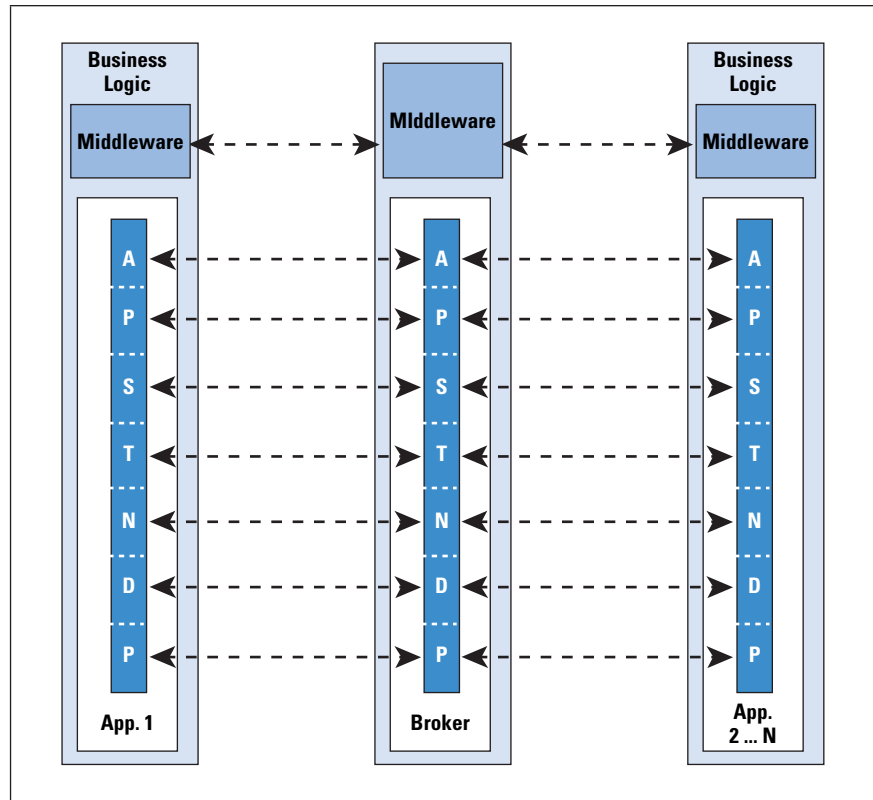
Traditionally, hand-coding proprietary interfaces were required to interoperate between your applications, as Figure 1 illustrates. This task is a trivial one if you have only a few applications, but if you have numerous disparate applications, all requiring interfaces into one another, the result is a complex, many-to-many web of connections between them. In the 1980s, *Electronic Data Interchange* (EDI) was developed to standardize the message formats that trading partners use to exchange text-based transaction data residing on mainframes, making it an early predecessor to SOA.

Figure 1: The Proprietary Messaging Layer



In the mid-1990s, standard middleware (or integration brokers) became available, such as CORBA and DCOM mentioned previously, to integrate advanced client-server applications. Figure 2 shows how integration brokers allow you to perform the translations between end systems over a standard messaging layer without creating application-specific interfaces between each system. During the same time, numerous software vendors, such as IBM WebSphere and TIBCO, also developed standard messaging layer protocols, which required adding vendor-specific adapters within the common integration brokers. Additionally, with newer application development environments being adopted, such as *Java 2 Sun Enterprise Edition* (J2EE) and Microsoft .NET, even more programming complexity is required when considering application interoperability without using the SOA. Fortunately, these new platforms currently support the SOA, allowing an application developed in one platform to tap into the data supplied by an application developed in the other.

Figure 2: The Standard Messaging Layer



Figures 1 and 2 illustrate how the proprietary and standard messaging layers sit above the network stack—at best, a traditional network device can operate only up to and including the transport layer. For example, by tracking TCP connection state information, a firewall device allows you to configure security services for your applications. Some firewalls can inspect the context of the application, but only to ensure the application behavior is RFC-compliant and not performing some sort of malicious activity. Additionally, at the next layer down the stack, you can configure Layer 3 *Quality of Service* (QoS) functions, such as *IP Precedence*, *Differentiated Services* (DiffServ), traffic shaping, and resource reservation, to ensure delivery of traffic to your critical applications. Although the network layers can provide these intelligent network services to your applications, they do not add any value toward accelerating your SOA.

Notice how the middleware portion in the proprietary messaging layer in Figure 1 takes up a larger portion of the application stack than the standard messaging layer from Figure 2. This situation occurs because the list of available messages that your standard messaging layer applications support is now much smaller—the broker takes care of the interfacing complexity on behalf of your applications. A reduced number of messages requires that you maintain much less middleware programming code on your applications than if every application in your network had to account for the messages of every other application.

Optimizing the SOA

Now that you understand SOA, you can better understand where XML networking fits into the scheme of things. Figure 3 illustrates how network equipment vendors can add specialized “application-aware” intelligence into Layers 5 through 7 of the OSI model.

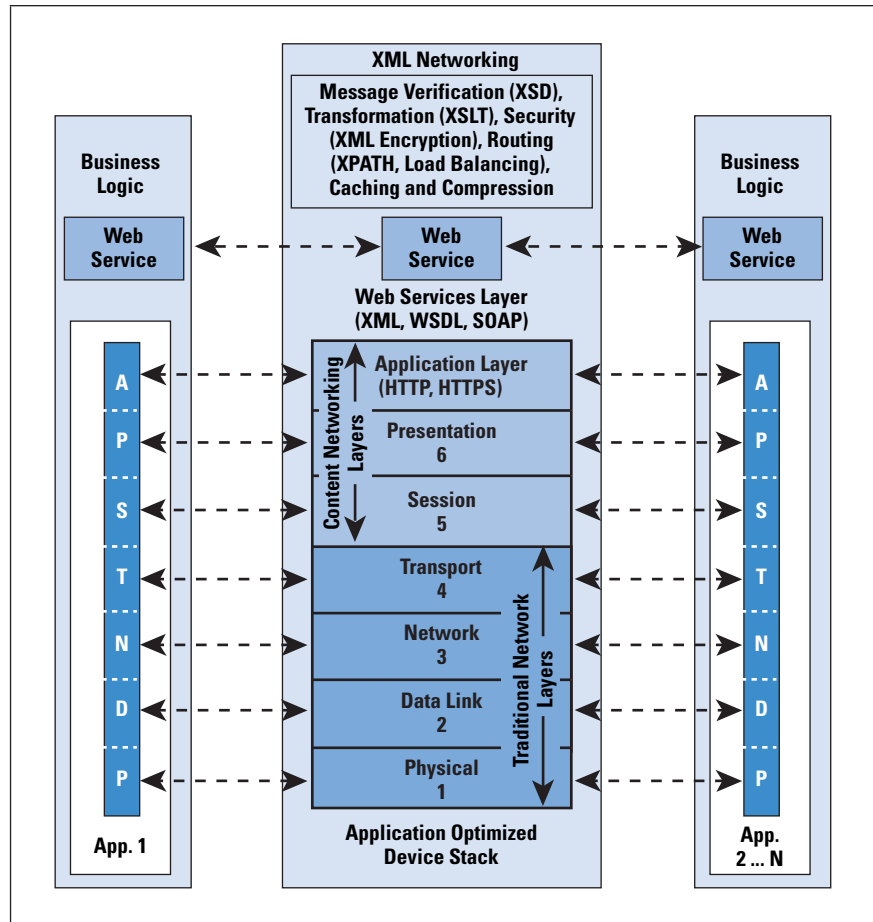
You can start by contrasting XML networking with traditional content networking technologies^[3]. As you can see in Figure 3, by incorporating content networking services into the network, such as *Server Load Balancing* (SLB), caching, and *Secure Sockets Layer* (SSL) acceleration, network vendors give you the ability to transparently accelerate your applications without the need of application hardware upgrades. However, by residing only within the OSI model, content networking services and protocols provide a “network-oriented” way to accelerate your applications. In order to achieve full application awareness, you must look not only into the application headers, but also into the application payload. Although the content networking protocols can inspect into the packet payload, they are meant for providing network layer services but *not* application integration services. For example, *Network-Based Application Recognition* (NBAR) allows you to mark the IP DiffServ field in packets containing high-priority application traffic by first detecting the behavior of the application. However, like the network layers, the content networking layers cannot fulfill SOA acceleration requirements either.

In contrast, XML networking provides integration services by inspecting the full context of the application transaction and adding XML standards-based intelligence on top of the TCP/IP stack. An XML-enabled network provides you greater control, flexibility, and efficiency for integrating your applications than integration brokers. Figure 3 shows how you can inspect the XML-based “Web services” layer to accelerate your applications developed within an SOA model without the need of an integration broker.

The most popular Web services protocol is *Simple Object Access Protocol* (SOAP)^[4]. With SOAP, your applications can request services from one another with XML-based requests and receive responses as data formatted with XML. Because SOAP uses XML, its Web services are self-descriptive and very simple to use.

You define your SOAP Web services with the XML-based *Web Services Description Language* (WSDL)^[5]. The WSDL binds the SOAP messages to the Web services layer, as discussed later in this article. You can then transport your SOAP messages over standard application layer protocols, such as HTTP and HTTPS, between your client-server applications.

Figure 3: The Web Services Layer

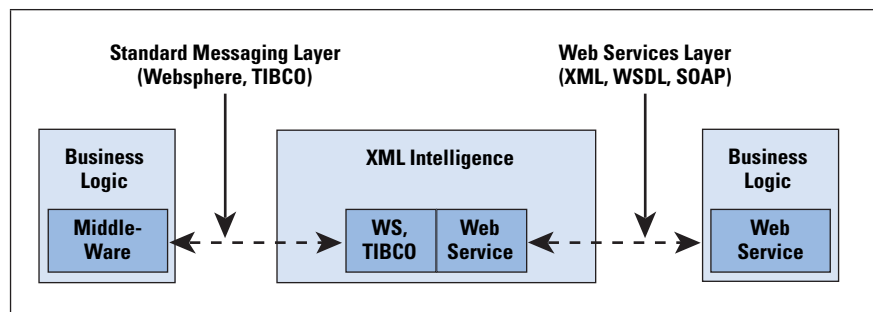


Similar to content networking technologies, XML networking can transparently add value to your applications within the network. When the XML network device receives the standard XML document from the Web services layer, you can configure the device to perform application-oriented services on the document. But because XML networking operates at the middleware layer and uses standard documents to integrate your applications, it provides you with fully standard functions using languages for:

- *Message Verification:* You can develop *World Wide Web Consortium (W3C) XML Schema Definitions (XSDs)* that your XML network can use to verify the syntax of your XML documents^[6].
- *XML Translation:* Using *XML Stylesheet Language Transformations (XSLT)*, you can translate XML documents to other non-XML formats, and conversely, directly within your network^[7].
- *Context-Based Routing:* Use *XML Path (XPath)* to route messages based on data stored within XML documents^[8]. A popular example is to route stock exchange quotes to a desired location when the value of the stock drops below a certain threshold.
- *High Availability:* Messages containing specific content can be load balanced across numerous identical origin servers.

- *Data Security*: You can accelerate XML encryption computations using either hardware or software XML-accelerated devices.
- *Compression and Caching*: You can cache frequently requested XML documents and compress XML documents to reduce network bandwidth. Like XML encryption, XML caching and compression can be performed using either hardware or software XML-accelerated devices.
- *Application-Layer Request Translation*: You can use XML networking to convert non-Web service requests into standard Web service requests. As with integration brokers, vendor-specific adapters are required to translate between WebSphere MQ, TIBCO, and SOA Web services. For example, Figure 4 shows how you can use network-level XML intelligence to translate between Websphere or TIBCO messages and Web services layer XML-based messages.

Figure 4: Intelligent Protocol Switching



Introducing XML Service Languages

Now that you have a general understanding of both SOA and XML networking, you can examine the specific XML technologies used for application interoperability, including:

- *XML* is used to format application data for storage and transmission.
- *XSLT* is used to translate between one XML format to another.
- *XSD* is used to describe, control, and verify an XML document format.
- *XPath* is a way to address items in an XML document hierarchy.
- *SOAP* is a messaging protocol used to encode information in Web service request and response messages before sending them over a network.

XML has its roots in the late 1960s from the *Generalized Markup Language* (GML), which was used to organize IBM's mainframe-based legal documents into a searchable form. The *Standard Generalized Markup Language* (SGML) was officially standardized in 1986 as an ISO international norm (ISO 8879). Since then, XML has become the predominant markup language for describing content. XML differs from HTML because it is not concerned with presenting or formatting content; instead, XML is used for describing the data using tags and attributes that you define yourself. Figure 5 is a sample XML document that organizes a police department's traffic ticket information.

Figure 5: A Traffic Ticket XML Example

```
<?xml version="1.0"?>
<dept-tickets>
  <dept-chief>Greg Sanguinetti"/>
  <dept-id>12389289/>
  <ticket id="034567910" code="301">
    <offender>
      <name>John Smith</name>
      <license-number>10003887</license-number>
      <plate-number>9AER9876</plate-number>
    </offender>
    <offence-date>09/30/2005</offence-date>
    <location>
      <state>CA</state>
      <city>SJ</city>
      <intersection>West Tasman Dr.-Great America Pkwy.</intersection>
    </location>
    <officer>
      <officer-name>Paul Greene</officer-name>
      <officer-badge>7652323</officer-badge>
      <cruiser-plate-number>6TYX0923</cruiser-plate-number>
    </officer>
    <description>Failure to stop at red light</description>
    <fine>100</fine>
  </ticket>
  <ticket id="..." code="...">
    ...
  </ticket>
  <ticket id="..." code="...">
    ...
  </ticket>
</dept-tickets>
```

The XML in Figure 5 identifies the group of tickets for a police department by the department ID and the department chief's name. This example gives the data for a single traffic ticket as defined by the "ticket" element (or tag); however, you could include as many tickets as you want within the element "dept-tickets." The "ticket" element has two self-explanatory attributes (in dark blue), called "id" and "code," referring to the identification number for the individual ticket and the offense code, respectively. The sub-elements of the "ticket" element are also self-explanatory: "offender," "offence-date," "location," "officer," "description" and "fine."

In order to build a well-formed XML document, you must embrace the data for each element within its respective open and close tags (for example, <ticket>...</ticket>, or <ticket>.../>), properly nest all elements, and make sure all element names are in the proper case. You must also specify the XML version with the "<?xml version="1.0"?>" tag at the beginning of the XML document. HTML is less rigid than XML because it is case-insensitive and most Web browsers will allow you to leave out the close tags of an element. Because XML is very strict about following the correct syntax (that is, by making sure the XML is well-formed), XML processors are much easier to develop than traditional HTML Web browsers.

To verify that your documents are valid XML, you can check them against XSD files, which define the XML elements and their sequence, as discussed later in this article.

Transforming XML Using XSLT

You can use XSLT to translate one XML-based language into another. For example, you can translate standard XML into HTML. To translate the XML from Figure 5 into HTML for online viewing, you can use the XSLT file in Figure 6.

Figure 6: XSLT Translation – From XML to HTML

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <br><b>Chief: </b><xsl:value-of select="dept-tickets/dept-chief"/></br>
        <br><b>Department No: </b><xsl:value-of select="dept-tickets/dept-id"/>
        </br>
        <table border="5">
          <!-- Output the HTML table headings -->
          <th>Ticket Number</th>
          <th>Offender's Name</th>
          <th>License Number</th>
          <th>State of Offense</th>
          <th>Officer's Name</th>
          <!-- Output the HTML table data -->
          <xsl:for-each select="dept-tickets/ticket">
            <tr>
              <td align="center"><xsl:value-of select="@id"/></td>
              <td align="left"><xsl:value-of select="offender/name"/></td>
              <td align="center"><xsl:value-of select="offender/license-number"/></td>
              <td align="center"><xsl:value-of select="location/state"/> </td>
              <td align="left"><xsl:value-of select="officer/officer-name"/></td>
              <td align="right">${xsl:value-of select="fine"/}</td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

You must use a namespace to differentiate elements among the XML-based languages that you use in your XML document. As Figure 6 illustrates, the namespace is the string “xsl:”, which prefixes all of the XSLT elements. The particular application that parses the document (whether it is your XML device or a standalone XSLT parser^[9]) will know what to do with the specific elements based on the prefix. For example, an XSLT parser will look for the specific *Universal Resource Indicator* (URI) string constant that the W3C assigned to XSLT (that is, <http://www.w3.org/1999/XSL/Transform>) and perform the intended actions based on the elements in the document.

XML parsers do not use the URI of the namespace to retrieve a schema for the namespace—it is simply a unique identifier within the document. According to W3C, the definition of a namespace simply defines a two-part naming system (for example, “xslt:for-each”) and nothing else. After you define the namespace, the XML parser will understand the elements used within the document, such as “for-each” and “value-of” specified in Figure 6. For XSD documents, you must use a different namespace URI (that is, <http://www.w3.org/2001/XMLSchema>), as the next section discusses.

When you configure an XSLT parser or XML networking device to apply XSLT to an XML document, the parser starts at the top of the XSLT document by matching the root XML element within the source XML file. For example, the `<xslt:template match="/">` element in Figure 6 matches the “dept-tickets” root element from the XML file in Figure 5. The XSLT parser then creates the destination XML document (that is, a well-formed HTML file, in this example) and outputs the `<html>` and `<body>` tags to the new document. The XSLT parser then outputs the HTML table headers and loops through the XML document “ticket” elements, outputting selected items within the columns of the HTML table. The resulting HTML is given in Figure 7 for three sample tickets.

Figure 7: Resulting HTML Table – Source View

```
<html>
<body>
<br><b>Chief: </b>Greg Sanguinetti<br><b>Department No: </b>12389289
<table border="5">
<th>Ticket Number</th><th>Offender's Name</th>
<th>License Plate</th><th>State of Offense</th>
<th>Officer's Name</th><th>Fine Amount</th>
<tr>
<td>034567910</td><td>John Smith</td><td>10003887</td>
<td>CA</td><td>Paul Greene</td><td>100</td>
</tr>
<tr>
<td>042562930</td><td>Gerald Rehnquist</td><td>11023342</td>
<td>CA</td><td>Joel Patterson</td><td>200</td>
</tr>
<tr>
<td>182736493</td><td>Jenny Barker</td><td>47281938</td>
<td>CA</td><td>Emily Jones</td><td>120</td>
</tr>
</table>
</body>
</html>
```

Figure 8 illustrates the resultant HTML table that clients would see within a Web browser after the XSLT translation takes place.

Figure 8: Resulting HTML Table –
Browser View

Ticket Number	Offender's Name	License Plate	State of Offense	Officer's Name	Fine Amount
034567910	John Smith	10003887	CA	Paul Greene	100
042562930	Gerald Rehnquist	11023342	CA	Joel Patterson	200
182736493	Jenny Barker	47281938	CA	Emily Jones	120

Verifying XML Using XSD

Because you can customize the structure and tags within an XML document, you should verify its syntax using XSDs. The XSD file in Figure 9 verifies the XML document given previously in Figure 5.

Figure 9: XSD File for Validating
Traffic Ticket XML

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="dept-tickets">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="dept-chief"/>
        <xsd:element name="dept-id"/>
        <xsd:element name="ticket" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="offender">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="name"/>
                    <xsd:element name="license-number"/>
                    <xsd:element name="plate-number"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="offence-date"/>
              <xsd:element name="location">
                ...
              </xsd:element>
              <xsd:element name="officer">
                ...
              </xsd:element>
              <xsd:element name="description"/>
              <xsd:element name="fine"/>
            </xsd:sequence>
            <xsd:attribute name="id"/>
            <xsd:attribute name="code"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

You must define an XSD namespace with the URI “<http://www.w3.org/2001/XMLSchema>” and prefix all the XSD elements that you use in the XSD file, such as “element,” “complex-type,” and “attribute,” with this namespace. At the top of your XSD file, you must specify the root XML element; the remaining elements within your XML document can be defined within the root element. Using the “complex-type” XSD element, you can specify elements that contain child elements (in contrast, “simple-type” indicates that the element does not contain any child elements). In this example, the “dept-tickets” element may contain a sequence of one or more child elements (as represented by the <xsd:sequence> element), including “dept-chief,” “dept-id,” and any number of element “ticket.”

Routing Messages Using XPATH

XPATH was developed primarily to be used with XSLT to transform the XML tags within an XML document based on the path of the data. Previously, in Figure 6, you saw how to select the entire list of tickets using the XSLT “select” attribute:

```
xsl:value-of select="dept-tickets/ticket"
```

However, within an XML network, you can also use XPATH to search within an XML document to route XML messages based on the values of the document data. For example, a state government may need the headquarters police department to route unpaid tickets that are within a tolerable threshold amount to the motor vehicle department for processing—there, the driver’s license can be suspended until the ticket is paid. However, those unpaid tickets that exceed a maximum threshold amount must be routed to the court service government department for processing. The court may decide to press further charges, depending on the driver’s previous driving record. Additionally, severe infractions, such as drunken or reckless driving, must be routed automatically to the court, regardless of whether the ticket is paid or not. The XPATH expression “dept-tickets/ticket” given previously returns the entire list of traffic tickets. Alternatively, if you want only the unpaid tickets with a fine value of greater than \$100, you could use the XPATH expression:

```
dept-tickets/ticket[@paid='no' and fine>100]
```

The XPATH symbol “@” here indicates that an attribute is being selected, and not an element. To select tickets with codes 309 and 310 (that is, fictitious codes for severe infractions), you can use the following XPATH expression:

```
dept-tickets/ticket[@code=309 or @code=310]
```

Using SOAP Web Services

SOAP provides a standard way to send transaction information over TCP/IP application protocols, such as HTTP. For example, you could create a SOAP request-response operation over HTTP for exchanging traffic ticket information between two applications. As Figure 10 illustrates, the requesting client application sends a “getFineRequest” message to the server, which in turn responds with the appropriate fine amount within a “getFineResponse” message.

Figure 10: A Sample SOAP Request-Response Operation

```

Client Request :
POST /getticketfine HTTP/1.1
Host: www.example.com
Content-Type: application/soap+xml;

<?xml version="1.0"?>
<soap:envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:body>
  <tn:getFineRequest xmlns:tn="http://example.com/getticketfine">
    <tn:ticket-id>034567910</tn:ticket-id>
  </tn:getFineRequest >
</soap:body>
</soap:envelope>

Server Response:
HTTP/1.1 200 OK
Content-Type: application/soap+xml;

<?xml version="1.0"?>
<soap:envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:body>
  <tf:getFineResponse xmlns:tf="http://example.com/getticketfine">
    <tf:fine>100</tf:fine>
  </tf:getFineResponse>
</soap:body>
</soap:envelope>

```

You encapsulate each SOAP message within the “Envelope” SOAP element. Within Envelope, you need to prefix the SOAP elements with the SOAP namespace, called “soap:” in this example, which you define as an attribute within Envelope. The “encodingStyle” attribute of the Envelope element defines the data types in the SOAP document. You must also define a custom namespace (that is, “tf,” which stands for “ticket-fine”), with which you prefix all the application-specific elements.

To define the structure of the SOAP Web service running within your applications, you can use WSDL, which you develop so that your clients know the exact specification of the services that they can request, the types of responses they should expect to receive, and the protocols (for example, SOAP or HTTP) with which they should send messages.

For example, you can publish the WSDL to your clients, who may not be aware of the messages available within your Web services layer. The clients can retrieve the WSDL file and send the appropriate SOAP messages to the SOAP Web service running on your application. To publish the WSDL file to your clients, you can use a publicly available *Universal Description, Discovery and Integration* (UDDI) registry, such as XMethods^[10], or you could create your own UDDI registry^[11].

WSDL uses XSD to define your SOAP application data types. For example, for one application to request a fine amount (of XSD type `xs:integer`) for a given ticket ID (of XSD type `xs:string`) from your SOAP Web service called “ticketFineService,” you could use the WSDL in Figure 11.

Figure 11: WSDL for SOAP Request-Response Operation

```
<?xml version="1.0"?>
<definitions name="TicketInfo"
  targetNamespace="http://example.com/ticketinfo.wsdl"
  xmlns:tns="http://example.com/ticketinfo.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="getFineRequest">
    <part name="ticket-id" type="xs:string"/>
  </message>

  <message name="getFineResponse">
    <part name="value" type="xs:integer"/>
  </message>

  <porttype name="ticketFine">
    <operation name="getTicketFine">
      <input message="tns:getFineRequest"/>
      <output message="tns:getFineResponse"/>
    </operation>
  </porttype>

  <binding name="ticketBinding" type="ticketFine">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getTicketFine">
      <soap:operation soapAction="getTicketFine"/>
      <input>
        <soap:body use="encoded"/>
      </input>
      <output>
        <soap:body use="encoded"/>
      </output>
    </operation>
  </binding>

  <service name="ticketFineService">
    <documentation>WSDL File for ticketFineService</documentation>
    <port name="ticketFine" binding="ticketBinding">
      <soap:address location="http://example.com/getticketfine"/>
    </port>
  </service>
</definitions>
```

You start your WSDL file by declaring all the required namespaces. In order for the WSDL file to refer to element names that are defined within the same file (for example, “tns:getFineRequest” within the “porttype” element), you must use the “targetNamespace” element to define a custom URI that your custom namespace uses (that is, “tns,” meaning “this name space”).

You define the WSDL namespace for SOAP-specific elements with *xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap*. For WSDL-only elements, you can use the default namespace *xmlns=http://schemas.xmlsoap.org/wsdl/*. Note that elements within the file that do not have a prefix use the default namespace.

After you create the namespaces for the WSDL file, you can then create the two messages for the transaction, “getFineRequest” and “getFineResponse,” using WSDL “message” elements. WSDL ports create the request-response transaction flow using the “operation” element, by specifying which message is the request (input) and which is the response (output). After you define the transaction, you must bind it to SOAP with WSDL using the WSDL “binding” element. Additionally, to set the transport to HTTP, you must use the “binding” SOAP-specific element. You then link the operation you created previously within the WSDL “port-type” element to SOAP using the “operation” subelement within the parent “binding” element.

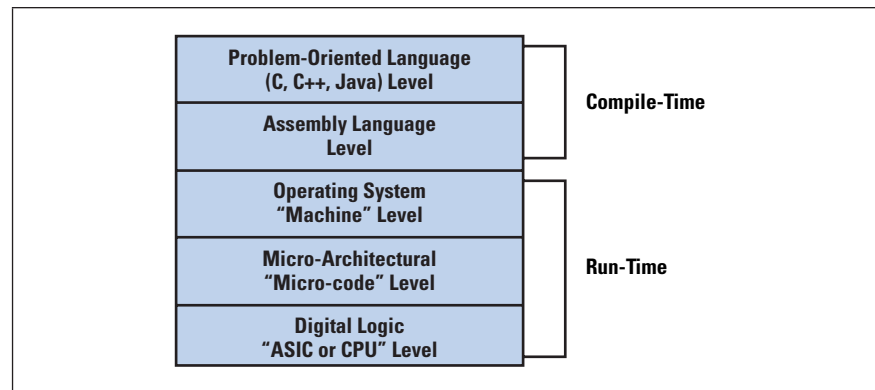
If you set the “use” element to “encoding,” you do not need to use an XSD “type” attribute for defining SOAP data types in your SOAP messages. However, you must specify the “encodingStyle” URI to **http://www.w3.org/2001/12/soap-encoding**, as you learned previously in Figure 10. Otherwise, if you set “use” to “literal,” then you would need to use the *type="xsd:string"* attribute in Figure 10 within the “tn:ticket-id” element when sending a request.

To define the SOAP Web service, you must use the WSDL “service” element. The SOAP element “address” within this element is the location where SOAP clients can send the “getTicketFine” requests, as Figure 10 illustrates.

Hardware vs. Software XML Acceleration

To help you understand the difference between hardware and software XML acceleration, Figure 12 illustrates a typical multilevel computer architecture^[12]. The highest level is where you would typically program your applications. When you compile your application, the compiler would typically “assemble” your various objects into assembly language prior to generating the machine-level code. This machine code is what is normally stored in an “.exe” file, which only your operating system can understand. When you execute the “.exe” at run time, the operating system converts the machine-level code into microcode, which the digital logic level within the CPU hardware can execute directly.

Figure 12: A Multilevel Computer Architecture



Examples of software-based network applications in the past that have transitioned to hardware acceleration include IP routing, encryption, firewalling, caching, and load balancing. XML networking is also a recent candidate for hardware acceleration; it is available by XML vendors that use XML *Application-Specific Integrated Circuits* (ASICs) or *Field Programmable Gate Arrays* (FPGAs) in their products^[13]. By programming the digital logic layer with the necessary circuits to perform intensive XML computations such as XSLT transformation, XML encryption, and XML schema validation, you can drastically increase the performance of the hardware platform.

However, some vendors have also found clever ways of accelerating XML computations on general-purpose hardware. Accelerating XML in software requires bypassing the additional machine-level step at run time. By “compiling” XML-based language instructions directly into microcode at the micro-architectural level, you can introduce XML computations to the underlying hardware directly at run time. That is, executing XML microcode at the digital logic level bypasses additional processing at the operating system “machine” level.

Summary

When a technology matures as a software agent running within an application, the need often arises to move the agent’s functions to the network. Indeed, this was the case with numerous software-based technologies of the past, such as IP routing, encryption, stateful firewall filtration, and server load balancing.

To facilitate the interoperability of diverse applications, SOA was developed as a prescription to complexity problems faced by commonly used distributed-object technologies. As SOA matures, the need to introduce XML-based functions to the network will grow. In order to streamline the responsibilities of an SOA-based application, you can transition your XML technologies, such as XML translation, validation, and security, from within the application to an XML-enabled network.

For Further Reading

- [1] Hao He, “What Is Service-Oriented Architecture?” O’Reilly, September 30, 2003,
<http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>
- [2] Werner Vogels, “Web Services Are Not Distributed Objects,” *Computing in Science and Engineering*, November-December 2003, <http://computer.org/internet/>
- [3] Christophe Deleuze, “Content Networks,” *The Internet Protocol Journal*, Volume 7, Number 2, June 2004.
- [4] “SOAP Version 1.2 Part 0: Primer,” W3C Recommendation, 24 June 2003,
<http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>
- [5] “Web Services Description Language (WSDL) Version 2.0 Part 0: Primer,” W3C Working Draft, 3 August 2005,
<http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803/>
- [6] “XML Schema Part 0: Primer Second Edition,” W3C Recommendation, 28 October 2004,
<http://www.w3.org/TR/xmlschema-0/>
- [7] “XSL Transformations (XSLT), Version 2.0,” W3C Recommendation, 16 November 1999,
<http://www.w3.org/TR/xslt>
- [8] “XML Path Language (XPath) Version 1,” W3C Recommendation, 16 November 1999, <http://www.w3.org/TR/xpath>
- [9] www.xmlspy.com
- [10] www.xmethods.net
- [11] www.uddi.org
- [12] Andrew S. Tanenbaum, *Structured Computer Organization*, (5th edition), ISBN 978-0131485211, Prentice Hall, 2005.
- [13] Michael John Sebastian Smith, *Application-Specific Integrated Circuits*, ISBN 978-0201500226, Addison-Wesley, June 1997.

SILVANO DA ROS currently works as a networking consultant in Toronto and has worked previously as a Systems Engineer for Cisco Systems. He is the author of *Content Networking Fundamentals*, published by Cisco Press. He holds a Bachelor of Computer Science and a Masters of Engineering (in Internetworking) from Dalhousie University. E-mail: sdaros@sympatico.ca

Time to Live

As I read the very fine article entitled “IPv6 Internals” (IPJ Volume 9, No. 3, September 2006), I was prompted to review the history of the *Time to Live* (TTL) as discussed in section 5.3.1 of RFC 1812. Being gray of head, little facts from other eras come quickly to mind. The *Xerox Network Systems* (XNS) Internet Transport on which Novell Netware was based required that no router ever store a packet in queue longer than 6 seconds. Requirements of RFC 791 were also softened in RFC 1812; rather than *requiring* the TTL to be decremented at least once and additionally once per second in queue, that document requires that the TTL be treated as a hop count and—reluctantly—reduces the treatment of TTL as a measure of time to a suggestion.

The reason for the change is the increasing implementation of higher-speed lines. A 1,500-byte datagram occupies 12,000 bits (and an asynchronous line sends those as 15,000 bits), which at any line speed below 19.2 kbps approximates or exceeds 1 second per datagram. Any time there are several datagrams in queue, the last message in the queue is likely to sit for many seconds, a situation that in turn can affect the behavior of TCP and other transports. However, 56-kbps lines became common in the 1980s, and T1 and T3 lines became common in the 1990s. Today, hotels generally offer Ethernet to the room; we have reports of edge networks connected to the Internet at 2.5 Gbps, and residential broadband in Japan and Europe at 26 Mbps per household. At 56 kbps, a standing queue of five messages is required to insert a 1-second delay, and at T1 it requires a queue depth of more than 100 messages. At higher speeds, the issue becomes less important.

That is not to say that multisecond queues are now irrelevant. Although few networks are being built today by concatenating asynchronous links, in developing countries—and on occasion even in hotels here in Santa Barbara, California—people still use dialup lines. In Uganda, some networks that run over the instant messaging capacity of GSM [*Global System for Mobile Communications*], which is to say using 9,600-bps datagrams, have been installed under the supervision of Daniel Stern and UConnect.org (www.uconnect.org). Much of the world still measures *round-trip times* (RTTs) in seconds, and bit rates in tens of kbps.

The TCP research community, one member of which recently asked me whether it was necessary to test TCP capabilities below 2 Mbps, and the IETF community in general would do well to remember that the ubiquity of high bandwidth in Europe, North America, Australia, and Eastern Asia in no sense implies that it is available throughout the world, or that satellite communications and other long-delay pipelines can now be ignored.

—Fred Baker, Cisco Systems
fred@cisco.com

The author responds:

Although to the casual observer the evolution of the Internet seems one of continuously increasing speed and capacity, reality is slightly different. The original ARPANET used 50-kbps modems in the late 1960s. In the next three decennia or so, the maximum bandwidth of a single link increased by a factor 200,000 to 10 Gbps. Interestingly enough, the minimum speed used for Internet connections went down to a little under 10 kbps, so where once the ARPANET had a uniform link speed throughout the network, the difference between the slowest and the fastest links is now six orders of magnitude. The speed difference between a snail and a supersonic fighter jet is only five orders of magnitude. Amazingly, the core protocols of the Internet—IP and TCP—can work across this full speed or bandwidth gamut, although changes were made to TCP to handle both extremes better, most notably in RFCs 1144 and 1323.

Even though I don't think keeping track of the time that packets are stored in buffers, as suggested in the original IPv4 specification, makes much sense even in slow parts of the network, Fred makes a good point: many Internet users still have to deal with speeds at the low end of the range; some of us only occasionally when connecting through a cellular network, others on a more regular basis. Even in Europe and the United States many millions of Internet users connect through dialup. For someone who is used to having always-on multimegabit connectivity, going back to 56 kbps or worse, 9,600 bps can be a bizarre experience. Many of today's Websites are so large that they take minutes to load at this speed. Connecting to my mail server using the *Internet Mail Access Protocol* (IMAP) takes 15 minutes. And one of my favorite relatively new applications, podcasting, becomes completely unusable: downloading a 50-minute audio program takes hours at modem speeds.

And that's all IPv4. It is possible to transport IPv6 packets over the *Point-to-Point Protocol* (PPP) that is used for almost all low-speed connections, but in practice this isn't workable because there are no provisions for receiving a dynamic address from an ISP [*Internet Service Provider*]. With IPv4, Van Jacobson did important work to optimize TCP/IP for low-speed links (RFC 1144). By reducing the *Maximum Transmission Unit* (MTU) of the slow link and compressing the IP and TCP headers, it was possible to achieve good interactive response times by avoiding the situation where a small packet gets stuck between a large packet that may take a second or more to transmit over a slow link while at the same time reducing the header overhead. Although the IETF has later done work on IPv6 header compression, it doesn't look like anyone has bothered to implement these techniques, and the minimum MTU of 1,280 bytes creates significant head-of-line blocking when IPv6 is used over slow links.

Another example where low bandwidth considerations are ignored is the widespread practice of enabling RFC 1323 TCP high-performance extensions for all TCP sessions. RFC 1323 includes two mechanisms: a window scale factor that allows much larger windows in order to attain maximum performance over high-bandwidth links with a long delay, and a timestamp option in the TCP header that allows for much more precise round-trip time estimations. With these options enabled, every TCP segment includes 8 extra bytes with timestamp information. In addition to increasing overhead, the timestamp option introduces an unpredictable value into the TCP header that makes it impossible to use header compression, thereby negating the usefulness of RFC 1144. To add insult to injury, almost no applications allocate enough buffer space to actually use the RFC 1323 mechanisms.

Moral of the story for protocol designers and implementers: spend some time thinking about how your protocol works over slow links. You never know when you'll find yourself behind just such a link.

—Iljitsch van Beijnum
iljitsch@muada.com

Gigabit TCP and MTU Size

I appreciated Geoff Huston's thorough description about the current obstacles and research involving Gigabit TCP (IPJ, Volume 9, No. 3, June 2006). I have already shown the article to many of my colleagues. It appears that Geoff did not address one of the solutions, which is to increase the networkwide *Maximum Transmission Unit* (MTU). In theory that would allow the existing TCP congestion control to handle higher-speed connectivity. Perhaps he did not address the issue because it is infeasible to increase the MTU setting Internetwide, especially with 10-Gigabit Ethernet interfaces sporting a default MTU setting of 1,500 bytes. On the other hand, projects that own their own backbone infrastructure may find increasing the default MTU a feasible approach.

For more information about raising the MTU, please see:
<http://www.psc.edu/~mathis/MTU/>

—Todd Hansen, UCSD/SDSC
tshansen@hpwren.ucsd.edu

The author responds:

Yes, it's true that increasing the size of the packet makes sound sense when the available bandwidth has increased. If the bandwidth increases by one order of magnitude and the packet size is increased by the same amount, then it is theoretically possible to effectively increase the throughput of the system without changing the packet processing load.

Effectively, if you regard the protocol interaction as a time sequence, then a coupling of increased bandwidth and comparably increased packet size preserves the time sequence interaction. Of course, as bandwidth on the network has increased we have not seen a comparable increase in MTU sizes, and today's networks exhibit a wide variety of MTUs and the importance of *Path MTU Discovery*, and coherent transmission of related MTU ICMP [*Internet Control Message Protocol*] messages becomes more critical as a consequence. Although the article concentrated on modifications to the TCP control algorithm, there is no doubting the importance of high-speed TCP senders and receivers using large TCP buffers to maximize the payload throughput potential.

—*Geoff Huston, APNIC*
gih@apnic.net

Drop us a Line!

We welcome any suggestions, comments or questions you may have regarding anything you read in this journal. Send us an e-mail to **ipj@cisco.com**. Also, don't forget to let us know if your delivery address changes. You can use the online subscription system to change your own information by supplying your Subscription ID and e-mail address. The system will then send you an e-mail with a "magic" URL which will allow you to update your database record. If you don't have your Subscription ID or encounter any difficulties, just send us the updated information via e-mail.

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

Book Review

Internet Measurement

Internet Measurement: Infrastructure, Traffic & Applications, by Mark Crovella, Balachander Krishnamurthy, ISBN 0-470-01461-X, Wiley, 2006.

This book is a comprehensive reference guide to about 900 journal, conference, and workshop papers, and RFCs on the important and rapidly advancing field of Internet measurement. Interest in this growing field arises for three major reasons: commercial, social, and technical. Readers need nothing more than a keen interest in a methodical study of the subject matter from either a practical or research perspective to glean something from this book.

Organization

The book is centered on three architectural pillars relevant to measurement: *infrastructure*, *traffic*, and *applications*. Within each of these pillars, the topics are organized into four sections: *properties*, *challenges*, *tools*, and *state-of-the-art*. In the properties section, the authors review metrics that are important to measure in each area. In the challenges section, they discuss various difficulties and limitations that arise when trying to measure the metrics. The tools section covers some of the popular methods and products used to measure these metrics and work around the challenges mentioned previously. The intent is not to provide “user guides” for these tools. The state-of-the-art section presents the latest measurement results about covered properties and metrics, noting that they are subject to relatively fast obsolescence because of the rapidly evolving Internet.

The first three chapters provide background material. The first chapter provides an obligatory introduction to the Internet architecture, including how the “end-to-end” principle has been used for nearly 20 years to guide many design decisions in the Internet. The second chapter provides the analytic background necessary to study the Internet and cast its measurements in quantitative terms. The third chapter examines the nuts and bolts of Internet measurement, addressing the practical topics to consider in designing and implementing them, including the role of time and its sources.

The second part of this book also consists of three chapters, which cover the three pillars in depth. The first chapter defines metrics of interest for measuring the Internet and describes some of the barriers to their measurement, in particular “middleboxes,” *Network Address Translators* (NATs), firewalls, and proxies that deviate from the end-to-end architecture principle, may block *User Datagram Protocol* (UDP) or *Internet Control Message Protocol* (ICMP) packets, or hinder visibility to endpoint IP addresses. The authors next explore various tools and methods for active and passive measurement, estimation, and inference of these metrics.

Readers may wonder why two important metrics are left out—router reliability and high availability—where *Open Shortest Path First* (OSPF) and the *Border Gateway Protocol* (BGP) “Graceful Restart” would be of interest.

The next chapter focuses on traffic properties that are important to understand, measure, and model. The authors examine the challenges in capturing, processing, storing, and managing large volumes of packets and flows, as well as those related to their statistical characterization. Readers engaged in data modeling and performance analysis will benefit from this chapter. The last chapter in this part of the book examines some popular applications: The *Domain Name System* (DNS), Web, and *Peer-to-Peer* (P2P). The authors discuss the shifts in application mix from the 1980s, when FTP was dominant, to the 1990s, when the *Hypertext Transfer Protocol* (HTTP) became dominant, to today, when by most accounts P2P is the dominant Internet protocol. Next, there is a thorough coverage of the what (properties), why (justification), and how to (tools) facets of measurement of the three popular applications, as well as some coverage of online games and streaming media.

The third part of the book covers material that spans multiple areas. Its first chapter deals with anonymization of collected measurement data, which arises because of the need for data sharing, while preserving identity-related, personal-sensitive, or business-sensitive information for applications previously examined. The second chapter provides a short—but important—coverage of the key areas where Internet measurement has played a role in security enforcement. Various attack types and tools to combat them are discussed. The third chapter examines numerous low-level monitoring tools for high-speed traffic capture, as well as an insightful look at the software architecture of two toolsets, *dss* and *Gigascop*, reflecting the experience of one of the authors at AT&T Labs with them. It also reviews some large-scale measurement platforms at the *Cooperative Association for Internet Data Analysis* (CAIDA), the *Réseaux IP Européens* (RIPE) community, and the *High Energy Physics* (HEP) community. The book concludes with a recap of trends, concerns, and emerging questions in Internet measurement.

Synopsis

The authors have blended their academic research and practical experience in Internet measurement and traffic modeling to provide the reader with a structured view to these vast subjects. I would have liked to see a more extensive coverage of *Voice over IP* (VoIP) and its associated performance measurement protocols, *RTP Control Protocol* (RTCP), *RTCP Extended Report* (XR), and RAQMON, given the gradual but inevitable shift of voice traffic from the *Public Switched Telephone Network* (PSTN) to the Internet with *Session Initiation Protocol* (SIP) peering.

Most probably, this book had already been published when the *Federal Communications Committee* (FCC) issued an order in May 2006 for all VoIP service providers to demonstrate compliance with the *Communications Assistance for Law Enforcement Act* (CALEA) wiretapping requirement within a year. This directive represents a notable departure from data anonymization principles covered in the book.

Overall, I consider this book an excellent reference source for diverse research and practical articles published in the field of Internet measurement. I highly recommend it to network planners, engineers, and managers responsible for instrumentation, traffic modeling, or performance analysis.

—*Reza Fardid, Covad Communications*
rfardid@covad.com

Read Any Good Books Lately?

Then why not share your thoughts with the readers of IPJ? We accept reviews of new titles, as well as some of the “networking classics.” In some cases, we may be able to get a publisher to send you a book for review if you don’t have access to it. Contact us at **ipj@cisco.com** for more information.

Bob Braden and Joyce K. Reynolds receive the 2006 Postel Service Award

Bob Braden and Joyce K. Reynolds are this year's recipients of the Internet Society's prestigious *Jonathan B. Postel Service Award*. The award was presented "For their stewardship of the RFC (*Request for Comments*) series that enabled countless others to contribute to the development of the Internet." The presentation was made by Internet pioneer Steve Crocker (a member of this year's Postel award committee and author of the very first RFC) during the 67th meeting of the *Internet Engineering Task Force* (IETF) in San Diego, California.

The award is named after Dr. Jonathan B. Postel to commemorate his extraordinary stewardship exercised over the course of a thirty year career in networking. Between 1971 and 1998, Postel managed, nurtured and transformed the RFC series of notes created by Steve Crocker in 1969. Postel was a founding member of the Internet Architecture Board and the first individual member of the Internet Society, where he also served as a trustee.

"It is a pleasure and an honor for the Internet Society to recognize the contribution of Bob and Joyce to the evolution of the Internet," said Crocker. "Since its humble beginnings, the RFC series has developed into a set of documents widely acknowledged and respected as a cornerstone of the Internet standards process. Bob and Joyce have participated in this evolution for a very long time and have been primarily responsible for ensuring the quality and consistency of the RFCs since Jon's death in 1998."

Joyce K. Reynolds worked closely with Postel, and together with Bob Braden she has been co-leader of the RFC Editor function at the University of Southern California's *Information Sciences Institute* (ISI) since 1998. In this role she performed the final quality control function on most RFC publications. Reynolds has also been a member of the IETF since 1988, and she organized and led the User Services area of the IETF from 1988 to 1998. In her User Services role, she was an international keynote speaker and panelist in over 90 conferences around the world, spreading the word on the Internet.

Bob Braden, who has more than 50 years of experience in the computing field, joined the networking research group at ISI in 1986. Since then, he has been supported by NSF for research concerning NSFnet and the DETER security testbed, and by DARPA for protocol research. Braden came to ISI from UCLA, where he had technical responsibility for attaching the first supercomputer (IBM 360/91) to the ARPAnet, beginning in 1970. Braden was active in the ARPAnet Network Working Group, contributing to the design of the FTP protocol in particular. He also edited the Host Requirements RFCs and co-chaired the RSVP working group.

The Jonathan B. Postel Service Award was established by the *Internet Society* (ISOC) to honor those who, like Postel, have made outstanding contributions in service to the data communications community. The award is focused on sustained and substantial technical contributions, service to the community, and leadership. With respect to leadership, the nominating committee places particular emphasis on candidates who have supported and enabled others in addition to their own specific actions.

Previous recipients of the Postel Award include Jon himself (posthumously and accepted by his mother), Scott Bradner, Daniel Karrenberg, Stephen Wolff, Peter Kirstein, Phill Gross and Jun Murai. The award consists of an engraved crystal globe and \$20,000.

ISOC (<http://www.isoc.org>) is a not-for-profit membership organization founded in 1992 to provide leadership in Internet related standards, education, and policy. With offices in Washington, DC, and Geneva, Switzerland, it is dedicated to ensuring the open development, evolution and use of the Internet for the benefit of people throughout the world. ISOC is the organizational home of the IETF and other Internet-related bodies who together play a critical role in ensuring that the Internet develops in a stable and open manner. For over 14 years ISOC has run international network training programs for developing countries and these have played a vital role in setting up the Internet connections and networks in virtually every country connecting to the Internet during this time.

First Internet Governance Forum Meeting Concludes

The inaugural meeting of the *Internet Governance Forum* (IGF) took place in Athens, Greece from October 30 – November 2, 2006. For more information see: <http://www.intgovforum.org>

The Government of Brazil will host the next IGF meeting. It will take place in Rio de Janeiro November 12 – 15, 2007.

ARIN to Provide 4-Byte AS Numbers

On August 30, 2006, the *American Registry for Internet Numbers* (ARIN) Board of Trustees, based on the recommendation of the Advisory Council and noting that the Internet Resource Policy Evaluation Process had been followed, adopted the following policy proposal: “2005-9: 4-Byte AS Number.”

Per the implementation schedule contained in the policy (*Number Resource Policy Manual* [NRPM] Section 5.1), commencing January 1, 2007, ARIN will process applications that specifically request 32-bit AS Numbers.

For more information see: <http://www.arin.net/registration>

[Ed. See also: “Exploring Autonomous System Numbers,” by Geoff Huston in *The Internet Protocol Journal*, Volume 9, No. 1, March 2006.]

Celebrating the 25th Anniversary of the TCP/IP Internet Standards

Two of the core protocols that define how data is transported over the Internet are now 25 years old. The *Internet Protocol* (IP) and the *Transmission Control Protocol* (TCP), together known as “TCP/IP,” were formally standardized in September 1981 by the publication of RFC 791 and RFC 793.

Vint Cerf and Robert Kahn are widely credited with the design of TCP/IP, and many others involved in the ARPANET project made significant contributions. The core of the documents was RFC 675, published in December 1974 by Cerf together with co-authors Carl Sunshine and Yogen Dalal. The subsequent sequence of documents leading up to RFC 791 and 793 benefited from the participation of many people including Dave Clark, Jon Postel, Bob Braden, Ray Tomlinson, Bill Plummer, and Jim Mathis, as well as other unnamed contributors to the definition and implementation of what became the Internet’s core protocols.

“We can’t yet say that the Internet is mature,” says Brian Carpenter, chair of the IETF, “but it’s a great tribute to its pioneers that the two most basic specifications that were published a quarter of a century ago are still largely valid today. I hope the IP version 6 standard will do as well.”

The *Request For Comments* (RFC) series, which was launched in 1969 by Steve Crocker at UCLA (and edited for many years by the late Jon Postel), continues today as the public archive of the Internet’s fundamental technology. Since 1977 it has been hosted by The University of Southern California’s *Information Sciences Institute* (ISI). ARPA support ended in 1998, at which time ISOC took over providing funding for the publication of Internet standards. More recently, ISOC extended its support to include other areas critical to the open development of Internet standards.

See also:

<http://www.ietf.org/rfc/rfc0791.txt>

<http://www.ietf.org/rfc/rfc0793.txt>

<http://www.isoc.org/standards/tcpip25years>

<http://www.isoc.org/internet/history/brief.shtml>

Call for Papers

The Internet Protocol Journal (IPJ) is published quarterly by Cisco Systems. The journal 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. The journal carries tutorial articles (“What is...?”), as well as implementation/operation articles (“How to...”). It provides readers with technology and standardization updates for all levels of the protocol stack and serves as a forum for discussion of all aspects of internetworking.

Topics include, but are not limited to:

- Access and infrastructure technologies such as: ISDN, Gigabit Ethernet, SONET, ATM, xDSL, cable, fiber optics, satellite, wireless, and dial systems
- 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, trouble-shooting, and mapping
- Value-added systems and services such as: Virtual Private Networks, resource location, caching, client/server systems, distributed systems, network 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, and regulatory topics such as: copyright, content control, content liability, settlement charges, “modem tax,” and trademark disputes in the context of internetworking

In addition to feature-length articles, IPJ will contain standardization updates, overviews of leading and bleeding-edge technologies, book reviews, announcements, opinion columns, and letters to the Editor.

Cisco will pay a stipend of US\$1000 for published, feature-length articles. Author guidelines are available from Ole Jacobsen, the Editor and Publisher of IPJ, reachable via e-mail at ole@cisco.com

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.

The Internet Protocol Journal

Ole J. Jacobsen, Editor and Publisher

Editorial Advisory Board

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

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

David Farber
Distinguished Career Professor of Computer Science and Public Policy
Carnegie Mellon University, USA

Peter Löthberg, Network Architect
Stupi AB, Sweden

Dr. Jun Murai, General Chair Person, WIDE Project
Vice-President, Keio University
Professor, Faculty of Environmental Information
Keio University, Japan

Dr. Deepinder Sidhu, Professor, Computer Science &
Electrical Engineering, University of Maryland, Baltimore County
Director, Maryland Center for Telecommunications Research, USA

Pindar Wong, Chairman and President
Verifi Limited, Hong Kong

*The Internet Protocol Journal is published quarterly by the Chief Technology Office, Cisco Systems, Inc. www.cisco.com
Tel: +1 408 526-4000
E-mail: ipj@cisco.com*

Copyright © 2006 Cisco Systems, Inc. All rights reserved. Cisco, the Cisco logo, and Cisco Systems are trademarks or registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries. All other trademarks mentioned in this document or Website are the property of their respective owners.

Printed in the USA on recycled paper.



The Internet Protocol Journal, Cisco Systems
170 West Tasman Drive, M/S SJ-7/3
San Jose, CA 95134-1706
USA

ADDRESS SERVICE REQUESTED

PRSRT STD U.S. Postage PAID PERMIT No. 5187 SAN JOSE, CA
--