The Internet Protocol Journal

June 2004 Volume 7, Number 2

A Quarterly Technical Publication for Internet and Intranet Professionals

In This Issue

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

From The Editor

The Internet Protocol Journal continues to be a forum for discussion of current and emerging technologies. In this issue, we first look at *content networking*. One can describe the Internet as a system of interconnected devices, but equally as a collection of information, called *content*, that resides on a distributed set of *servers* and is accessed by numerous *clients*. Our first article is by Christophe Deleuze.

Engineers are hard at work planning for an eventual transition to the next version of IP — IPv6. We've published several articles about IPv6 in previous editions. This time, François Donzé describes the automatic address configuration feature of IPv6. Of note is also the increasing global support for IPv6 deployment, (refer to "Fragments" on page 31).

Our final article returns to our recurring theme: adding security to existing Internet protocols. Because many malicious attacks on the Internet are perpetrated by "spoofing" information in one form or another, it makes sense to look at the *Domain Name System* (DNS), a critical component of the Internet infrastructure. Today, it is possible to create systems which provide fake answers to DNS queries. Miek Gieben explains what is being done to address this issue in his tutorial on DNSSEC, the secure version of the DNS protocols.

Please take a moment to renew or update your subscription to this journal. You can do so by visiting **www.cisco.com/ipj** and clicking on the "Subscription Information" link on the left. You will need to supply your subscription ID and e-mail address in order to gain access to your database record. If you have any questions, please send a note to **ipj@cisco.com**.

This is the 25th edition of IPJ. The journal now has more than 32,000 subscribers world-wide, and is available on paper and electronically on our Website in PDF and HTML format. The Website, located at **www.cisco.com/ipj**, contains all our back issues, and will soon offer a cumulative index in ASCII format that will make it easier to find particular articles. As always, we welcome your feedback.

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

Content Networks

by Christophe Deleuze

The Internet is constantly evolving, in both usage patterns and underlying technologies. In the last few years, there has been a growing interest in *content-networking* technologies. Various differing systems can be labelled under this name, but they all share the ability to access objects in a location-independent manner. Doing so implies a shift in the way communications take place on the Internet.

The Classic Internet Model

The Internet protocol stack comprises three layers, shown in Figure 1. The network layer is implemented by IP and various routing protocols. Its job is to bring datagrams hop by hop to their destination host, as identified by the destination IP address. IP is "best effort," meaning that no guarantee is made about the correct delivery of datagrams to the destination host.

The transport layer provides an end-to-end communication service to applications. Currently two services are available: a reliable ordered byte stream transport, implemented by the *Transmission Control Protocol* (TCP), and an unreliable message transport, implemented by the *User Datagram Protocol* (UDP).

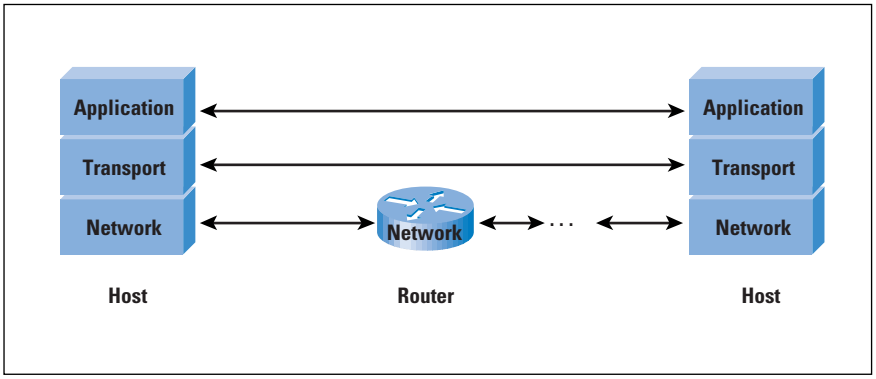


Figure 1: The Three Layers of the Internet Protocol Stack

Above the transport layer lies the application layer, which defines application message formats and communication semantics. The Web uses a client-server application protocol called *Hypertext Transfer Protocol* (HTTP)[10].

A design principle of the Internet architecture is the "end-to-end principle," which states that everything that can be done in the end hosts should be done there, and not in the network itself[8]. That is why IP service is so crude, and transport and application layer protocols are implemented only in the end hosts.

Application objects, such as Web pages, files, etc. (we will simply call those "objects") are identified by URLs. (Actually URLs identify "resources" that can be mapped to different objects called "variants." A variant is identified by a URL and a set of request header values, but in order to keep things simple, we will not consider this in the following.) URLs for Web objects have the form **`http://host:port/path`**. This means that the server application lives on a host with *hostname* (or possibly IP address) on port *N* (with default value of 80), and knows the object under the name *path*. Thus URLs, as their name implies, tell where the object can be found. To access such an object, a TCP connection is open to the server running on the specified host and port and the object named path is requested.

Content Networks

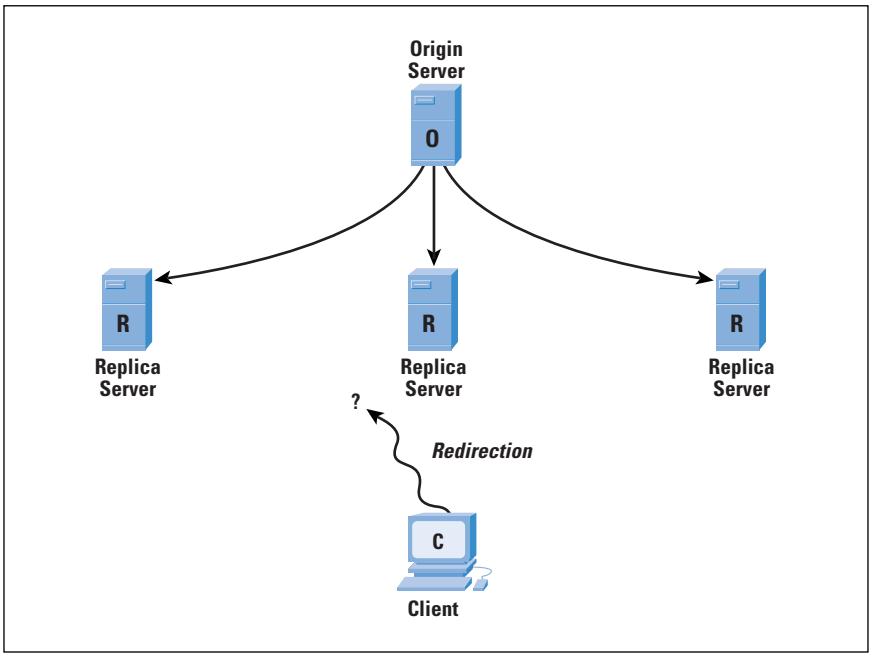
Content networks aim to provide location-independent access to an object, most commonly because they handle some kind of (possibly dynamic) replication of the objects. By design, URLs are not suited to identify objects available on several places on the network.

Handling such replication and location-independent access usually involves breaking the end-to-end principle at some point. Communication is no more managed end to end: intermediate network elements operating at the application layer (whose most common types are "proxies") are involved in the communication. (Content networks are not the only case where this principle is violated.)

In the same way that IP routers relay IP datagrams (that is, network layer protocol data units), routing them to their destination according to network layer information, those application layer nodes relay application messages, using application layer information (such as content URLs) to decide where to send them. This is often called *content routing*.

So the goal of a content network is to manage replication, handling two different tasks: *distribution* ensures the copying and synchronization of the instances of an object from an *origin server* to various *replica servers,* and *redirection* allows users to find on instance of the object (possibly the one closest to them.) (By "replica," we mean any server of any kind other than the origin that is able to serve an instance of the object. This term often has a narrower meaning, not applying, for example, to caching proxies.) This is illustrated in Figure 2.

Figure 2: Elements of a Content Network



Various kinds of content networks exist, differing in the extent to which they handle these tasks and in the mechanisms they use to do so. There are many possible ways to classify them. In this article, we use a classification based on who owns and administers the content network. We thus find three categories: content networks owned by network operators, content providers, and users.

Network Operators' Content Networks

Network operators (also called *Internet Service Providers*, or ISPs) often install caching proxies in order to save bandwidth[11]. Clients send their requests for objects to the proxy instead of the origin server. The proxy keeps copies of popular objects in its cache and can answer directly if it has the requested object in cache. (To be precise, such a caching proxy does not cache objects, but server responses.) If this is not the case, it gets the object from the origin server, possibly stores a copy in its cache, and sends it back to the client.

This caching proxy scheme can be used recursively, making those proxies contact parent proxies for requests they cannot fulfill from their local store. Such hierarchies of caching proxies actually lead to constructing content-distribution trees. This makes sense if the network topology is tree-like, although there are some drawbacks, including the fact that less popular objects (those not found in any cache) experience delays, which increase with the depth of the tree. Another problem is with origin servers whose closest tree node is not the root.

The Squid caching proxy[5] can be configured to choose the parent proxy to query for a request based on the domain name of the requested URL (or to get the object directly for the origin server). This allows setting up multiple logical trees on the set of proxies, a limited form of content routing.

Such manual configuration is cumbersome, especially because domain names do not necessarily (and actually most do not) match network topology. Thus the administrator must know where origin servers are in the network to use this feature effectively.

The same effects can be achieved, to some extent, in an automatic and dynamic fashion using ICP, the *Internet Cache Protocol* [16, 15]. ICP allows a mesh of caching proxies to cooperate by exchanging hints about the objects they have in cache, so that a proxy missing an object can find a close proxy that has it. One advanced feature of ICP allows you to select among a mesh of proxies the one that has the smallest *Round-Trip Time* (RTT) to the origin server.

One design flaw of ICP is that it identifies objects with URLs. We mentioned previously that a URL actually identifies a resource that can be mapped to several different objects called variants. Thus information provided by ICP is of little use for resources that have multiple variants. However, in practice most resources have only one variant, so this weakness does little harm.

Users normally configure their browsers to use a proxy, but automatic configuration is sometimes possible. Multiple proxies can be used by a client with protocols such as the *Cache Array Routing Protocol* (CARP)[14]. To avoid configuration issues, a common trend is for ISPs to deploy *interception proxies*. Network elements such as routers running the Cisco *Web Cache Communication Protocol* (WCCP)[6,7] redirect HTTP traffic to the proxy, without the users knowing. The proxy then answers client requests pretending to be the origin server. This poses numerous problems, as discussed in [12].

Caching proxies have limited support for ensuring object consistency. Either the origin server gives an expiration date or the proxy estimates the object lifetime based on the last modification time, using an heuristic known as *adaptive TTL* (time to live).

Content Providers' Content Networks

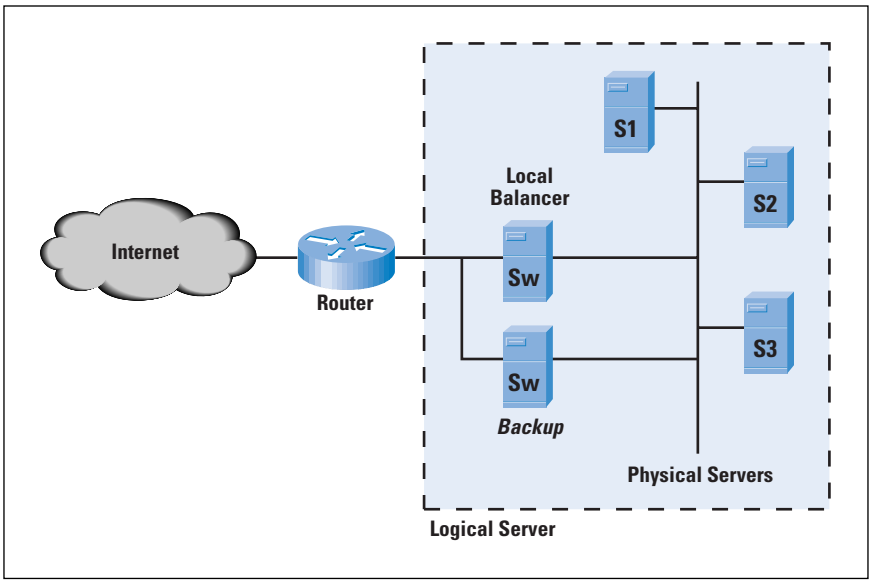
Contrary to ISPs whose main goal is to save bandwidth, content providers want to make their content widely available to users, while staying in control of the delivery (including ensuring that users are not delivered stale objects). We can again roughly classify such content networks in three subcategories:

- *Server farms:* Locally deployed content networks aimed at providing more delivery capacity and high availability of content
- *Mirror sites:* Distributed content networks making content available in different places, thus allowing users to get the content from a close mirror
- *Content-Delivery Networks* (CDNs): Mutualized content networks operated for the benefit of numerous content providers, allowing them to get their content replicated to a large number of servers around the world at lower cost.

Server Farms

Server farms are made of a load-balancing device (we will call it a *switch*) receiving client requests and dispatching them to a series of servers (the *physical* servers). The whole system appears to the outside world as a single *logical* server. The goal of a server farm is to provide scalable and highly available service. The switch monitors the physical servers and uses various load metrics in its dispatching algorithm. Because the switch is a single point of failure, a second switch is usually set up in a hot failover standby mode, as shown in Figure 3.

Figure 3: Server Farm



Some switches are called *Layer 4 switches* (4 is the number of the transport layer in the *OSI Reference Model*), meaning they look at network and transport layer information in the first packet of a connection to decide to which physical server the incoming connection should be handed. They establish a state associating the connection with the chosen physical server and use it to relay all packets of the connection. The exact way the packets are sent to the physical servers varies. It usually involves some form of manipulation of IP and TCP headers in the packets (like *Network Address Translation* [NAT] does) or IP encapsulation. These tricks are not necessary if all the physical servers live on the same LAN.

More complex *Layer 7 switches* (7 is the number of the application layer in the OSI Reference Model) look at application layer information, such as URL and HTTP request headers. They are sometimes called *content switches*. On a TCP connection, application data is available only after the connection has been opened. A proxy application on the switch must thus accept the connection from the client, receive the request, and then open another connection with the selected physical server and forward the request. When the response comes back, it must copy the bytes from the server connection to the client connection.

Such a splice of TCP connections consumes much more resources in the switch than the simple packet manipulation occurring in Layer 4 switches. Bytes arrive at one connection and are handed to the proxy application, which copies them to the other connection—all of this involving multiple kernel mode-to-user mode memory copy operations and CPU context switches. Various optimizations are implemented in commercial products. The simplest one is to put the splice in kernel mode. After it has sent the request to the physical server, the proxy application asks the kernel to splice the two connections, and forgets about them. Bytes are then copied between the connections directly by the kernel, instead of being given to the proxy application and back to the kernel.

It is even possible to actually merge the two TCP connections, that is, simply relay packets at the network layer to establish a direct TCP connection between the client and the physical server. This requires manipulating TCP sequence numbers (in addition to addresses and ports) when relaying packets, because the two connections will not have used the same initial sequence numbers. This can be much more complex (or even impossible) to perform if TCP options differ in the two connections.

Mirror Sites

In such a content network, a set of servers are installed in various places in the Internet, and they are defined as *mirrors* of the master server. Synchronization is most commonly performed periodically (often every night), using FTP or specialized tools such as *rsync*[4].

Redirection is performed by the users themselves for most sites. The master server, to which the user initially connects, displays a list of mirrors with geographic information and suggests that users choose a mirror close to themselves, by simply clicking on the associated link.

This process can be automated sometimes. One trick is to store the user's choice in a *cookie,* such that the next time the user connects to the master site, the information provided in the cookie will be used to issue an *HTTP redirect* (an HTTP server response asking the client to retry the request on a new URL) to the previously selected site.

Other schemes involve trying to find which of the mirrors is closest to the user based on information provided in the user request (such as preferred language) or indicated by network metrics. Such schemes were not very common for simple mirror sites, but today many commercial products allowing for this kind of "global load balancing" are available.

In any case (except if redirection is automatic and *Domain Name System* [DNS] based—this is discussed in the next section) the URLs of objects change across mirrors.

CDNs

Most content providers cannot afford to own numerous mirror sites. Having servers in different places around the world costs lots of money. Operators of CDNs own a large replication infrastructure (Akamai, the biggest one, claims to have 15,000 servers) and get paid by content providers to distribute their content. By mutualizing the infrastructure, CDNs are able to provide very large reach at affordable costs.

CDN servers do not store entire sites of all the content providers, but rather cache a subset according to local client demand. Such servers are called *surrogates*. They manage their disk store like proxies do, and serve content to clients like mirrors do (that is, contrary to proxies, they act as the authoritative source for the content they deliver).

Because the number of surrogates can be so large, and because of the argument that "no user configuration is necessary," CDNs typically include complex redirection systems that allow them to perform automatic and user-transparent redirection to the selected surrogate. The selection is based on information about surrogate loads and on network metrics collected by various ways such as routing protocol information, RTTs measured by network probes, etc. The client is made to connect to the selected surrogate either by sending it an HTTP redirect message, or by using the DNS system: when the client tries to resolve the host name of the URL in an IP address to connect to, it is given back the address of the selected surrogate instead. Using the DNS ensures that the URL is the same for all object copies. In this case, CDNs actually turn URLs into location-independent identifiers.

In addition to proxy-like on-demand distribution, content can also be "pushed" in surrogates in a proactive way. Synchronization can be performed by sending invalidation messages (or updated objects) to surrogates.

CDN principles are also being used in private intranets for building *Enterprise CDNs* (ECDNs).

Users' Content Networks

User-operated content networks are better known as *Peer-to-Peer* (P2P) networks. In these networks, the costly replication infrastructure of other content networks is replaced by the users, who make some of their storage and processing capacities available to the P2P network. Thus, no big money is needed, and no one has control over the content network.

One advantage P2P networks have over other content networks is that they are usually built as overlay networks and do not strive for transparent integration with the current Web. Thus they are free to build new distribution (some of them allow downloading files from multiple servers in parallel) and redirection mechanisms from scratch, and even to use their own namespace instead of being stuck with HTTP and URLs.

P2P networks basically handle the distribution part of replication in a straightforward way: the more popular an object is, the more users will have a copy of it, thus the more copies of the object will be available on the network. More complex mechanisms can be involved, but this is the basic idea.

The redirection part of replication is more problematic with most current P2P networks. It can be handled by a central directory as in *Napster:* every user first connects to a central server, updates the directory for locally available objects, and then looks up the directory for locations of objects the user wants to access. Of course, such a central directory poses a major scalability and robustness problem.

Gnutella and *Freenet,* for example, use a distributed searching strategy instead of a centralized directory. A node queries neighbors that themselves query neighbors, and so on until either one node with the requested object is found or a limit on the resources consumed by the search has been hit. Although there is no single point of failure, such a scheme is no more scalable that the central directory. It seems easy to perform denial-of-service attacks by flooding the network with requests. Additionally, you can never be sure you have found the object even if someone has it.

These examples are primitive and have serious flaws, but much research work is being performed on this topic; refer to [13] for a summary.

Although they are currently used mainly for very specific file-sharing applications, P2P networks do provide new and valuable concepts and techniques. For example, *Edge Delivery Network* is a commercially available software-based ECDN inspired by Freenet. Various projects use a *scatter/gather* distribution scheme, useful for very large files: users download several file chunks in parallel from other currently downloading users, thus refraining from using server resources for long periods of time.

Some projects attempt to integrate P2P principles in the current Web architecture and protocols. Examples are [3] and [1].

Conclusion

Current networks have been designed and deployed as ad-hoc solutions of specific problems occurring in the current architecture of the network. Caching proxies lack proper means to ensure consistency, but CDNs tricks the DNS to turn URLs into location-independent identifiers. P2P networks are mostly limited to file-sharing applications.

Content networks implement mechanisms to ensure distribution of content to various locations, and redirection of users to a close copy. They often have to break the end-to-end principle in order to do so, mainly because current protocols assume each object is available in only one statically defined location.

Probably the first step in building efficient distribution and redirection mechanisms for providing an effective replication architecture is the setting up of a proper replication-aware namespace. Applications would pass an object name to a name resolution service and be given back one or more locations for this object. The need for such a location-independent namespace was anticipated a long time ago. URLs are actually defined as one kind of *Uniform Resource Identifier* (URI), another one being *Uniform Resource Names* (URNs) intended to provide such namespaces. A URN IETF working group [2] has been active for a long time, and recently published a set of RFCs (3401 to 3406).

Work on the topic of content networking has also been performed by the now closed *Web Replication and Caching* (WREC) IETF working group, which issued a taxonomy in [9]. An interesting survey of current work on advanced content networks is [13].

References

[1] BitTorrent: **`http://bitconjurer.org/BitTorrent/`**

[2] IETF URN Working Group: **`http://www.ietf.org/html.charters/urn-charter.html`**

[3] Open Content Network: **`http://www.open-content.net`**

[4] Rsync: **`http://rsync.samba.org`**

[5] Squid Internet Object Cache: **`http://www.squid-cache.org`**

[6] M. Cieslak and D. Forster, "Web cache coordination protocol v1.0," Expired Internet Draft, **`draft-forster-wrec-wccp-v1-00.txt`**, Cisco Systems, July 2000.

[7] M. Cieslak, D. Forster, G. Tiwana, and R. Wilson, "Web cache Coordination Protocol v2.0," Expired Internet Draft, **`draft-wilson-wrec-wccp-v2-00.txt`**, Cisco Systems, July 2000.

[8] David D. Clark, "The design philosophy of the DARPA Internet protocols," *Computer Communication Review,* Volume 18, No. 4, August 1988. Originally published in Proceedings of SIGCOMM'88.

[9] Ian Cooper, Ingrid Melve, and Gary Tomlinson, "Internet Web Replication and Caching Taxonomy," RFC 3040, January 2001.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol — HTTP/1.1," RFC 2616, June 1999.

[11] Geoff Huston, "Web Caching," *The Internet Protocol Journal,* Volume 2, No. 3, September 1999.

[12] Geoff Huston, "The Middleware Muddle," *The Internet Protocol Journal,* Volume 4, No. 2, June 2001.

[13] H. T. Kung and C. H. Wu, "Content Networks: Taxonomy and New Approaches," 2002. **http://www.eecs.harvard.edu/htk/publication/2002-santa-fe-kung-wu.pdf**.

[14] Vinod Valloppillil and Keith W. Ross, "Cache array routing protocol v1.0," Expired Internet Draft, **draft-vinod-carp-v1-03.txt**, February 1998.

[15] D. Wessels and K. Claffy, "Application of Internet Cache Protocol (ICP), Version 2," RFC 2187, September 1997.

[16] D. Wessels and K. Claffy, "Internet Cache Protocol (ICP), Version 2," RFC 2186, September 1997.

CHRISTOPHE DELEUZE holds a Ph.D. degree in computer science from Université Pierre et Marie Curie, Paris. He worked on quality-of-service architectures in packet networks, and then spent three years in a start-up company designing CDN systems. He has also been a teacher. E-mail: **christophe.deleuze@free.fr**

IPv6 Address Autoconfiguration

by François Donzé, HP

Since 1993 the *Dynamic Host Configuration Protocol* (DHCP)[1] has allowed systems to obtain an IPv4 address as well as other information such as the default router or *Domain Name System* (DNS) server. A similar protocol called DHCPv6[2] has been published for IPv6, the next version of the IP protocol. However, IPv6 also has a stateless autoconfiguration protocol[3], which has no equivalent in IPv4.

DHCP and DHCPv6 are known as *stateful* protocols because they maintain tables within dedicated servers. However, the stateless autoconfiguration protocol does not need any server or relay because there is no state to maintain.

This article explains the IPv6 stateless autoconfiguration mechanism and depicts its different phases.

Scope of IPv6 Addresses

Every IPv6 system (other than routers) is able to build its own unicast global address. A *unicast* address refers to a unique interface. A packet sent to such an address is treated by the corresponding interface—and *only* by this interface. This type of address is directly opposed to the multicast address type that designates a group of interfaces. Most of this article deals with unicast addresses. For simplicity, we will omit the unicast qualifier when there is no ambiguity.

Address types have well-defined destination scopes: *global, site-local* and *link-local*. Packets with a link-local destination must stay on the link where they have been generated. Routers that could forward them to other links are not allowed to do so because there has been no verification of uniqueness outside the context of the origin link.

Similarly, border-site routers cannot forward packets containing site-local addresses to other sites or other organizations. The IETF is currently working on a way to remove or replace site-local addresses. Hence, this article will refrain from any other reference to this address type. Finally, a global address has an unlimited scope on the worldwide Internet. In other words, packets with global source and destination addresses are routed to their target destination by the routers on the Internet. A fundamental feature of IPv6 is that all *Network Interface Cards* (NICs) can be associated with several addresses.

At minimum, a NIC is associated with a single link-local address. But in the most common case a NIC is assigned a link-local and at least one global address. The following command displays the configuration of network interface **`eth1`** on a Red Hat system. This interface is associated with two IPv6 addresses. One of them starts with **`fe80::`** and the other with **`3ffe:`**. The scope of the first one is the link and the second has a global scope.

```
root# ip address list eth1
3: eth0: <BROADCAST,MULTICAST,UP mtu 1500 qdisc pfifo_fast qlen 100
link/ether 00:0c:29:c2:52:ff brd ff:ff:ff:ff:ff:ff
inet6 fe80::20c:29ff:fec2:52ff/10 scope link
inet6 3ffe:1200:4260:f:20c:29ff:fec2:52ff/64 scope global
```

Creation of the Link-Local Address

An IPv6 address is 128 bits long. It has two parts: a *subnet prefix* representing the network to which the interface is connected and a *local identifier*, sometime called token. In the simple case of an Ethernet medium, this identifier is usually derived from the EUI-48 *Media Access Control* (MAC) address using an algorithm described later in this article. The subnet prefix is a fixed 64-bit length for all current definitions. Because IPv4 manual configuration is a well-known pain, one could hardly imagine manipulating IPv6 addresses that are four times longer. Moreover, a DHCP server is not always necessary or desired; in the case of a remote control finding the DVD player, a DHCP environment is not always suitable.

Because the prefix length is fixed and well-known, during the initialization phase of IPv6 NICs, the system builds automatically a link-local address. After a uniqueness verification, this system can communicate with other IPv6 hosts on that link without any other manual operation.

For a system connected to an Ethernet link, the build and the validation of the link-local address is the following:

1. An identifier is generated, supposedly unique on the link.
2. A tentative address is built.
3. The uniqueness of this address on the link is verified.
4. If unique, the address from phase 2 is assigned to the interface. If not unique, a manual operation is necessary.

Although a local policy can decide to use a specific token, the most common method to obtain a unique identifier on an Ethernet link is by using the EUI-48 MAC address and applying the modified IEEE EUI-64 standard algorithm. A MAC address (IEEE 802) is 48 bits long. The space for the local identifier in an IPv6 address is 64 bits. The EUI-64 standard explains how to stretch IEEE 802 addresses from 48 to 64 bits, by inserting the 16 bits **`0xFFFE`** at the 24th bit of the IEEE 802.

By doing so, transforming MAC address **`00-0C-29-C2-52-FF`** using the EUI-64 standards leads to **`00-0C-29-FF-FE-C2-52-FF`**. Using IPv6 notation, we get **`000C:29FF:FEC2:52FF`**. Recall that the notation of IPv6 addresses requires 16-bit pieces to be separated by the character "**`:`**". Then, it is necessary (RFC 3513) to invert the universal bit ("u" bit) in the 6th position of the first octet. Thus the result is:
`020c:29ff:fec2:52ff`.

Universal uniqueness of IEEE 802 and EUI-64 is given by a "u" bit set to 0. This global uniqueness is assured by IEEE, which delivers those addresses for the entire planet. Inverting the "u" bit allows ignoring it for short values in the manual configuration case, as explained in paragraph 2.5.1 of RFC 3513[4].

The second phase of creating automatically a link-local address is to prepend the well-known prefix **`fe80::/64`** to the identifier resulting from phase one. In our case we obtain **`fe80::20c:29ff:fec2:52ff`**. This address is associated with the interface and tagged "tentative." Before final association, it is necessary to verify its uniqueness on the link. The probability of having a duplicate address on the same link is not null, because it is recognized that some vendors have shipped batches of cards with the same MAC addresses.

This is the goal of the third phase, called *Duplicate Address Detection* (DAD). The system sends ICMPv6 packets on the link where this detection has to occur. Those packets contain *Neighbor Solicitation* messages. Their source address is the undefined address "`::`" and the target address is the tentative address. A node already using this tentative address replies with a *Neighbor Advertisement* message. In that case, the address cannot be assigned to the interface. If there is no response, it is assumed that the address is unique and can be assigned to the interface.

We are reaching the last step of the automatic generation of a link-local address. This phase removes the "tentative" tag and formally assigns the address to the network interface. The system can now communicate with its neighbors on the link.

Global Prefixes

In order to exchange information with arbitrary systems on the global Internet, it is necessary to obtain a global prefix. Usually (but not necessarily), the identifier built during the first step of the automatic link-local autoconfiguration process is appended to this global prefix.

However, before assigning this global address, the system verifies again that no duplicate address exists on the link. DAD is performed for all addresses before they are assigned to an interface, because uniqueness in one prefix does not automatically assure uniqueness in any other available prefixes.

Generally, global prefixes are distributed to the companies or to end users by *Internet Service Providers* (ISPs).

Random Identifiers

The EUI-48-to-EUI-64 transform process is attractive because it is simple to implement. However, it generates a privacy problem. Global unicast as well as link-local addresses may be built with an identifier derived from the MAC address. A Website tracking where a node frequently attaches can collect private information such as the time spent by employees in the enterprise or at home.

Because a MAC address follows the interface it is attached to, the identifier of an IPv6 address does not change with the physical location of the Internet connection. Hence it is possible to trace the movements of a portable laptop or *Personal Digital Assistant* (PDA) or other mobile IPv6 device.

RFC 3041[5] allows the generation of a *random* identifier with a limited lifetime. Because IPv6 architecture permits multiple suffixes per interface, a single network interface is assigned two global addresses, one derived from the MAC address and one from a random identifier. A typical policy for use of these two addresses would be to keep the MAC-derived global address for inbound connections and the random address for outbound connections. A reason for not using it for inbound connections is the need to update the DNS just as frequently as it is changes.

Such a system, with two different global addresses—one of which changes regularly—becomes very difficult to trace.

By default, Microsoft enables this feature on Windows XP and Windows Server 2003. The random-identifier-based global addresses of Microsoft systems have the address type "temporary." EUI-64 global addresses have type "public." Those types as well as other information can be displayed in a **`cmd.exe`** DOS-box with the command line:

```
netsh interface ipv6 show address
```

IPv6 Routers

By definition, a router is a node that forwards IP packets not explicitly addressed to it. IPv6 routers are certainly compliant with this definition but, in addition, they regularly advertise information on the links to which they are connected—provided they are configured to do so. These advertisements are *Internet Control Message Protocol Version 6* (ICMPv6) *Router Advertisement* (RA) messages, sent to the multicast group **`ff02::1`**. All the systems on a link must belong to this group, and nodes configured for autoconfiguration, among other things, analyze the option(s) of those messages. They might contain any routing prefix(es) for this segment.

Router Solicitation

Upon reception of one of those RA messages and according to local algorithm policy, an autoconfiguring node not already configured with the corresponding global address will prepend the advertised prefix to the unique identifier built previously.

However, the advertisement frequency, which is usually about ten seconds or more, may seem too long for the end user. In order to reduce this potential wait time, nodes can send *Router Solicitation* (RS) messages to all the routers on the link. Nodes that have not configured an address yet use the unspecified address "`::`". In response, the routers must answer immediately with a RA message containing a global prefix. This router solicitation corresponds to ICMPv6 messages of type RS, sent to the all-router multicast group: **`ff02::2`**. All routers on the link must join this group.

Thus, a node soliciting on-link routers in such a way is able to extract a prefix and build its global address. Note that this method using an advertised prefix is possible only for end nodes. Today IPv6 routers are usually manually configured. The reason is obvious: a stateless automatic configuration requires the advertisement of a prefix. This prefix is sent by a router. The router sending the prefix must be fully configured to do so. The easiest way to break this seemingly unsolvable problem is to manually configure IPv6 routers. However, some automatic methods are being developed[6].

Conclusion

Stateless address autoconfiguration is a new concept with IPv6. It gives an intermediate alternative between a purely manual configuration and stateful autoconfiguration. In addition to ease of use with no dedicated server or relay, this mechanism removes problems that have not been discussed here, such as the mismatch between the DCHP server and the router (prefix topology) or the IPv4 need to readdress subnets that have outgrown their prefix. Moreover, automatic renumbering (prefix change) is also possible on nodes using stateless autoconfiguration.

References

RFCs can be found at **`http://www.ietf.org/rfc/`**

[1] Droms, R., "Dynamic Host Configuration Protocol," RFC 1531, October 1993.

[2] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., Carney, M., "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," RFC 3315, July 2003.

[3] Thomson, S., Narten, T., "IPv6 Stateless Address Autoconfiguration," RFC 2462, December 1998.

[4] Hinden, R., Deering, S., "Internet Protocol Version 6 (IPv6) Addressing Architecture," RFC 3513, April 2003.

[5] Narten, T., Draves, R., "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," RFC 3041, January 2001.

[6] Prefix delegation: **`http://www.ietf.org/internet-drafts/draft-ietf-dhc-dhcpv6-opt-prefix-delegation-06.txt`**

FRANÇOIS DONZÉ studied at the University of Utah in Salt Lake City. In 1989 he joined Digital Equipment Corporation as a UNIX and network teacher. He is now a technical consultant at HP, based in Sophia-Antipolis, France, promoting IPv6 and other leading-edge technologies. The author of several internal articles, he also publishes in French magazines. E-mail: **`francois.donze@hp.com`**

DNSSEC: The Protocol, Deployment, and a Bit of Development

by Miek Gieben, NLnet Labs

> "One Key to rule them all,
> one Key to find them,
> one Key to bring them all
> and in the Resolver bind them."
> —Modified from *Lord of the Rings*.

The *Domain Name System* (DNS) (RFCs 1034 and 1035) is a highly successful and critical part of the Internet infrastructure. Without it the Internet would not function. It is a globally distributed database, whose performance critically depends on the use of caching.

Unfortunately the current DNS is vulnerable to so-called *spoofing attacks* whereby an attacker can fool a cache into accepting false DNS data. Also various man-in-the-middle attacks are possible. The *Domain Name System Security Extension* (DNSSEC) is not designed to end these attacks, but to make them detectable by the end user. Or more technically correct: detectable by the security-aware *resolver* doing the work for the end user. This saves users from doing online banking on the wrong server even if a secured connection is used and the address in the browser looks correct.

DNSSEC is about protecting the end user from DNS protocol attacks. In order to make it work, zone owners (such as **`.com`**, **`.net`**, **`.nl`**, etc.) need to deploy DNSSEC in their zones. End users then need to update their resolvers to become security-aware (that is, understand DNSSEC) and add some trusted keys. These keys are called *anchored keys;* they are configured in the resolver and cannot be changed or updated very easily. If this is all configured, the end user will (finally) be able to detect attacks.

DNSSEC, as defined in (hopefully soon-to-be-obsoleted) RFC 2535, adds data origin authentication and data integrity protection to the DNS. The *Public Key Infrastructure* (PKI) in DNSSEC may be used as a means of public key distribution, which may be used by other protocols. *IP Security* (IPSec) and the *Secure Shell* (SSH) protocol, for example, are already considering the use of DNSSEC to carry their keying material.

In the course of early-deployment experiments carried out by various organizations, it became evident that RFC 2535 introduced an administrative key-handling and maintenance nightmare. This in turn would mean the DNSSEC deployment would never start (or be successful, for that matter).

The IETF DNSEXT working group decided to fix this problem, and to incorporate all drafts and RFCs written since RFC 2535 into a new DNSSEC specification.

This (still ongoing) effort became known as the *RFC 2535bis* DNSSEC specification. This work has resulted in three drafts, each handling a specific part of the new specification. These drafts follow:

1. dnssec-intro[1] provides an introduction into DNSSEC.
2. dnssec-records[2] introduces the new records for use in DNSSEC.
3. dnssec-protocol[3] is the main document, which details all the protocol changes.

The documents are now almost ready (July 2004) to be submitted to the *Internet Engineering Steering Group* (IESG) for review. It is hoped that soon after this is done the drafts will become RFCs. It could be that 2004 will be the year of DNSSEC.

In this article I use the terms *domain* and *zone*. These are important concepts in the DNS and in DNSSEC. The difference between a zone and a domain is worth highlighting. A domain is a part of the DNS tree. A zone contains the domain names and data that that domain contains *except* for the domain names and data that are delegated elsewhere. Also refer to [4].

Consider, for instance, the **`.com`** *domain,* which includes everything that ends in **`.com`**. **`CNN.com`** is in the **`.com`** domain. The **`.com`** *zone,* however, is the entity handled by VeriSign.

One other important concept in DNS is the *Resource Record* (RR) and the *Resource Record Set* (RRset). An RR in DNS is, for instance:

```
www.example.org.  IN   A    127.0.0.1
```

... where **`www.example.org`** is the "ownername" or "name." **`IN`** is the class (IN stands for Internet). **`A 127.0.0.1`** is the type (together with its rdata). **`A`** stands for "address." This 3-tuple (name, class, type) together make up the resource record. RRset are all the RRs that have an identical name, class and type. Only the rdata is different. Thus:

```
www.example.org.  IN   A    127.0.0.1
www.example.org.  IN   A    192.168.0.1
```

... together form a RRset, but:

```
www.example.org.  IN   A    127.0.0.1
www.example.org.  IN   MX   mail.example.org.
```

... do not (their type is different). In the DNS an RRset is considered *atomic* and the smallest data item. In DNSSEC each RRset gets a signature.

What Is DNSSEC?

DNSSEC adds data origin authentication and data integrity to the DNS. To achieve this, DNSSEC uses public key cryptography; (almost) everything in DNSSEC is digitally signed.

Public key cryptography uses a single key split in two parts: a private and a public component. The *private* component, also known as the *private key*, must be kept secret. The *public* component (the *public key*) can be made public. Both these keys can be used for cryptographic operations, albeit with different goals.

If a message is scrambled with the public key, it can be decrypted only with the private key. This is called *encryption* of the message and it ensures that only the holder of the private key can read the original message. When the private key is used to scramble a message, everybody can use the available public key to decipher the message. This last operation is called (digitally) *signing* a message (for increased speed usually a hash of the message is signed). In this case you know where the message comes from (*authenticated data origin* in cryptographic jargon). An added benefit of signing messages is that when the data is mangled during transport the signature is no longer valid. This last property is called *authenticated data integrity*. A more lengthy introduction on public key cryptography can be found at [10]. In DNSSEC only digital signatures (signing) are used, and nothing is ever encrypted.

For every secure zone there must be a public key in the DNS for use by DNSSEC. Each zone administrator generates a key to be used for securing a zone. The private key is (of course) kept private and is used in the "signing process" to create the signatures. The public key is published in DNSSEC as a DNSKEY record, which is the zone key. The generated signatures are published as RRSIG records.

If RRsets in DNSSEC do not have a valid signature, they are labeled bogus by the resolver. Bogus data should not be trusted, because probably somebody is trying to conduct a spoof attack. DNSSEC further distinguishes between:

- Verifiable secure—The data has signatures that are valid.
- Verifiable unsecure*—The data has no signatures.
- Old-style DNS—A non-DNSSEC lookup is done.

* Yes, Unsecure. This word has somehow evolved from "insecure."

Verifiable secure data is data that has valid signatures, and the key used to create those signatures is trusted (anchored in the resolver). Verifiable unsecure data is data for which we know for sure we do not need to do signature validation. Old-style DNS is the current (insecure) method of getting DNS data.

The signing of data in DNSSEC is comparable to the *Gnu Privacy Guard* (GPG) signing of e-mail. If I trust a public key from someone, I can use that key to verify the GPG signature and authenticate the origin of the e-mail.

The problem with both DNSSEC and GPG lies in the "...If I trust the public key from someone." GPG solves this with public key servers, key signing parties at various events and thus the creation of a web of trust. For DNSSEC such solutions are impractical. DNSSEC uses a different, but very elegant mechanism called the *chain of trust*.

The chain of trust makes it possible to start with a root zone key, the highest possible key in the DNS tree, and following cryptographic pointers to lower zones. Each pointer is validated with the previous validated zone key. (The root key is the key used in the root zone of the Internet; it is the key used in the . (dot) zone. It could take a while before the root is signed.)

By using this mechanism only the root key is needed to validate *all* DNSSEC keys on the Internet. With these DNSSEC keys the DNS data in each zone can then be validated. So, unlike GPG, we need to distribute only one key. This can be done by publishing it on the World Wide Web or in a newspaper or putting an ad on TV, etc.

One of the current items in the DNSSEC community is to outline procedures and guidelines on how to update this root and other keys.

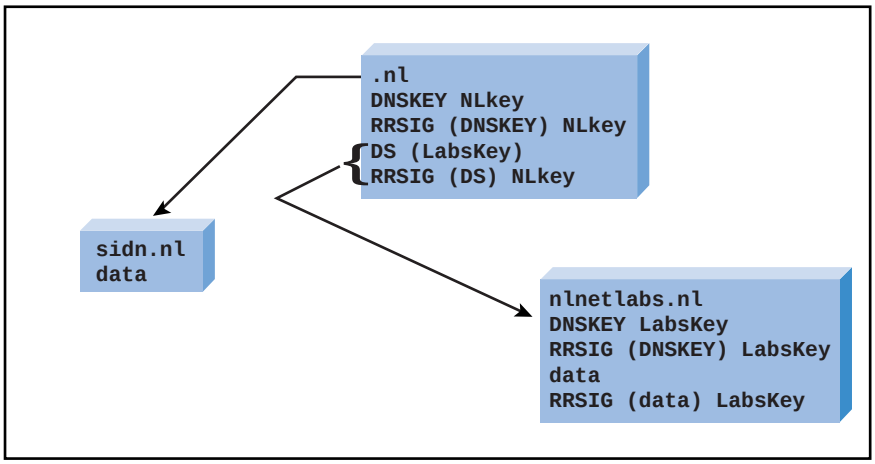
Chain of Trust

To start securely resolving in DNSSEC, a root key must be anchored in the resolver at your local computer or nameserver. Only when a resolver knows and trusts a zone key can it validate the signatures belonging to that zone. Because of the chain of trust, a resolver has to carry only a few zone keys to be able to validate DNSSEC data on the Internet.

The chain of trust works by following "secured pointers," which are called *secured delegation* in DNSSEC. A special, new record called the *Delegation Signer* (DS) record delegates trust from a parental key to a child's zone key.

The DS record holds a hash (*Secure Hash Algorithm 1* [SHA-1]) of a child's zone key. This DS record is signed with the zone key from the parent. By checking the signature of the DS record, a resolver can validate the hash of the child's zone key. If this is successful, the resolver can compare this (validated) hash with the (yet-to-be-validated) hash of the child's zone key. If these two hashes match, the child's real zone key can be used for validation of data in the child's zone. Note: by successfully following a secured delegation, the amount of trust a resolver has in the parental key is transferred to a child's key. This is the crux of the chain of trust.

Figure 1: **nlnetlabs.nl** *is a secured delegation under* **.nl**. *RTSIG(x)y denotes that a signature over a data x is created with key y.*



In Figure 1 the following takes place.

The **.nl** zone contains the following:

```
nl.   IN    SOA (soa-parameters)
; the zone key
nl.   IN    DNSKEY NLkey
nl.   IN    RRSIG(DNSKEY)NLkey
nl.   IN    RRSIG(SOA)NLkey

nl.   IN    NS ns5.domain-registry.nl.
; this NS is authoratitive
nl.   IN    RRSIG(NS) NLkey

nlnetlabs.nl.   IN   NS open.nlnetlabs.nl.
; no RRSIG here (nonauthoritative data is not signed)

; DS record with a hash of the child's zone key
nlnetlabs.nl.   DS  hash(LabsKey)
; The signature of the parent
nlnetlabs.nl.   RRSIG(DS)NLkey
```

Note: It is important to see that we now have linked a parental signature to something that is *almost* the key of the child.

And the **nlnetlabs.nl** zone has the following:

```
nlnetlabs.nl.    IN    SOA (soa-parameters)
; The zone key
nlnetlabs.nl.    IN    DNSKEY LabsKey
nlnetlabs.nl.    IN    RRSIG(SOA)Labskey
; The (self) signature of the zone key
nlnetlabs.nl.    IN    RRSIG(DNSKEY)Labskey
nlnetlabs.nl.    IN    NS open.nlnetlabs.nl.
nlnetlabs.nl.    IN    RRSIG(NS)LabsKey
```

So the chain of trust looks like the following:

.nl DNSKEY –> nlnetlabs.nl DS –> nlnetlabs.nl DNSKEY

... and with that last key we can validate the data in the **nlnetlabs.nl** zone.

With this "trick" all keys from all the secure **.nl** zones can be chained from the **.nl** "master" key. So instead of one million (the number of zones in **.nl** currently) we need to configure only one key.

As you might have guessed, getting the root zone signed as soon as possible will make it possible to have one key that validates all other keys on the Internet.

We can also look at it from the resolver side. A resolver wants to get an answer. With DNSSEC it has to deal with signatures, keys, and DS records, but those are "side issues"; it still wants an answer.

Suppose **.nl** is secured and a secure delegation to **nlnetlabs.nl** exists. Our resolver has the key of **.nl** anchored. The nameservers of the root zone are also known to the resolver. We further assume the root is not signed. The resolver wants to resolve the address (A record) of **www.nlnetlabs.nl**. What does the actual resolving process look like in DNSSEC? Numerous steps need to be performed:

1. Go to a root server and ask our question.
2. The root server does not know anything about **www.nlnetlabs.nl**, but it *does* know something about **.nl**. The root nameserver refers us to the **.nl** nameservers. This kind of answer is called a *referral*.

3a. Notice that we have a key for **.nl** anchored.

3b. Go to the **.nl** nameserver and ask the **.nl** DNSKEY.

4a. Compare the two DNSKEYs. Continue with the secure lookup only if they match.
The **.nl** DNSKEY is now validated.

4b. Optionally, the RRSIG on the DNSKEY also can be checked.

5. Ask a **.nl** nameserver our question.
6. The **.nl** nameserver is also oblivious about **www.nlnetlabs.nl**, but it does know something about **nlnetlabs.nl**. It returns a secure referral consisting of a DS record plus the RRSIG and some nameservers.
7. The resolver now checks the signature on the DS record. If the signature is valid, the hash of the **nlnetlabs.nl** zone key is ok. The nameservers in the referral do not have any signatures on them.
The hash of the **nlnetlabs.nl** DNSKEY is validated with the **.nl** DNSKEY.
8. Go to the nameserver as specified in the referral and ask for the **nlnetlabs.nl** DNSKEY.
9. Hash the DNSKEY of **nlnetlabs.nl** and compare this hash with the hash in the DS record. If they match continue with the secure lookup.
The **nlnetlabs.nl** DNSKEY is now validated.
10. Ask the nameserver of **nlnetlabs.nl** our question.

11. The nameserver now responds with an answer consisting of the A record of **`www.nlnetlabs.nl`** and an RRSIG made with the **`nlnetlabs.nl`** DNSKEY.
12. The resolver now uses the already validated **`nlnetlabs.nl`** DNSKEY to check the RRSIG. If that signature is valid the RR with the answer is ok and can be given to the application.
13. After these steps we find out that the address of **`www.nlnetlabs.nl`** is 213.154.224.1. We also know it is not a spoofed answer.

This looks like a lot of work and it is—a recursive resolver is a complicated piece of software. Keep in mind, though, that only steps 3ab, 4ab, 7, 8, 9, and 12 are needed for DNSSEC; the rest is how resolving is done in the DNS today.

Deployment

As mentioned earlier, each zone owner generates its own key. To make the secure delegation actually work, this key must somehow be securely transferred to the parent, which is usually the local registry. The registry must have procedures in place to determine whether or not the uploaded key really belongs to the domain it claims to come from. During the *Secure Registry* (SECREG) experiment[5] NLnet Labs has researched the impact DNSSEC has on registries.

But even before the key can be actually uploaded to the parent, a zone administrator still has to do some work; the DNS zone must be signed. This process, called *zone signing*, turns a DNS zone into a DNSSEC zone.

The signing is done offline; first you sign, and then you load the zone. This setup was chosen because at the time (late 1990) computers were not fast enough to generate the signature in real time. Currently it would be possible to do this, but having a server sign every answer it gives is a *Denial-of-Service* (DoS) attack waiting to happen. Especially root servers will be unable to do this.

In DNSSEC a zone can have multiple keys. The signed zone then has multiple signatures per RRset (one for each key). There is no protocol limit on the number of keys. Here we sign with only one zone key. Also signatures in DNSSEC have a start and end date, that is, before and after a certain date interval the signature can no longer be used for validation.

If you use DNSSEC, you must re-sign your zone to generate new signatures with a new validity interval.

The signing of a zone consists of the following steps:

1. The zone key is added to the zone file.
2. The zone file is sorted.

3. Each owner name (for example, a host name) in the zone gets a *Next SECure* (NSEC) record. (Refer to the section "Authenticated Denial of Existence.")
4. For each secured delegation, a DS record is added.
5. The entire zone is then signed with the private key of the zone. Each authoritative RRset gets a signature, including the newly generated NSEC records.

Berkeley Internet Name Domain (BIND)[6] version 9—a popular implementation of the DNS protocols—contains a tool *dnssec-signzone,* which does steps 2 through 5 automatically; we only (manually) need to add the zone key to the zone file. The net result is that we have a bigger, signed, DNSSEC zone. A typical DNSSEC zone is 7 to 10 times larger than its DNS equivalent.

Experiments have shown that this does not pose much of a problem, even for such so-called country code *Top Level Domains* (ccTLDs) as **`.nl`**. The signed **`.nl`** zone was 350 megabytes, slightly more than a half a CD-ROM. And even if scaling problems are occurring, 64-bit machines would certainly help.

A few years ago there was much concern about the signing time. There was fear that it would be impossible to sign large zones, such as **`.com`**.

Experiments disproved this fear. Furthermore, a zone can be split up in pieces and each piece can be signed on a different machine. Later all the signed pieces can be put back together. Signing DNS zones is a highly parallel process.

After signing the zone, it can be loaded in the nameserver. If a resolver is DNSSEC-aware and has been configured with a trusted key that has a chain of trust to the zone key, it can validate the answers. If an answer does not validate, something is wrong and the DNS data must not be used.

The actual Internet-wide deployment of DNSSEC can happen incrementally. Each zone can decide to join independently. It is expected that initially DNSSEC is deployed in subsections of the Internet. These so-called *Islands of Trust* can appear anywhere on the Internet or even in intranets. The only requirement is that the key of the island of trust is distributed to the resolver. Resolvers configured with the key of a certain island of trust are called the *resolvers of interest*. Of course when DNSSEC is widely deployed on the Internet all resolvers are resolvers of interest and will have that key preconfigured.

Authenticated Denial of Existence

As mentioned previously, all records are signed offline. When a nameserver receives a query it looks up the answer plus the signature and returns the two (RRSIG + RRset) to the resolver. The signature is thus not created in real time. How can a secure-aware nameserver then respond to a query for something it does not know (that is, give an NXDOMAIN answer)? The only way to have offline signing and NXDOMAIN answers work together is to somehow sign the data you do not have.

In DNSSEC this is accomplished by the *Next SECure* (NSEC) record. This NSEC record holds information about the next record; it spans the nonexistence gaps in a zone, so to say. For this to work, a DNSSEC zone must be sorted (this is where that requirement stems from). To clarify this, consider an example.

We have a DNS zone, with (for the sake of clarity only the NSEC records are shown):

a.nl
d.nl
e.nl

Next we generate (with the signer) our DNSSEC zone:

a.nl
a.nl NSEC d.nl (span from **a.nl** to **d.nl**)

d.nl
d.nl NSEC e.nl (span from **d.nl** to **e.nl**)

e.nl
e.nl NSEC a.nl (loop back to **a.nl**)

1. If a resolver asks information about b.nl, the nameserver tries to look up the record fails. Instead it finds **a.nl**. It must then return: **a.nl NSEC d.nl** together with the signature. The resolver must then be smart enough to process this information and conclude that **b.nl** does not exist. If the signature is valid, we have an *authenticated denial of existence*. These NSEC records together with their signatures are the major cause of the zone size increase in DNSSEC.

Road to the DS Record

This section briefly considers the history of DNSSEC and, in particular, why the DNSEXT working group has invented this peculiar DS record, which can only exist at the parent side of a zone cut.

In RFC 2535 the DS record did not exist, and this is the reason that the key management in RFC 2535-DNSSEC is very, very cumbersome. In 2000 NLnet Labs ran its first experiment to test deployment of DNSSEC in the Netherlands. Because **.nl.nl** was chosen as the zone under which the secure tree would grow, this experiment became known as the *nl-nl-experiment*. With this experiment it was shown that the current DNSSEC standard (the soon-to-be-obsoleted RFC 2535) was difficult to deploy[7].

An update of a zone key in a child zone required up to 11 (coordinated and sequential) steps with the parent zone. The **.nl** zone now has more than 1 million delegations, so updating all the child zones would require more than 11 million steps. Because these updates could be quite frequent (once a month is typical), this is clearly an administrative nightmare.

Worse yet, if **.nl** lost its private key, all child-zone administrators would have to be notified and they would have to resubmit their public key for re-signing with the new **.nl** key. And because under these conditions the DNS may have been hacked and is thus untrusted, **.nl** is limited in its communication through the Internet; e-mail may not be the preferred method. A telephone call would be more safe, but what kind of organization can make up to one million phone calls in a few days ..?

After various failed attempts (sig@parent[8]) to fix this behavior, the DS record was introduced[1,3]. With this record the administration nightmare is solved, because DS introduces an indirection from the parent zone to a child's zone key.

If **.nl** loses its private key, it can easily resign its own zone, *without* contacting all its children. The DS to child key indirection is still valid, and only the signature of the DS record needs to be updated. This is a local operation.

To test this new DNSSEC specification, a new experiment was set up, which would build a shadow DNSSEC tree in the **.nl** zone. This experiment, called *SECREG,* was to test the new procedures in DNSSEC and, of course, the new DS record. Detailing the conclusions of this experiment is beyond the scope of this article, but in short the conclusion was that the new DNSSEC procedures do not pose much difficulty. At some point, more than 15,000 zones were delegated from the secure tree. A writeup of the experiment and the conclusions can be found in "DNSSEC in NL"[5].

Settings and Parameters in DNSSEC

DNSSEC brings many new parameters to the DNS, including cryptographic ones such as key sizes, algorithm choices, and key and signature lifetimes. Because DNS never has involved cryptography, the best values for these parameters are still open for debate. There is, however, some documentation and knowledge available on this topic (refer to [9] for instance).

One of the major issues is how large (bit length) to make a zone key and how often to re-sign a zone file. The current view is that a parent zone should use larger keys and re-sign more often than a child zone. Also the signature lifetime should be shorter in a parent zone.

Because a parent zone has a DS record (and signature) of a child's zone key, it can decide how long this DS RRSIG must be valid. The shorter this validity interval is, the better protected the child. If a cracker steals a child's zone key, it can forge DNS data. This data looks genuine because the cracker has access to the private key. As long as there is a valid chain of trust to this hijacked key, the child is vulnerable. This chain of trust is broken as soon as the RRSIG of the DS record expires. This argues in favor of a very short parental RRSIG over the DS record.

However, making this interval too short opens the door for accidental mishaps. If a child zone makes an error and somehow the chain of trust is broken, it has until the RRSIG expires to fix the problem. This would recommend a longer signature lifetime. In DNSSEC these and other trade-offs have to be made.

The IETF DNSOP working group is currently addressing these parameters and their trade-offs. The current data came (and comes) from workshops and early test deployments.

Outlook and Prospects

Because DNSSEC requires some additions to the (cc/g)TLD registration process, it could be a while before ccTLDs are capable of deploying DNSSEC. If the protocol is completed this year (2004), it will probably take a few years before registries can advertise DNSSEC domain names.

It is important to consider what DNSSEC actually wants to accomplish; it makes spoofing attacks in the DNS visible—and nothing more. It is not a PKI with all the extra features because key revocation is, for instance, not implemented in DNSSEC. Seen in this light, the protection of private keys in DNSSEC is important, but when a private key is compromised we are just back to plain old DNS.

On the other hand, because DNSSEC does introduce cryptographic material in the DNS and allows for the addition of other (non-DNS) keys, some interesting possibilities emerge. Many technologies on the Internet want to have some kind of simple key distribution mechanism in place; for example: SSH and IPSec. What DNSSEC promises is a system in which we can validate the SSH key from an unknown host with only one key. If the validation is successful, we are quite certain the SSH host key comes from the host from which it claims to come. We get this without any extra effort or cost (from a client's perspective at least). The possibilities are probably endless.

References

[1] Roy Arends, Rob Austein, Dan Massey, Matt Larson, and Scott Rose, "DNS Security Introduction and Requirements," Work In Progress,
http://www.ietf.org/internet-drafts/draft-ietf-dnsext-dnssec-intro-10.txt

[2] Roy Arends, Rob Austein, Dan Massey, Matt Larson, and Scott Rose, "Resource Records for the DNS Security Extensions," Work In Progress,
http://www.ietf.org/internet-drafts/draft-ietf-dnsext-dnssec-records-08.txt

[3] Roy Arends, Rob Austein, Dan Massey, Matt Larson, and Scott Rose, "Protocol Modifications for the DNS Security Extensions," Work In Progress,
http://www.ietf.org/internet-drafts/draft-ietf-dnsext-dnssec-protocol-06.txt

[4] DNS and BIND Talk Notes:
http://www.tfug.org/helpdesk/general/dnsnotes.html

[5] R. Gieben, "DNSSEC in NL,"
http://www.miek.nl/publications/dnssecnl/index.html

[6] BIND9, Berkeley Internet Name Domain, Version 9:
http://www.isc.org/sw/bind/

[7] R. Gieben, "Chain of Trust: The parent-child and keyholder-keysigner relations and their communication in DNSSEC," NIII report CSI-R0111:
http://www.cs.kun.nl/research/reports/info/CSI-R0111.html
http://www.miek.nl/publications/thesis/CSI-report.ps

[8] R. Gieben and T. Lindgreen, "Parent's SIG over Child's KEY,"
http://www.nlnetlabs.nl/dnssec/dnssec-parent-sig-01.txt

[9] O. Kolkman and R. Gieben, "DNSSEC Operational Practices," Work In Progress,
http://www.ietf.org/internet-drafts/draft-ietf-dnsop-dnssec-operational-practices-01.txt

[10] Netscape Communications Corporation, "Introduction to Public-Key Cryptography,"
http://developer.netscape.com/docs/manuals/security/pkin/contents.htm

MIEK GIEBEN graduated in Computer Science in 2001 from the University of Nijmegen (Netherlands) on the subject of DNSSEC. He has been employed by NLnet Labs since that time. He has been using Linux and the Internet since 1995. Currently he is involved in DNSSEC deployment and has co-written parts of NSD2 (which is now fully DNSSEC aware). His personal home page can be found at **http://www.miek.nl/**. The home page of NLnet Labs can be found at **http://www.nlnetlabs.nl/**.
E-mail: **miekg@atoom.net**

Book Review

Network Management

Network Management, MIBs and MPLS by Stephen B. Morris, ISBN 0131011138, Prentice Hall, June 2003.

Few people would question the need for good network management, and books about the S*imple Network Management Protocol* (SNMP) have been circulating for more than ten years now. But the key differentiator of this book is well recognized in its title—it's about SNMP in the context of a *Multiprotocol Label Switching* (MPLS) network. MPLS is now recognized as the convergence technology, and an increasing number of mission-critical services are being deployed over it. World-class network management is vital to keep these services running to the "five nines" level we've all come to expect.

Organization

In this book, Stephen Morris offers a very approachable and comprehensive look at SNMP and the methodology behind the all-important *Management Information Base* (MIB). The first chapter gives the obligatory justification for network management and sets the scene nicely for the rest of the book.

It's amazing to think that SNMP has been around since the late 1980s, and yet if you ask any MPLS operations person, the odds are that person is still using a *Command-Line Interface* (CLI) to actually configure boxes. CLI is a man-machine interface, not a machine-machine interface like SNMP. Even centralized provisioning platforms, such as the former Orchestream (now Metasolve) VPN Manager, simply created a friendly *Graphical User Interface* (GUI) front end for the provisioning procedure, and then ran CLI scripts frantically in the background. The drawbacks of CLI configuration are too numerous to list here, but the basic solution to the problem is to create a scalable and secure machine-to-machine interface. In the IP world the candidate technology for this is SNMPv3, and Morris discusses both the MIB structure (the key to scalability) and the security model in Chapter 2. Because premium MPLS-based services demand secure and robust provisioning, SNMPv3 is the technology of choice.

Chapter 3 describes what Morris calls the "Network Management Problem," although in fact this is described as a whole set of problems, some of which are caused by deficiencies in the SNMP architecture, whereas others are caused by the scale and pace of operations in a modern network. A specific problem that Morris addresses very sensibly is the way that the rapid pace of network technology development impacts the ability to manage these networks. In other words, new technologies tend to appear too quickly for management mechanisms to be optimized for these protocols. To solve this problem, Morris (a software engineer by training) presents a series of "Linked Overviews" (these describe the properties of a given network technology—MPLS, *Asynchronous Transfer Mode* (ATM), etc.—in a procedural framework. In essence this is a kind of recipe for the software developer. In addition, the text is liberally sprinkled with "Developers Notes" that I'm sure will provide invaluable help for people trying to write management system code.

Chapter 4 then takes the approach of solving the "Network Management Problem" to a higher, and perhaps longer-term level, with the proposed development of smarter network management components and more integrated data frameworks. This culminates in a description of *Directory Enabled Networking,* a technology that seemed to flower briefly in the context of network management a few years ago, but then was buried when the telecom recession hit the industry. My own feeling is that the time is right for a rebirth of this approach in modern, converged networks.

Chapter 5 looks at some real *Network Management System* (NMS) issues, using the HP OpenView Network Node Manager as a worked example. Morris is quick to point out that this is not an endorsement of the product, but because it is the most well-known and widely used product in this class, it is the logical choice.

Chapters 6 and 7 look at software components, and Morris's background in software development shines through here in the level of detail, coupled with well-structured explanations.

Chapter 8 describes a very useful case study of using SNMP to provision a tunnel through an MPLS network—a task that is typically performed today using crude CLI techniques.

Chapter 9 contrasts theory and practice in network management, and deals with the loose ends of various topics such as end-to-end security and the integration of a third-party *Open Source Software* (OSS) using standardized northbound *Element Management System* (EMS) interfaces.

Recommended

Overall this is an excellent book that really does deliver what it claims—a comprehensive and practical look at the latest SNMP technologies and techniques. In this regard it stays highly focused, and doesn't waste time with irrelevant discussion on other topics. For example, at first I was disappointed to note that only a page or two of brief explanation is devoted to topics such as *Common Object Request Broker Architecture* (CORBA) and *Extensible Markup Language* (XML). But in the context of what this book is trying to tell us, it makes perfect sense. Each of these topics really needs it own book to cover the topic in similar detail to Morris's work.

Similarly, if you're expecting a description of emerging IP/MPLS *Operations, Administration, and Maintenance* (OA&M), then this book is not for you. Again, I would defend Morris's use of Occam's Razor because OA&M protocols are usually demanded by network staff, and not by OSS operatives. In my own opinion, this situation will gradually change in the next few years, as OA&M is recognized as the "eyes and ears" of the OSS. Perhaps this would be a good place for Mr. Morris to start his next book.

—Geoff Bennett, Heavy Reading
bennett@heavyreading.com

Cooperative Support for Global IPv6 Deployment

The *Regional Internet Registries* (RIRs), the *IPv6 Task Forces* and the *IPv6 Forum* are working in cooperation to support global IPv6 deployment.

The four RIRs, APNIC, ARIN, LACNIC and the RIPE NCC, are responsible for the management of global Internet numbering resources, including IPv4 and IPv6 address space, throughout the world. The RIRs confirm their commitment and continued support towards the deployment of IPv6 in cooperation with the IPv6 Task Forces and with the support of the IPv6 Forum.

The IPv6 Task Forces are focused on rapid IPv6 deployment. They see the adoption of IPv6 by industry, governments, schools and universities is particularly important. The extra address space offered by IPv6 will facilitate the deployment of widespread "always-on" Internet services including broadband access for all. In addition, IPv6's built-in encryption will help improve Internet security and is promoted by many government institutions globally.

The cooperation among the RIRs and the IPv6 Task Forces includes key aspects such as:

- Supporting awareness, education and deployment of IPv6;
- Disseminating information on the progress of IPv6 deployment;
- Encouraging dialogue and ensuring the necessary cooperation between all involved parties;
- Benchmarking IPv6 deployment progress;
- Supporting the adoption of Domain Name Service infrastructure necessary for IPv6;
- Encouraging the participation of all those who are interested in the IPv6 policy development process.

This cooperative effort between the RIRs and the IPv6 Task Forces recognises that while IPv4 address space will be available for many years, new users and usages of the Internet have the potential to rapidly increase the utilisation of IPv4 address space. With the advent of multiple always-on devices, wireless handhelds and 3G mobile handsets, the Internet community needs to prepare for a sharp increase in IP address space utilisation. In order to prevent future operational problems, the global rollout of IPv6 is essential for enabling the development and adoption of new applications and services.

The rollout of IPv6 on this scale requires significant preparation, particularly in terms of training and planning. The RIRs and the IPv6 Task Forces encourage early evaluation by network operators and industry players, in order to promote the necessary technical dialogue and to facilitate widespread adoption. *Internet Service Providers* (ISPs) can already deploy IPv6 in non-disruptive ways that do not require additional investment while providing added value to their customers.

"The RIPE NCC has supported IPv6 from an early stage. We are committed to ensuring that IPv6 resources are provided to RIPE NCC members whenever they are required. We will continue to use the long-established system of address distribution where IP addresses are allocated according to demonstrated need wherever that need is demonstrated," stated Axel Pawlik, Managing Director of the RIPE NCC. "The RIPE NCC is already providing IPv6 training to our members and other tools required to facilitate IPv6 deployment," he added.

Jordi Palet, Founding Member of the EU IPv6 Task Force and co-chair of the IPv6 Forum's Awareness and Education Working Group, sees the formalisation of this cooperative support of IPv6 deployment as an important development. "This cooperative effort ensures the global recognition of the strategic importance of IPv6 in enabling the continued development of the Internet and the worldwide information society. This ongoing coordination will have a positive global benefit for end users and the industry, by reinforcing the resilience of the Internet while allowing for the development of ever-improving applications and services," he said.

Paul Wilson, APNIC Director General, noted that significant advances have been taking place in all the RIR regions with respect to IPv6 allocation and policy. "The RIRs are already working with the IANA and large ISPs to facilitate the delegation of large blocks of IPv6 address space," he stated. "In the Asia Pacific region, a number of countries are taking the lead in terms of IPv6 deployment, and APNIC will continue to offer its support in these areas, and elsewhere, to allow the entire region to benefit from IPv6."

"In the ARIN region, we have received clear direction from the community to make all necessary preparations for IPv6 deployment. This includes work on the allocation policies and procedures, as well as making our own services available via IPv6," stated John Curran, Acting President of ARIN

"LACNIC is involved in the formation of the Latin American and Caribbean IPv6 Task Force and is active in encouraging the participation of its members and the community in IPv6 deployment and policy, and our services are already available over IPv6," said Raúl Echeberría, CEO of LACNIC.

"This global cooperation signals another historic milestone to further accelerate take-up of IPv6 for the global good," applauded Latif Ladid, President of the IPv6 Forum.

"The North American IPv6 Task Force supports the worldwide collaboration with the RIRs to further support the deployment of IPv6 and the next generation Internet mobile society using IPv6," stated Jim Bound, Chair NAv6TF and IPv6 Forum CTO.

As an IPv6 Forum Board member and an ICANN Address Council member, Takashi Arano of the Asia Pacific IPv6 Task Force steering committee supports this collaboration. "Address management, which the RIRs are in charge of, is one of the crucial components for the commercial deployment of IPv6 and its stable operation."

"I hope collaboration between IPv6 Task Forces and the RIRs will result in the advent of an IPv6-powered 'everything-everywhere-every time' networking world," he stated.

IPv6 is a new version of the data networking protocols on which the Internet is based. The *Internet Engineering Task Force* (IETF) developed the basic specifications during the 1990s. The primary motivation for the design and deployment of IPv6 was to expand the available "address space" of the Internet, thereby enabling billions of new devices (PDAs, cellular phones, appliances, etc.), new users and "always-on" technologies (xDSL, cable, Ethernet-to-the-home, fibre-to-the-home, Power Line Communications, etc.).

The existing IPv4 protocol has a 32-bit address space providing for a theoretical 232 (approximately 4 billion) unique globally addressable network interfaces. IPv6 has a 128-bit address space that can uniquely address 2^{128} (340,282,366,920,938,463,463,374,607,431,768,211,456) network interfaces.

The *European IPv6 Task Force* is a volunteer organisation, with over 500 members, open to all the interested parties in advancing the IPv6 deployment in the European region, in cooperation with the rest of the world and other related entities. Further information is available on the IPv6 Task Forces website: **http://www.ipv6tf.org**

Four RIRs exist today. They provide number resource allocation and registration services that support the operation of the Internet globally. The RIRs are independent, not-for-profit organisations that work together to meet the needs of the global Internet community. They facilitate direct participation by all interested parties and ensure that the policies for allocating Internet number resources (such as IP addresses and *Autonomous System Numbers*) are defined by those who require them for their operations.

The RIRs ensure that number resource policies are consensus-based and that they are applied fairly and consistently. The RIR framework provides a well-established combination of bottom-up decision-making and global cooperation that has created a stable, open, transparent and documented process for developing number resource policies.

The RIR framework contributes to the common RIR goal and purpose of ensuring fair distribution, responsible management and effective utilisation of number resources necessary to maintain the stability of the Internet. The RIRs currently consist of:

APNIC: *Asia Pacific Network Information Centre*
http://www.apnic.net

ARIN: *American Registry for Internet Numbers*
http://www.arin.net

LACNIC: *Latin American and Caribbean Internet Addresses Registry*
http://www.lacnic.net

RIPE NCC: *RIPE Network Coordination Centre*
http://www.ripe.net

The *IPv6 Forum* is a world-wide consortium of over 160 leading Internet service vendors, National Research & Education Networks and international ISPs, with a clear mission to promote IPv6 by improving market and user awareness, creating a quality and secure New Generation Internet and allowing world-wide equitable access to knowledge and technology. The key focus of the IPv6 Forum today is to provide technical guidance for the deployment of IPv6. IPv6 Summits are hosted by the IPv6 Forum and staged in various locations around the world to provide industry and market with the best available information on this rapidly advancing technology. **`http://www.ipv6forum.org`**

The *North American IPv6 Task Force* is an all-volunteer non-vendor/service/provider or other entity interest with the IPv6 mission of assisting the North American geography as sub task force of the IPv6 Forum for deployment, education, awareness, technical analysis/direction, transition analysis, political/business/economic/social analysis support and other efforts as required. The members see IPv6 as more important than their own self-interests. **`http://www.nav6tf.org`**

Upcoming Events

The *Internet Corporation for Assigned Names and Numbers* (ICANN) will meet in Kuala Lumpur, Malaysia, July 19–23, 2004, and in Cape Town, South Africa, December 1–5, 2004. For more information see: **`http://www.icann.org`**

ICANN and *The International Telecommunications Union* (ITU) will be jointly hosting a workshop on *country code Top Level Domains* (ccTLDs), in Kuala Lumpur on 24 July. The purpose of this joint ICANN/ITU-T open workshop is to focus on the operation and practical operational issues facing the ccTLDs and to give the opportunity for ccTLD operators and ITU Member States to share their experiences. The Workshop is not a policy meeting, but rather it is intended as a forum for the exchange of views and discussions. Written presentations are encouraged, but not required. Written presentations can be submitted to **`ICANN-ITU-T-Workshop@icann.org`**. Additional information can be found at the ITU-T website: **`http://www.itu.int/ITU-T/worksem/cctld/kualalumpur0704/index.html`**

The IETF will meet in San Diego, CA, August 1–6, 2004 and in Washington, DC, November 7–12, 2004. For more information, visit: **`http://ietf.org`**

Useful Links

The following is a list of Web addresses that we hope you will find relevant to the material typically published in the IPJ.

- The *Internet Engineering Task Force* (IETF). The primary standards-setting body for Internet technologies. **`http//:www.ietf.org`**
- *Internet-Drafts* are working documents of the IETF, its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts. Internet-Drafts are not an archival document series.

These documents should *not* be cited or quoted in any formal document. Unrevised documents placed in the Internet-Drafts directories have a maximum life of six months. After that time, they must be updated, or they will be deleted. Some Internet-Drafts become RFCs (see below). **http://www.ietf.org/ID.html**

- The *Request for Comments* (RFC) document series. The RFCs form a series of notes, started in 1969, about the Internet (originally the ARPANET). The notes discuss man aspects of computer communication, focusing on networking protocols, procedures, programs, and concepts but also including meeting notes, opinion, and sometimes humor. The specification documents of the Internet protocol suite, as defined by IETF and its steering group the IESG, are published as RFCs. Thus, the RFC publication process plays in important role in the Internet standards process. **http://www.rfc-editor/org/**
- The *Internet Society* (ISOC) is a non-profit, non-governmental, international, professional membership organization. **http://www.isoc.org**
- The *Internet Corporation for Assigned Names and Numbers* (ICANN) "...is the non-profit corporation that was formed to assume responsibility for the IP address space allocation, protocol parameter assignment, domain name system management, and root server system management functions." **http://www.icann.org**
- The *North American Network Operators' Group* (NANOG) "...provides a forum for the exchange of technical information, and promotes discussion of implementation issues that require community cooperation." **http://www.nanog.org**
- The *Regional Internat Registries* (RIR) provides IP address block assignments for Internet Service Providers and others. See page 33 for links to APNIC, ARIN, LACNIC and RIPE NCC.
- The *World Wide Web Consortium* (W3C) "...develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential as a forum for information, commerce, communication, and collective understanding." **http://www.w3.org**
- The *International Telecommunication Union* (ITU) "... is an international organization within which governments and the private sector coordinate global telecom networks and services." **http://www.itu.int**

The Internet Protocol Journal

Ole J. Jacobsen, Editor and Publisher

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

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
Cisco Systems, Inc.