DNS

Although it contains just a few simple rules, DNS has grown into an enormously complex system.

DNS (domain name system) is a distributed, coherent, reliable, autonomous, hierarchical database, the first and only one of its kind. Created in the 1980s when the Internet was still young but overrunning its original system for translating host names into IP addresses, DNS is one of the foundation technologies that made the worldwide Internet (and the World Wide Web) possible. Yet this did not all happen smoothly, and DNS technology has been periodically refreshed and refined. Though it's still possible to describe DNS in simple terms, the underlying details are by now quite sublime. This article explores the supposed and true definitions of DNS (both the system and the protocol) and shows some of the tension between these two definitions through the lens of the Internet protocol development philosophy.

SIMPLIFIED VIEW

The DNS namespace has a tree structure, where every node has a parent except the root node, which is its own parent. Nodes have labels that are from 1 to 63 characters long, except the root node whose label is empty. A *domain* is a node in context, and a *fully qualified* domain name has a presentation form that is just the node names, bottom up, with each followed by a period (.). For

Complexity

PAUL VIXIE, INTERNET SYSTEMS CONSORTIUM

DNS Complexity

example, www.google.com is the fully qualified name of a node whose name is www, whose parent is google, whose grandparent is com, and whose great-grandparent is the DNS root.

Nodes are grouped together into *zones*, the apex of each being called a *start of authority* and the bottom edges being called *delegation points* if other zones exist below them, or *leaf nodes* if not. Zones are served by *authority servers* that are either *primary* (if the zone data comes to them from outside the DNS) or *secondary* (if their zone data comes to them from primary servers via a *zone transfer* procedure). For example, root, org, acm.org, and hq.acm.org are separate zones of administrative authority.

Every node can have RRs (*resource records*) that contain the actual content of DNS. Depending on its name, type, and data, an RR can map a host name to an IP address or vice versa, or describe the mail servers for a domain, or serve a growing variety of other purposes. Every RR has a name, class, type, TTL (time to live), and data. TTL is measured in seconds and begins to decrement whenever an RR is transmitted from an authority server. This TTL eventually ticks down to zero inside intermediate caching servers; thus, the authoritative server's stated TTL puts an upper bound on the reuse lifetime of an RR.

DNS clients are most often found inside the runtime libraries of TCP/IP initiators. These runtime libraries are called *resolvers* and most often will not have caches of their own (thus, they are *stub resolvers*). Stub resolvers request *recursive* service from their designated upstream *full resolvers*. A full resolver is capable of caching data for reuse, and of surfing the zone hierarchy to locate a DNS RR no matter where in the namespace it is located or on which authority servers it may be stored.

This view of DNS might not sound "simplified." Take heart, it's actually *over*simplified. Read on to find out what's really going on in there.

ACTUAL VIEW

The character set of a DNS label is modified US-ASCII. It's eight-bit clean except for the values 0x41 to 0x5A (uppercase letters) and the values 0x61 to 0x7A (lowercase letters). These are considered equivalent ranges for the purpose of searching and matching, but their distinctions are retained on the wire and in presentation, in support of possible mixed-case English language trademarks encoded as domain names. Therefore, out of the 256 possible values that an octet can contain, only 230 are unique. In practice, only printable US-ASCII letters and numbers are used, and sometimes a hyphen for internal punctuation. Internationalization of the DNS protocols has been ongoing for 10 years and has no virtual market presence thus far.

Label case is supposed to be preserved when DNS data is cached and forwarded, in support of possible trademarks. The internal data structures that are universally used to support DNS caching, however, keep only one copy of each label. For example, there will be only one com TLD (top-level domain) label in a DNS cache, even though millions of other domain names can be stored "under" that TLD. The net effect here is that if the first .com domain you encounter uses all uppercase letters for its TLD domain label, then all other .com domains you encounter will appear the same way. Therefore, if you cache vix.com first, then you will cache example.com, even if you really did hear example.com next.

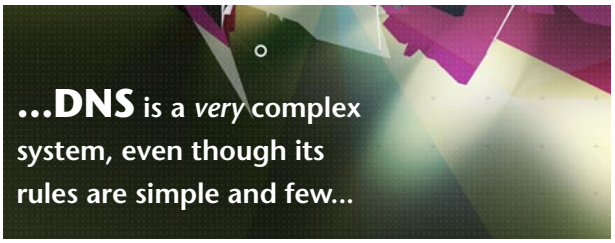
The trailing period (.) of a fully qualified domain name can be omitted in presentation, which can mean either that the name is not fully qualified and has to be searched in the default context or that it is fully qualified. For example, if you are inside ACM world headquarters and point your Web browser at internal, your resolver library will likely assume that you mean internal.hq.acm.org. Disambiguation is either by application convention or by actual searching (maybe you really meant internal.acm.org). One common application convention is, "If there are other period characters in the domain name, then assume that the name is fully qualified." (So, if you were in the San Francisco office, your resolver library might assume that internal means internal.acm.org but it would never guess that internal.hq means internal.hq.acm.org.)

RRs used to describe downward delegations must be present both at the bottom edge of the parent (delegating) zone and at the apex of the child (delegated) zone. These records are expected to be identical, but differences are common and the meaning of such differences is undefined. The system is very robust in the face of this and other undefined conditions, and protocol agents

are prepared to retry pretty hard—and try every possible data path—before giving up. (Thus are local configuration errors transformed into silent resource drains on the world at large.)

RRs are stored and transmitted in semi-atomic sets (<*name*,*class*,*type*> → {*data*}). If a message cannot contain a full RR set, then the transmitting agent can indicate that *truncation* has occurred. Receiving agents are supposed to choose whether the omitted data warrants a new transaction, but in practice, truncation always leads to a new transaction since receivers literally don't know what they're missing. The retry-after-truncation transaction will most likely use a more expensive transport protocol (TCP vs. UDP).

RR sets expire from caches atomically, yet each resource record has its own TTL. The implication is that the lowest TTL governs expiration, yet these separate



TTLs are maintained during transmission and storage. TTL can be clamped to an implementation-specific value to help manage cache size. Records received with TTL 0 expire at the conclusion of the current transaction even if that transaction takes considerable total time, possibly because of the need to fetch other data from other servers to gather all the bits and pieces necessary for completion.

Every RR has a *class*, yet the protocol specification is unclear as to whether the class is part of the tuple designator for an RR set or whether it is, like the *rdata* field, part of each record's payload. Modern software interprets *class* as a zone qualifier, such that every class can have its own namespace, and all records within any zone—and therefore within any RR set—will have the same class. Other interpretations are supported by scripture, but no current implementation works any other way.

The question section of a query message is allowed to contain more than one <*name*,*class*,*type*> tuple, but this is undefined by the protocol and is universally unimplemented. Modern convention requires the question section of a response message to contain a copy of the question section from the query. Implementers argue that the presence of the question inside a response helps to disambiguate responses when transaction rates are very high. This topic has been heavily debated within the DNS standards community.

A response can contain an *additional data* section that carries information that wasn't requested but might, according to the responder, be helpful in interpreting or consuming the answer. This data is optional, and its absence can be the direct cause of subsequent transactions. Its presence is often ignored, however, since this optional data is by nature less secure than the answer that was actually requested.

Two zones having different parents (msn.net and aol.com), whose name servers are each inside the other zone (so, msn.net's name server is ns.aol.com, and aol.com's name server is ns.msn.net), would be *unreachable*. This is because an additional data record in a delegation from the com zone for aol.com, which included the ns.msn.net address as additional data, would be ignored as untrustworthy—likewise for a delegation from net for msn.net containing ns.aol.com's address as additional data. There is no warning message when you do this, unless you count your pager's incessant beeping as a warning message (since that's what would happen next).

This information was gleaned from 20 years of implementation history inside BIND (Berkeley Internet Name Domain). Most of it is not written down anywhere, and some of it would still be considered arguable if you got two or three DNS implementers in a room to talk about it. (And, the hits just keep on coming!)

COMPLEXITY

From this overview, it is possible to conclude that DNS is a poorly specified protocol, but that would be unfair and untrue. DNS was specified *loosely*, on purpose. This protocol design is a fine example of what M.A. Padlipsky meant by "*descriptive* rather than *prescriptive*" in his 1984 thriller, *The Elements of Networking Style* (Prentice Hall). Functional interoperability and ease of implementation were the goals of the DNS protocol specification, and from the relative ease with which DNS has grown from its petri dish into a world-devouring monster, it's clear to me that those goals were met. A stronger document set would have eliminated *some* of the "gotchas" that DNS implementers face, but the essential and intentional looseness of the specification has to be seen as a strength rather than a weakness.

That having been said, a stronger document set written today would not be able to put all of the DNS genies back into their bottles. Too many implementations have guessed differently when presented with a loose specification, and interoperability today is a moving, organic

DNS Complexity

target. When I periodically itch to rewrite the specification from scratch, I know there are too many things that *must* be said that also *cannot* be said. It's as though, in a discussion of the meaning of some bit pattern, a modern description of the protocol—written with full perspective on all that has been done in the DNS field—would have to say, "It could mean *x* but some implementations will think it means *y* so you must be cautious."

If the objective meaning of a set of potential states and conditions and patterns has a complexity index that is a function, somehow, of the combinations and permutations of those states, conditions, and patterns, as well as the multiple interpretations and deliberate uncertainties, then DNS is a *very* complex system, even though its rules are simple and few, and even though a new DNS protocol agent can be constructed using only a few thousand lines of software code.

FUTURE COMPLEXITY

An important complexity consideration is, "How will all this look in the future?" As the protocol continues to evolve, protocol designers will add new signaling patterns meaningless to today's DNS clients and servers, and they might make the new patterns contingent on negotiation of protocol level. I say "might" because it's also possible that some new signaling pattern will be considered so harmless that it can be included even in messages that previous-generation protocol agents will process. Depending on the quality of implementation of those protocol agents, things that ought to be harmless might actually be harmless. It's equally possible that an untested and erroneous code path will get its first exercise on the day when somebody else on the Internet is doing something that should have been harmless and could even be described in a hot-off-the-presses Internet standards document. Let's explore some of the known possibilities in this area, including several that have been Internet standards for a decade, yet are still considered "new" by many people.

IDN (Internationalized Domain Names). The Internet may have gotten its start inside a country whose native language is US-ASCII, but it's a worldwide network now. Billions of users who would not otherwise know or use US-ASCII have learned about it because DNS labels are expressed in that notation. To express multilingual symbol sets usually means Unicode, whose binary representation is not directly compatible with the upper/lowercase "folding" required for DNS labels. Therefore, Unicode names must first be converted to a US-ASCII subset similar to Base64 or uuencode, then carried through DNS, then converted back to Unicode for presentation on the remote end. These encoded domain names will look like random hexadecimal line noise to any IDN-ignorant endpoint or middlebox.

EDNS (Extended DNS). When most of the fixed binary fields in the DNS protocol all sort of filled up at once, we came up with a way to negotiate larger sizes. The most important limit was in DNS message size for the UDP (user datagram protocol) transport, which had been capped at 512 octets. With EDNS, virtually all modern DNS initiators advertise a buffer size of 1,500 octets or larger. Note that other elements of the EDNS specification were less successful, such as the recommendation for supporting extended label types, which turns out to be nonnegotiable after all.

IXFR (Incremental Zone Transfer). If a DNS zone is very large, then the cost of transferring the entire zone from the primary authority servers to all secondary authority servers whenever the zone changes might be prohibitive (as measured in computing and network resources). With IXFR, it is possible to transfer only zone deltas. In fact, there is a degenerate case of an IXFR transaction where the entire zone delta is carried in a single UDP datagram (if it fits). This avoids the three-way TCP handshake normally required by a DNS zone transfer.

Dynamic Update. DNS originally assumed that all changes would come into a zone from outside DNS (for example, a text file on a server computer's file system). A machine-generated update—as, for example, if a DHCP server wanted to record a name address mapping following a successful address assignment—had no data path toward the zone. Dynamic updates made it possible to carry DNS update transactions inside the DNS protocol itself. Subject to a still-evolving access control scheme, fully mechanized updates are now quite popular.

Change Notification. In the old days, secondary name

servers would have to poll the primary name server to find out whether a zone's serial number (and thus, presumably, the zone's content) had changed. Any changes made to the zone on the primary name server could remain unpublished for up to that polling interval. Nowadays, a primary name server can transmit in-bound change notices to secondary name servers, thus triggering an early poll (and most likely an immediate incremental zone transfer).

Transaction Security. Since IP source addresses are nonrepudiable by design, and since DNS uses the stateless UDP, it follows that there is no cause for confidence in the identity of a purported signaler. TSIG (transaction signature) allows any DNS transaction to be digitally signed and verified using symmetric cryptography and a previously established shared secret. Although this could conceivably be used for controlling query access, its main practical uses are to control access for dynamic updates and zone transfers.

Data Authenticity. Referring once again to the nonrepudiable nature of IP source addresses, it is computationally trivial to pollute a caching name server with data that was never seen or published or approved by the editor of the administrative authority zone that it purports to have come from. This requires asymmetric cryptography wherein a zone administrator generates a key pair, publishes the public key in the zone apex, publishes a hash of the public key in the parent (delegating) zone, and uses the private key to generate per-RR-set digital signatures. Resolvers who choose to validate these signatures will have to fetch the entire key-signature chain back to some previously known trust anchor, which is usually the root zone's public key. DNSSEC (DNS Security Extensions) has been in production for 12 long years, without any production use to date, and we in the Internet standards community look forward to solving and re-solving this problem for many years to come. (But, I'm not bitter about it.) Meanwhile, the only reason that DNS isn't attacked more often is that nobody trusts its authenticity. (A catch-22, perhaps?)

PREDICTABILITY AND MODELING

Thomas J. Ryan's 1977 novel *The Adolescence of P-1* tells the story of the unintentional creation of artificial intelligence as a side effect of a programmer's desire to violate an operating system's security policy. In the story, a software implementation of game theory similar to "The Prisoner's Dilemma" creates unpredictability using only elements that have known and predictable behaviors.

On the one hand, this notion is clearly fanciful, and nobody has yet suggested that the Internet or even DNS has achieved or will ever achieve consciousness. On the other hand, the combination of things that were left unspecified in the protocol, things that were loosely specified in the protocol, and things that were unenforceably specified in the protocol—and implementations in the field that interpret the protocol specifications in all of their myriad ways—describes a rich and multidimensioned space where it's almost deliberately impossible to know exactly what's happening or exactly what would happen under describable circumstances. This holds true for the Internet's routing system as well, but in DNS there are more variables in every axis than in any other distributed system I've studied. We who work in the field think of DNS a little bit as though it were alive, and I hope that after reading this article, you can see why. Q

SUGGESTED FURTHER READING

Davis, C., Vixie, P., Goodwin, T., Dickinson, I. 1996. A means for expressing location information in the Domain Name System. IETF; http://www.ietf.org/rfc/rfc1876.txt.

Gulbrandsen, A., Vixie, P., Esibov, L. 2000. A DNS RR for specifying the location of services. IETF; http://www.ietf.org/rfc/rfc2782.txt.

Vixie, P. 1996. A mechanism for prompt notification of zone changes. IETF; http://www.ietf.org/rfc/rfc1996.txt .

Vixie, P., et al. 1997. Dynamic updates in the domain name system. IETF; http://www.ietf.org/rfc/rfc2136.txt.

Vixie, P., et al. 2000. Secret key transaction authentication for DNS. IETF; http://tools.ietf.org/html/rfc2845.

Vixie, P., Kato, A. 2004. Modern DNS as a coherent dynamic universal database. *IEICE Transactions on Communications* (October).

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

PAUL VIXIE has been contributing to Internet protocols and Unix systems as a protocol designer and software architect since 1980, and was a cofounder of ISC (Internet Systems Consortium) in 1994. Early in his career, he developed and introduced SENDS, proxynet, rtty, cron, and other lesser-known tools. He is considered the primary modern author and technical architect of BIND8. In 1995, Vixie cofounded PAIX (Palo Alto Internet Exchange), which was sold to AboveNet in 1999. He was named CTO in 2000, and then president of the PAIX subsidiary in 2001. He also cofounded MAPS (Mail Abuse Prevention System), a California nonprofit company established in 1998 with the goal of stopping spammers.