The Internet Protocol Journal

June 2002 Volume 5, 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 networking industry is full of acronyms, as the table of contents for this issue clearly illustrates. According to the dictionary, an acronym is "...a word formed from the initial letter or letters of each of the successive parts or major parts of a compound term." While neither BEEP nor ENUM are strictly speaking acronyms, these "short names" are becoming ever more prevalent and difficult to keep track of. We promise to continue to provide acronym expansion whenever possible.

BEEP is an example of a technology that came to life in a very short time. While IETF standards often take years from initial idea to protocol specification, BEEP seems to have happened in just over a year. There is already a textbook on BEEP from which our first article is adapted. Marshall Rose gives an overview of the BEEP framework and explains how you can get involved in its further development.

ENUM refers to the use of the *Domain Name System* (DNS) to look up telephone numbers and subsequently route telephone calls to the right destination using the Internet as the underlying routing fabric. This integration of the traditional telephone network with the Internet is becoming a reality and several standardization bodies are working on technologies to make this as seamless as possible. Geoff Huston explains the mechanisms and politics behind ENUM.

Our series "One Byte at a Time" examines the *Dynamic Host Configuration Protocol* (DHCP). This protocol is widely used to provide IP address and other basic routing information to clients. This is particularly useful for mobile devices, but it can be used in any network environment. Since the IP addresses are assigned as leases with a configurable time limit, DHCP also provides for effective address management. Douglas Comer explains the details of DHCP and its predecessor BOOTP.

As always, we appreciate your feedback. Send your comments and questions to **ipj@cisco.com**

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

An Overview of BEEP

by Marshall Rose, Dover Beach Consulting

The *Blocks Extensible Exchange Protocol* (BEEP) is something like "the missing link between the application layer and the *Transmission Control Protocol* (TCP)."

This statement is a horrific analogy because TCP is a transport *protocol* that provides reliable connections, and it makes no sense to compare a protocol to a layer. TCP is a highly-evolved protocol; many talented engineers have, over the last 20 years, built an impressive theory and practice around TCP. In fact, TCP is so good at what it does that when it came to survival of the fittest, it obliterated the competition. Even today, any serious talk about the transport protocol revolves around minor tweaks to TCP. (Or, if you prefer, the intersection between people talking about doing an "entirely new" transport protocol and people who are clueful is the empty set.)

Unfortunately, most application protocol design has not enjoyed as excellent a history as TCP. Engineers design protocols the way monkeys try to get to the moon—that is, by climbing a tree, looking around, and finding another tree to climb. Perhaps this is because there are more distractions at the application layer. For example, as far as TCP is concerned, its sole reason for being is to provide a full-duplex octet-aligned pipe in a robust and network-friendly fashion. The natural result is that while TCP's philosophy is built around "reliability through retransmission," there isn't a common mantra at the application layer.

Historically, when different engineers work on application protocols, they come up with different solutions to common problems. Sometimes the solutions reflect differing perspectives on inevitable tradeoffs; sometimes the solutions reflect different skill and experience levels. Regardless, the result is that the wheel is continuously reinvented, but rarely improved.

So, what is BEEP and how does it relate to all this? BEEP integrates the best practices for common, basic mechanisms that are needed when designing an application protocol over TCP. For example, it handles things like peer-to-peer, client/server, and server/client interactions. Depending on how you count, there are about a dozen or so issues that arise time and time again, and BEEP just deals with them. This means that you get to focus on the "interesting stuff."

BEEP has three things going for it:

- It's been standardized by the *Internet Engineering Task Force* (IETF), the so-called "governing body" for Internet protocols.
- There are open source implementations available in different languages.
- There's a community of developers who are clueful.

The standardization part is important, because BEEP has undergone a lot of technical review. The implementation part is important, because BEEP is probably available on a platform you're familiar with. The community part is important, because BEEP has a lot of resources available for you.

Application Protocols

An application protocol is a set of rules that says how your application talks to the network. Over the last few years, the *Hypertext Transfer Protocol* (HTTP) has been pressed into service as a general-purpose application protocol for many different kinds of applications, ranging from the *Internet Printing Protocol* (IPP)[1] to the *Simple Object Access Protocol* (SOAP)[2]. This is great for application designers: it saves them the trouble of having to design a new protocol and allows them to reuse a lot of ideas and code.

HTTP has become the reuse platform of choice, largely because:

- It is familiar.
- It is ubiquitous.
- It has a simple request/response model.
- It usually works through firewalls.

These are all good reasons, and—if HTTP meets your communications requirements—you should use it. The problem is that the widespread availability of HTTP has become an excuse for not bothering to understand what the requirements really are. It's easier to use HTTP, even if it's not a good fit, than to understand your requirements and design a protocol that does what you really need.

That's where BEEP comes in. It's a toolkit that you can use for building application protocols. It works well in a wide range of application domains, many of which weren't of interest when HTTP was being designed.

BEEP's goal is simple: you, the protocol designer, focus on the protocol details for your problem domain, and BEEP takes care of the other details. It turns out that the vast majority of application protocols have more similarities than differences. The similarities primarily deal with "administrative overhead"—things you need for a working system, but aren't specific to the problem at hand. BEEP mechanizes the similar parts, and lets you focus on the interesting stuff.

Application Protocol Design

Let's assume, for the moment, that you don't see a good fit between the protocol functions you need and either the e-mail or the Web infrastructures. (We'll talk more about this later on in the section "The Problem Space".) It's time to make something new.

First, you decide that your protocol needs ordered, reliable delivery. This is a common requirement for most application protocols, including HTTP and the *Simple Mail Transfer Protocol* (SMTP).[3] The easiest way to get this is to layer the protocol over TCP.

So, you decide to use TCP as the underlying transport for your protocol. Of course, TCP sends data as an octet stream—there aren't any delimiters that TCP uses to indicate where one of your application's messages ends and another one begins. This means you have to design a framing mechanism that your application uses with TCP. That's pretty simple to do—HTTP uses an octet count and SMTP uses a delimiter with quoting.

Since TCP is just sending bytes for you, you need to not only frame messages, but have a way of marking what's in each message. (For example, a data structure, an image, some text, and so on.) This means you have to design an encoding mechanism that your application uses with the framing mechanism. That's also pretty simple to do—HTTP and SMTP both use *Multipurpose Internet Mail Extensions* (MIME).[4]

Back in the early 1980s, when I was a young (but exceptionally cynical) computer scientist, my advisor told me that protocols have two parts: *data* and *control*. It looks like the data part is taken care of with MIME, so it's onto the control part. If you are fortunate enough to know ahead of time every operation and option that your protocol will ever support, there's no need for any kind of capabilities negotiation. In other words, your protocol doesn't need anything that lets the participants tell each other which operations and options are supported. (Of course, if this is the case, you have total recall of future events, and really ought to be making the big money in another, more speculative, field.)

The purpose of negotiation is to find common ground between two different implementations of a protocol (or two different versions of the same implementation). There are lots of different ways of doing this and, unfortunately, most of them don't work very well. SMTP is a really long-lived, well-deployed protocol, and it seems to do a pretty good job of negotiations. The basic idea is for the server to tell the client what capabilities it supports when a connection is established, and then for the client to use a subset of that.

Well, that's just the first control issue. The next deals with when it's time for the connection to be released. Sometimes this is initiated by the protocol, and sometimes it's required by TCP because the network is unresponsive. To further complicate things, if the release is initiated by the protocol, maybe one of the computers hasn't finished working on something, so it doesn't want to release the connection just yet.

Some application protocols don't do any negotiation on connection release, and just rely on TCP to indicate that it's time to go away—even though this is inherently ambiguous. Is ambiguity a good thing in a protocol? Computers lack subtlety and nuance, so in protocols between computers, ambiguity is a bad thing. For example, in HTTP 1.0 (and earlier), you often didn't know whether a response was truncated or not. For a more concrete example, interested readers will be amused by page 2 of RFC 962.[5]

The final control issue deals with what happens between connection establishment and release. Most application protocols tend to be client/server in nature: one computer establishes a connection, sends some requests, gets back responses, and then releases the connection. But, are the requests and responses handled one at a time (in lock-step), or can multiple requests be outstanding, either in transit or being processed, at the same time (asynchronously)?

In the original SMTP, the lock-step model was implicitly assumed by most implementors; later on, SMTP introduced a capability to allow limited pipelining. Regardless, as soon as we move away from lock-stepping, it looks as though we'll need some way of correlating requests and responses.

Although this is a step in the right direction, some application protocols need even more support for asynchrony. The reasoning is a little convoluted, but it all comes down to performance. There's a lot of overhead involved in terms of establishing a connection and getting the right user state, so it makes sense to maximize the number of transactions that get done in a single connection. While this helps in terms of overall efficiency, if the transactions are handled serially, then transactional latency—the time it takes to transit the network, process the transaction, and then transit back—isn't reduced (and may even be increased); a transaction might be blocked while waiting for another to complete. The solution is to be able to handle transactions in parallel.

Earlier I mentioned how, back in the 1980s, protocols had two parts, *data* and *control*. Today, things have changed. First of all, I'm still cynical, but more comfortable with it, and—perhaps as important—many might argue that protocols now have a third part, namely *security*.

The really unfortunate part is that security is a moving target on two fronts:

- When you deploy your protocol in different environments, you may have different security requirements.
- Even in the same environment, security requirements change over time.

This introduces something of a paradox: modern thinking is that security must be tightly integrated with your protocol, but at the same time, you have to take a modular approach to the actual technology to allow for easy upgrades. Worse, it's very easy to get security very wrong. (Just ask any major computer vendor!) Few applications folks are also expert in protocol security, and obtaining that expertise is a time-consuming, thankless task, so there's a lot of benefit in having a security mechanism menu, developed by security experts, that applications folk can pick from.

Now the good news: there's already something around designed to meet just those requirements. It's called the *Simple Authentication and Security Layer* (SASL), and a lot of existing application protocols have been retrofitted over the last four years to make use of it.

Well, let's see what all this means. Without ever having talked about what your application protocol is going to do to earn a living, we have to develop solutions for:

- Framing messages
- Encoding data
- Negotiating capabilities (versions and options)
- Negotiating connection release
- Correlating requests and responses
- Handling multiple outstanding requests (pipelining)
- Handling multiple asynchronous requests (multiplexing)
- Providing integrated and modular security
- Integrating all these things together into a single, coherent framework

So, going back to the question "Why use BEEP?", the answer is pretty simple: if you use BEEP, you simply don't have to think about any of these things. They automatically get taken care of.

Now maybe you're the kind of hardcore engineer that really wants to solve these problems yourself. Okay, go right ahead! But first, I'll let you in on a little secret: engineers have been solving these problems since 1972. In fact, they keep solving them over and over again. For each problem, there are usually two or three good solutions, and while individual tastes may vary, the sad fact is that you can make any of them work great if you're willing to put in the hours. But why put in the hours if they have nothing to do with the primary reason for writing the application protocol to begin with? Isn't there something more productive that you'd care to do with your life than design yet another framing protocol?

So, what's really *new* about BEEP? The short answer is: not much. The innovative part is that some folks sat down, did an analysis of the problems and solutions, and came up with an integrated framework that put it all together. That's not really innovation, but it's really good news if you're already familiar with the building blocks that BEEP uses.

Doesn't all this stuff add a lot of overhead? The short answer is: nope. The reason is a little more complex. BEEP is fairly minimalistic—it provides a simple mechanism for negotiating things on an à la carte basis. If you don't want privacy, no problem; don't turn it on. If you don't want parallelism, that's easy; just say "no" if the other computer asks for it. The trick here is two-fold:

- BEEP's inner mechanisms (for example, framing) are pretty lightweight, so you don't incur a lot of overhead using them (even if you don't use all the functionality they provide).
- BEEP's outer mechanisms (for example, encryption) are all controlled via bilateral negotiation, so you can decide exactly what you want to get and pay for.

There's no free lunch, but if you want to start with something "lean and mean," BEEP doesn't slow you down, and when you want to bulk up (say, by adding privacy), BEEP lets you negotiate it. You incur only the overhead you need. (This overhead *will* show up, regardless of whether you use BEEP or grow your own mechanisms.)

It turns out that this philosophy can yield some interesting results. For example, take a look at this high-level scripting fragment:

```
::init -server example.com -port 10288 -privacy strong
```

This fragment is invoking a procedure to establish a BEEP session. With the exception of the last two terms, it looks pretty conventional.

The last two terms tell the procedure to "tune" the session by looking at the security protocols supported in common, selecting one that supports "strong privacy," and then negotiating its use. What's interesting here is that neither the person who designed the application protocol nor the person who wrote the application making the procedure call has to be a security expert. The choice to use strong privacy, and how it gets transparently used, is all an issue of provisioning. Of course, the application protocol designer may still provide security guidelines to the implementor; naturally, the implementor may bundle a wide range of security protocols with the code. However—and this is key—everyone got to focus on what they do best (even the security guys), and it still comes together into a working system.

The cool part here is how easily this all integrates into an evolving protocol. Back in the good ol' days (say the mid-1980s) when the *Post Office Protocol* (POP)[6] was defined, this kind of flexibility wasn't available. Whenever someone wanted to add a new security mecha-

nism for authentication or privacy, you had to muck with the entire protocol. With BEEP's framework, you just add a module that works seamlessly with the rest of the protocol. This means less work for everyone, and presumably fewer mistakes getting the work done.

Now we've come full circle: the reason for using BEEP is because it makes it a lot easier to specify, develop, maintain, and evolve new application protocols.

The Problem Space

BEEP works for a large class of application protocols. However, you should always use the right tool for the right job. Before you start using BEEP for a project, you should ask yourself whether your application protocol is a good fit for either the e-mail or Web models.

Dave Crocker, one of the Internet's progenitors, suggests that network applications can be broadly distinguished by five operational characteristics:

- Server push or client pull
- Synchronous (interactive) or asynchronous (batch)
- Time-assured or time-insensitive
- Best-effort or reliable
- Stateful or stateless

For example:

- The World Wide Web is a pull, synchronous, time-insensitive, reliable, stateless service.
- Internet mail is a push, asynchronous, time-insensitive, best-effort, stateless service.

This is a pretty useful taxonomy.

So, your first step is to see whether either of these existing infrastructures meet your requirements. It's easiest to start by asking if your application can reside on top of e-mail. Typically, the unpredictable latency of the Internet mail infrastructure raises the largest issues; however, in some cases it's a non-issue. For example, in the early 1990s, some of the earliest business-to-business exchanges were operated over e-mail (for example, USC/ISI's FAST project). If you can find a good fit between your application and Internet e-mail, use it!

More likely, though, you'll be tempted to use the Web infrastructure, and there are a lot of awfully good reasons to do so. After all, when you use HTTP:

- There's lots of tools (libraries, servers, etc.) to choose from.
- It's easy to prototype stuff.
- There's already a security model.
- You can traverse firewalls pretty easily.

All of this boils down to one simple fact: it is pretty easy to deploy things in the Web infrastructure. The real issue is whether you can make good use of this infrastructure.

HTTP was originally developed for retrieving documents in a LAN environment, so HTTP's interaction model is optimized for that application. Accordingly, in HTTP:

- Each session consists of a single request/response exchange.
- The computer that initiates the session is also the one that initiates the request.

What needs to be emphasized here is that this is a perfectly fine interaction model for HTTP's target application, as well as many other application domains.

The problem arises when the behavior of your application protocol doesn't match this interaction model. In this case, there are two choices: make use of HTTP's extensibility features, or simply make do. Obviously, each choice has some drawbacks. The problem with using HTTP's extensibility features is that it pretty much negates the ability to use the existing HTTP infrastructure; the problem with "just making do" is that you end up crippling your protocol. For example, if your application protocol needs asynchronous notifications, you're out of luck.

A second problem arises due to "the law of codepaths." The HTTP 1.1 specification, RFC 2616[10] is fairly rigorous. Even so, few implementors take the time to think out many of the nuances of the protocol. For example, the typical HTTP transaction consists of a small request, which results in a (much) larger response. Talk to any engineer who's worked on a browser and they'll tell you this is "obvious." So, what happens when the "obvious" doesn't happen?

Some time ago, folks wanted a standardized protocol for talking to networked printers. The result was something called the *Internet Printing Protocol* (IPP)[1]. IPP sits on top of HTTP. At this point, the old "obvious" thing (small request, big response) gets replaced with the new "obvious" thing—the request contains an arbitrarily large file to be printed, and the response contains this tiny little status indication. A surprising amount of HTTP software doesn't handle this situation particularly gracefully (that is, long requests get silently truncated). The moral is that even though HTTP's interaction model doesn't play favorites with respect to lengthy requests or responses, many HTTP implementors inadvertently make unfortunate assumptions.

A third problem deals with the unitary relationship between sessions and exchanges. If a single transaction needs to consist of more than one exchange, it has to be spread out over multiple sessions. This introduces two issues:

- In terms of stateful behavior, the server computer has to be able to keep track of session state across multiple connections, imposing a significant burden both on the correctness and implementation of the protocol (for example, to properly handle time-outs).
- In terms of performance, TCP isn't designed for dealing with back-to-back connections—there's a fair amount of overhead and latency involved in establishing a connection. This is also true for the security protocols that layer on top of TCP.

HTTP 1.1 begins to address these issues by introducing persistent connections that allow multiple exchanges to occur serially over a single connection, but still the protocol lacks a session concept. In practice, implementors try to bridge this gap by using "cookies" to manage session state, which introduces ad-hoc (in)security models that often result in security breakdowns (as a certain Web-based e-mail service provider found out).

This brings us to a more general fourth problem: although HTTP has a security model, it predates SASL. From a practical perspective, what this means is that it's very difficult to add new security protocols to HTTP. Of course, that may not be an issue for you.

If you can find a good fit between your application and the Web infrastructure, use it! (For those interested in a more architectural perspective on the reuse of the Web infrastructure for new application protocols, consider RFC 3205[7].)

Okay, so we've talked about both the e-mail and Web infrastructures, and we've talked about what properties your application protocol needs to have in order to work well with them. So, if there isn't a good fit between either of them and your application protocol, what about BEEP?

BEEP's interaction model is pretty simple, with the following three properties:

- Each session consists of one or more request/response exchanges.
- Either computer can initiate requests or notifications.
- It's connection-oriented.

By using BEEP, you get an amortization effect with respect to the cost of connection establishment and state management. This is largely derived from the first property. Similarly, the second property gives BEEP its ability to support either peer-to-peer or client-server interactions. What we really need to explain is the connection-oriented part.

To begin, all three of the interaction models we've looked at (BEEP, e-mail, and the Web) are connection-oriented. (Although e-mail may get delivered out of order, the commands sent over each e-mail "hop" are processed in an ordered, reliable fashion.) The connection-oriented model is the most commonly used for application protocols, but it does introduce some restrictions.

A connection-oriented interaction model means that data is delivered reliably and in the same order as it was sent. If you don't require ordered, reliable delivery, you don't need a connection-oriented interaction model. For example, Internet telephony applications don't fit this model, nor do traditional multicast applications.

So, BEEP is suitable for unicast application protocols (two computers are talking to each other). However, not all unicast applications need a connection-oriented model—for example, the *Domain Name System* (DNS) manages name-to-address resolutions just fine without it. In fact, if your protocol is able to limit each session to exactly one request/response exchange with minimalist reliability requirements, and also limit the size of each message to around 65K octets, then it's probably a good candidate for using the *User Datagram Protocol* (UDP) instead.

The IETF and BEEP

BEEP is an emerging standard from the *Internet Engineering Task Force* (IETF). The IETF is a voluntary professional organization that develops many of the protocols running in the Internet. (Of course, anyone is free to develop their own protocols to run in their own little part of the Internet, but if you want multi-vendor support, you need an organization like the IETF.) So why does the IETF care about BEEP?

The answer is that the largest area in the IETF deals with application protocols. There are usually over two dozen working groups developing different application protocols. And, the IETF has been doing this for a long, long time. It turns out that even though there are well-engineered solutions to the different overhead issues, BEEP is the first time that the IETF decided to develop a standard approach that integrates the best practices for each issue. Before BEEP, each working group would spend endless hours arguing about different solutions, and then, if any time was remaining, they might sit down and look at the actual problem domain. (Okay, this is an exaggeration... but not by much!)

So, here's the process by which BEEP got designed:

- Identify the common domain-independent problems.
- Determine the best solution for each problem.
- Integrate the solutions into a consistent framework.
- Declare victory.

Now, the obvious question is: how do you determine what's "best?"

The truth is that in some cases, the answer is obvious, and in other cases, the answer is arbitrary. (Protocol experts hate to admit this, but in some cases, there is no clear winner, and it's simply better to pick *one* and order another drink.) Since most of what BEEP does is hidden from the application designer and implementor, there's really not a lot of mileage in going through it here.

beepcore.org

Where can you find out more about BEEP? To start, you can always consult the two RFCs: the BEEP core framework[8] and the BEEP's mapping onto TCP[9]. However, it's probably better to start with the BEEP community Web site **`http://beepcore.org`** where you'll find:

- News about BEEP meetings and events
- Information about BEEP projects, programmers, and consultants
- Information about beepcore (open source) and commercial software
- BEEP-related RFCs, Internet-Drafts, and whitepapers

[This article is adapted from *Beep—The Definitive Guide,* by Marshall T. Rose, ISBN 0-596-00244-0, O'Reilly & Associates, 2002. Used with permission. **`http://www.oreilly.com/catalog/beep/`**]

References

[1] Herriot, R., Ed., Butler, S., Moore, P., Turner, R., "Internet Printing Protocol/1.0: Encoding and Transport," RFC 2565, April 1999.

[2] **`http://www.w3.org/TR/SOAP/`**

[3] Postel, J., "Simple Mail Transfer Protocol," RFC 821, August 1982.

[4] Freed, N., Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," RFC 2045, November 1996.

[5] Padlipsky, M. A., "TCP-4 prime," RFC 962, November 1985.

[6] Rose, M. T., "Post Office Protocol: Version 3," RFC 1081, November 1988.

[7] Moore, K., "On the use of HTTP as a Substrate," RFC 3205, February 2002.

[8] Rose, M., "The Blocks Extensible Exchange Protocol Core," RFC 3080, March 2001.

[9] Rose, M., "Mapping the BEEP Core onto TCP," RFC 3081, March 2001.

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

MARSHALL T. ROSE is the prime mover of the BEEP Protocol. In his former position as the Internet Engineering Task Force (IETF) area director for network management, he was one of a dozen individuals who oversaw the Internet's standardization process. Rose was responsible for the design, specification, and implementation of several Internet-standard technologies, and wrote more than 60 of the Internet's Requests For Comments (RFCs). With a Ph.D. in information and computer science from the University of California, Irvine, Rose is the author of several professional texts.
E-mail: **`mrose@dbc.mtview.ca.us`**

ENUM—Mapping the E.164 Number Space into the DNS

by Geoff Huston, Telstra

Many communications networks are constructed for a single form of communication, and are ill suited to being used for any other form. Although the Internet is also a specialized network in terms of supporting digital communications, its relatively unique flexibility lies in its ability to digitally encode a very diverse set of communications formats, and then support their interaction over the Internet. In this way many communications networks can be mapped into an Internet application and in so doing become just another distributed application overlayed on the Internet. From this admittedly Internet-centric perspective, voice is just another Internet application. And for the growing population of *Voice over IP* (VoIP) users, this is indeed the case. Being able to transmit voice over the Internet is not enough. Allowing one Internet handset to connect to any other Internet handset is still not enough. In the same way that walkie-talkies became ubiquitous mobile phones only when there was a seamless integration with the telephone network, a truly useful VoIP approach will be one that supports seamless integration with the telephone network.

The basics of the telephony world are very simple indeed. Telephone handsets are little more than a speaker and a microphone. When a call is made, the network connects the microphone of one party to the speaker of the other, and vice versa. Of course you don't need a specialized telephone network to support the carriage of voice. As any user of a desktop computer would confirm, there are now a plethora of applications that can deliver a voice signal across the network. For an application to support a voice conversation, a conventional approach is to use a network base of the *User Datagram Protocol* (UDP) transport protocol, with a *Real-Time Protocol* (RTP) overlay, and the RTP payload is an encoded version of the original analogue voice signal. Carrying voice signals in real time across an Internet is a well-understood network service, with an accompanying set of existing protocols and associated applications.

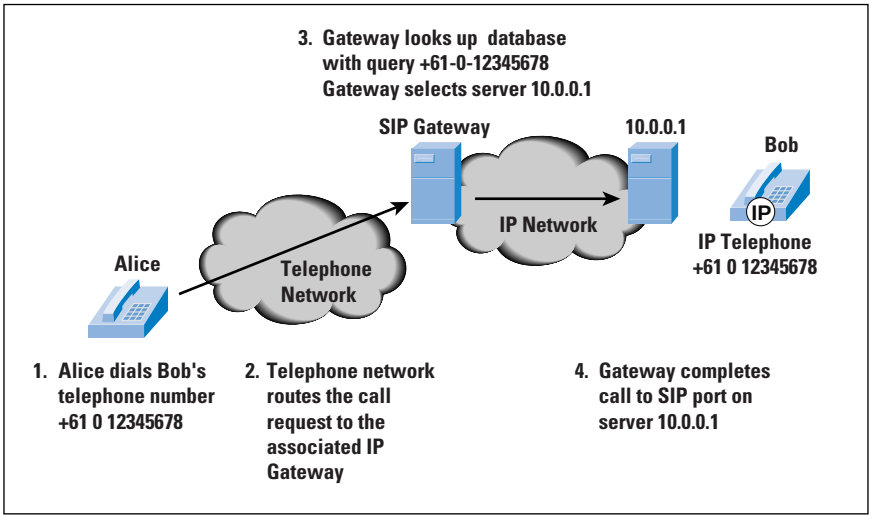
E.164 Addresses and IP Services

However, being able to transmit voice signals across a network is not enough. It was Strowger's step-by-step switching system of the late 19th century that transformed the telephone into a truly useful communications network, allowing any telephone subscriber to initiate a conversation with any other subscriber. This has evolved today into a global numbering plan where every device connected to the telephone network is assigned a unique numerical address. This numbering plan is administered by the *International Telecommunication Union* (ITU), and the plan, Recommendation E.164, involves the assignment of number prefixes to each country code administrator[1].

If the Internet voice domain interoperates seamlessly with the telephone network, supporting this E.164 numbering plain into the realm of the Internet is a critical step. To make Internet telephony truly useful, the Internet telephony world has to be able to interface to the telephone network by allowing Internet-connected telephone devices to make and receive calls to any other telephone device, whether the other device is connected to the Internet, connected to the telephone network, or connected to any other network that seamlessly interoperates with the telephone network. For this to work, one of the preconditions is that every Internet device that supports telephone operation needs to also have an alias in the form of a unique telephone address. But there's a bit more to it than simple numbering.

Each Internet telephone is also an IP device, and, for the Internet component of the end-to-end path, the voice traffic will be carried by IP packets. These packets obviously require the IP address of the Internet telephone device. So each Internet telephone requires both an Internet address and a telephone address. It is the mapping from a telephone number to an IP address that is the crucial part of this function.

Figure 1: Calling an IP Telephone



Consider an example. When Alice, on a normal telephone, wants to call Bob, on an Internet phone, all Alice needs to do is simply dial Bob's telephone number, or his E.164 address (Figure 1). Of course, because Bob's phone is connected to the Internet and can't directly receive Alice's call request, a gateway is necessary. The telephone system should be able to map Alice's call request to the Internet telephony gateway that is configured to act as Bob's gateway agent. The gateway then needs to translate Bob's E.164 phone number into an IP address. Then the gateway has to map the telephone network signals associated with Alice's call request to corresponding signals within an Internet session initiation protocol, and then send these IP packets to Bob's Internet phone. If Bob answers the call, the phone uses the same protocol to inform the gateway, which then sends a corresponding telephone call code across the telephone network to Alice.

When Bob accepts the call, the gateway can then pass all data originating from Alice to Bob's IP address, and all data received from Bob's IP address across to the telephone connection to Alice for the duration of the call. Alice never needs to know that Bob is using an Internet device. Alice dialed a phone number, heard it ring, and then heard Bob answer the call. For Alice, nothing has changed. Bob heard the phone ring, picked it up, and talked to Alice. For Bob, nothing has changed.

The simplest way to configure each gateway is to load each gateway with a configured list of E.164 phone numbers and corresponding IP addresses. This approach is currently very common, but, like all statically configured approaches, has its weaknesses. But what happens when the IP device is numbered dynamically using the *Dynamic Host Configuration Protocol* (DHCP), or if it's mobile, and moves from one service provider's IP network to another, or when the end subscriber changes providers and that subscriber's network is renumbered, or when the primary gateway fails and the providers want to switch to a secondary device? In other words, how can this mapping be dynamic rather than static?

The way a dynamic domain name-to-IP address mapping can be maintained on the Internet is through the Internet *Domain Name System* (DNS). The telephony gateway can use the the E.164 address as the DNS query, and request the DNS to return the corresponding IP address. In our example, when Alice rings Bob, the gateway can use the DNS to obtain Bob's current IP address. The gateway can then use the *Session Initiation Protocol* (SIP) to send to Bob's Internet phone a call request, which then starts Bob's phone ringing. If Bob changes IP address, then the corresponding change is a change in the DNS, not in the gateway itself. If the primary gateway fails and a secondary gateway is used, the secondary system can already access all necessary mappings through the DNS.

So the general approach of using the DNS to contain this mapping is one with some merit, but, as always, the devil is in the details. There are two parts to mapping a E.164 number into the DNS. The first is the nature of the transforms to be applied to the E.164 address to obtain a DNS query string, and the second is the form of the DNS response to this query.

Mapping E.164 Addresses into DNS Query Strings

One possible approach to mapping an E.164 number into the DNS is to simply place numbers as text blocks into the DNS. In this way, the number *+61-0-12345678* could be mapped to the DNS string **`61012345678.example.com`**. If this method were to be used for a sizable number of E.164 numbers, there are obvious DNS performance implications associated with the size of this DNS zone file, together with the issue of frequency of update of the zone and its cache characteristics.

There are also a large number of E.164 country code delegated authorities and, consequently, a large number of entities who would like to be the authority for parts of such a monolithic unstructured DNS zone file.

In order to avoid these issues, some structure in the E.164 address space has to be used to map into the hierarchical name structure used in the DNS. One helpful observation is that E.164 numbers and Internet domain names use opposite ordering. Whereas a fully qualified domain name, such as **`test.example.com,`** has the more specific parts to the left and the most general part, the root, on the right of the name, a telephone number code has the most general part, the reference to the country code prefix "**`+`**" to the left and the more specific parts to the right. If one were to reverse the order of E.164 symbols, then the two address domains would have a similar structure.

One of the first efforts to provide a mapping between E.164 number and the DNS was part of the TPC fax gateway service, started in 1993[2]. This approach uses a reversed E.164 number, and treats every digit as a node on the DNS name hierarchy. In our example, the E.164 address *+61 0 12345678* would map to the DNS query string **`8.7.6.5.4.3.2.1.0.1.6.tpc.int.`** (in the TPC service, the parent DNS zone of this mapping is **`tpc.int.`**)

This mapping has some very convenient properties. Each country code corresponds to a delegatable DNS domain, so that the international country code for Australia, *+61,* can have a corresponding DNS delegation for the zone **`1.6.tpc.int.`** Within the country code the DNS can be further delegated to operators in a manner that parallels the further delegation of E.164 common prefix number blocks.

This same mapping is used by ENUM, using a DNS name parent of **`e164.arpa.`** The mapping entails taking a complete E.164 address (including the country code), and then removing all nondigit symbols from the address. The digit string is reversed and a "**`.`**" is placed between each pair of digits. The string **`.e164.arpa.`** is then appended to make a complete DNS query string. Using this process, our example number *+61-0-12345678* is transformed into the DNS query:
`8.7.6.5.4.3.2.1.0.1.6.e164.arpa.`

Although this form of mapping is technically well suited to the DNS, it does mean that the DNS equivalent of the E.164 address is not very easily adapted to our conventional use of telephone numbers. The implication is that it is likely that Internet-based telephony applications will continue to present E.164 numbers in their user interfaces as conventional telephone numbers, and manipulate the DNS equivalent strings as internal objects.

The DNS Response

The telephone network supports more than simple voice conversations, and any serious attempt to bridge the telephone network and the Internet also should be able to also handle various forms of text messaging and paging services as well as document transmission undertaken as faxes. The desired outcome is that the interface between the telephone network and the Internet should be able to seamlessly redirect the telephone service to the appropriate Internet service. In other words, we are seeing a requirement that a set of services associated with the same E.164 address should be able to be mapped to a set of IP servers, rather than a single server with a single IP address.

The implication is that the DNS response to an ENUM query should have a richer functionality than simply returning a single IP address. In DNS terms, associating a conventional "A" DNS resource record with each ENUM domain name is not sufficiently flexible for our purposes.

The approach adopted by the TPC fax gateway service was to map a fax in the telephone environment to an e-mailed multimedia message in the Internet environment. To support this mapping, telephone numbers were mapped to DNS *Mail Exchange* (MX) resource records, and these records were mapped to a mail server's IP address in a second DNS lookup.

ENUM attempts to solve a more general model of providing mappings for any relevant service. One possible approach is to use a collection of DNS name roots, one for each mappable service. Thus, for example, **`fax.e164.arpa.`** could hold mappings for the fax service, while **`voice.e164.arpa.`** could hold mappings for voice services, and so on. However, this approach is not consistent with the generic architecture of the DNS, and the distribution of service information has the potential to lead to synchronization errors. Usefully, the DNS allows a collection of resource records to be associated with a DNS name, and this set of records is returned as the answer to a query. It is then left to the application to determine which particular record to use, with perhaps some preference hints provided in the DNS response. The approach used by ENUM takes advantage of this DNS capability, and ENUM uses the DNS to map an **`e164.arpa`** number onto a collection of service-specific *Uniform Resource Identifiers* (URIs)[3].

A gateway that uses ENUM to query the DNS will receive the complete collection of service-specific URIs in response to a request to translate an E.164 address to a URI. Depending on the type of service being requested, the gateway can then select the most appropriate URI and use the DNS a second time to translate the domain name part of the URI to an IP address using the URI-specific DNS resource record as a query term. The gateway can then use the full URI specification to open an IP session with the selected service port and complete the service transaction.

The URI resource records used by ENUM are *Naming Authority Pointers* (NAPTR) records[4]. This form of use of the DNS allows for entries where the entry itself can be decomposed into further delegations, using name formats that use URI syntax[5].

NAPTR fields contain numerous components:

- An *Order* field to specify the order in which multiple NAPTR records must be processed
- A *Preference* field to determine the processing order when multiple NAPTR records have the same order value
- A *Service* field to specify the resolution protocol and service
- *Flags* to modify the actions of further DNS lookups
- A *regular expression* to allow the query client to rephrase the original request in a DNS format
- A *Replacement* field to define the next DNS query object

The intended operation of ENUM is to first take the E.164 number and convert it to a query in the **`e164.arpa`** domain. The resultant set of services is specified by the returned collection of NAPTR records. The agent selects a service that matches the service characteristics of the original request, and takes the corresponding URI for further resolution by the DNS. The elements of this URI are further decomposed as per any rewrite rules in the NAPTR record. DNS queries are generated as per the sequence of preferred NAPTR rewrite operations. The ultimate result of this sequence of DNS queries is the specification of a protocol, an associated port address, and the IP address for a preferred server for the service.

An Example of the Use of ENUM

Let's say Bob's Internet telephone services are mapped to the E.164 address *+61-0-12345678*. When Alice tries to call Bob, the telephone network routes the call request toward the Internet gateway that is the nominated service agent for this E.164 number. The Internet gateway takes the call setup request with Bob's number and first reverses the digits, then inserts a "**`.`**" between each digit, and finally appends **`e164.arpa.`** The resultant DNS string is the fully qualified domain name **`8.7.6.5.4.3.2.1.0.1.6.e164.arpa.`** This name is then passed as a query to the DNS, to retrieve all associated NAPTR DNS resource records.

Bob has specified that he prefers to receive calls using SIP addressed to user **`bob`** at the server **`telebob.au`** by placing the following in the DNS:

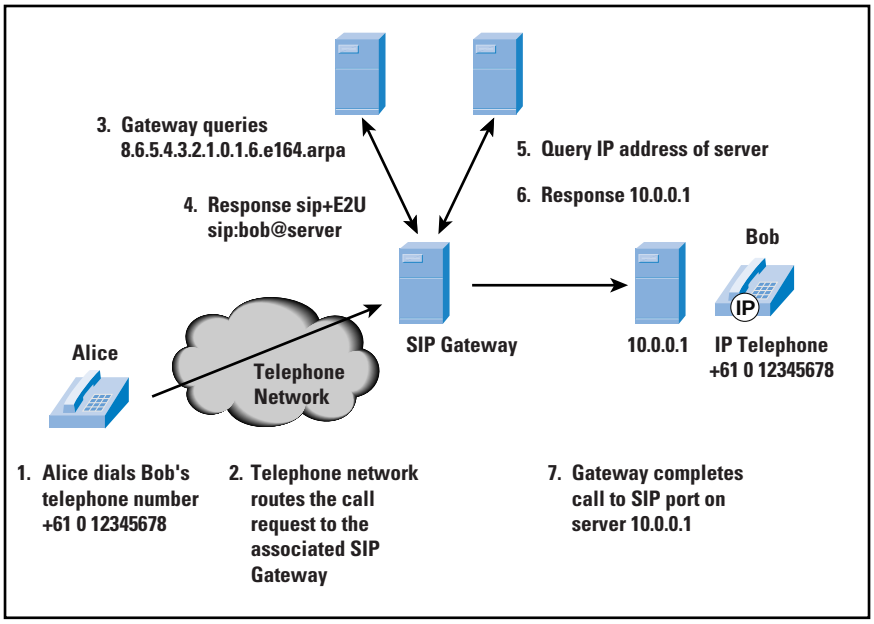
```
$ORIGIN 8.7.6.5.4.3.2.1.0.1.6.e164.arpa.
   IN NAPTR 100 10 "u" "sip+E2U"  "!^.*$!sip:bob@sip.telebob.au!" .
```

In this case the DNS entry uses an order value of 100 and a preference of 10. The "**u**" flag indicates that the rule is terminal and that the specified URI is to be used. The service field specifies that the SIP protocol is to be used, in conjunction with the E.164 to URI (E2U) resolution service[6]. The operation of the regular expression produces the URI of the form **sip:bob@telebob.au.**

For this call request, the gateway picks the **sip+E2U** service and performs the associated regular expression transform using the original E.164 number and the regular expression. This produces the **sip:** URI. The gateway then uses the DNS a second time to translate the domain part of the URI, **sip.telebob.au**, into an IP address using a DNS A record.

The gateway then opens up a session with UDP port 5060 on this SIP server to complete the call setup, requesting a voice session with the user Bob on this server. (Figure 2).

Figure 2: Using ENUM to Call an IP Phone



If, on the other hand, Alice is sending Bob a short text message, then Bob may want this to be delivered to him as mail. Bob would add the following entry into the DNS:

```
$ORIGIN 8.7.6.5.4.3.2.1.0.1.6.e164.arpa.
   IN NAPTR 100 10 "u" "sip+E2U"     "!^.*$!sip:bob@sip.telebob.au!"    .
   IN NAPTR 102 10 "u" "mailto+E2U"  "!^.*$!mailto:bob@mail.pobob.au!" .
```

In this case the gateway would use this **mailto:** URI and use the domain part of the URI as a MX DNS query. The DNS responses are a list of mail server names and associated preferences. The gateway then selects this more preferred server and resolves this name to an IP address by a further query to the DNS for an A address record.

The gateway can complete the original text message delivery request by opening a TCP session on port 25 of the mail server and sending the message as mail addressed to user **bob@mail.pobob.au.**

Services in ENUM

Other URIs can also be associated with an E.164 number, even services not normally associated with a mapping of a telephone function. These may include **http:** URIs, even other E.164 telephone numbers, specified by **tel:** URIs.

Let's complete the example of Bob, who wants his SIP phone, mail address, Web page, and mobile telephone to be referenced from a single telephone number.

```
$ORIGIN 8.7.6.5.4.3.2.1.0.1.6.e164.arpa.
   IN NAPTR 100 10 "u" "sip+E2U"     "!^.*$!sip:bob@sip.telebob.au!"   .
   IN NAPTR 100 10 "u" "mailto+E2U"  "!^.*$!mailto:bob@mail.pobob.au!" .
   IN NAPTR 100 10 "u" "http+E2U"    "!^.*$!http://www.webhostbob.au"  .
   IN NAPTR 103 10 "u" "tel+E2U"     "!^.*$!tel:+61-4-12341234"        .
```

Alice can enter the phone number *61012345678* into her browser and retrieve Bob's Web page in response. She can address e-mail to this number and thereby send mail to Bob. Or she can make a telephone call to Bob's SIP phone, and if it does not answer she can try Bob on his mobile phone. And she can do all this from a single number.

Numerous interesting technical issues still need to be resolved, such as the necessity and level of cacheing within the global ENUM system and the creation of a standard registry scheme for ENUM service definition.

The Politics of ENUM

There is quite some depth in the capabilities of the regular expression rewrite rules in ENUM, but the basic functionality is one of mapping a telephone number to a collection of service points that are associated with the telephone customer who was assigned that telephone number.

Despite this apparent functional simplicity, ENUM appears to have a powerful set of attractors for regulatory and social controversy.

A key benefit of moving into ENUM and the associated realm of IP-based voice communications is that service creation becomes a function of the edge and not the network. What were seen as telephone network functions such as no answer and busy redirect, call forwarding, number translation, and conference calls can all be implemented as edge applications driven by user scripts, rather than what we now see in the telephone network as value-added network-based services. One way of viewing this ENUM approach is that the DNS is functionally capable of assuming the role of service control point for telephone services, taking over the role undertaken by *Signaling System 7/Channel 7* (SS7/C7).

Service creation and signaling are slipping away from the hands of network operators into the hands of enterprises and eventually consumers, in much that same way that the Internet has redefined other services in terms of edge-based function instead of network mediation.

There is also the issue of ownership of these ENUM DNS zones, or to put it another way: who gets to populate the **e164.arpa** domain with all these URIs? It could be that this is a responsibility of existing telephone service providers, because after all these entities operate the E.164 address space in each country. It could also be that this is a responsibility of Internet Service Providers (ISPs), because the data in the resource records is describing Internet-based services. Or maybe the end subscribers get to populate the DNS with their own entries, based on a collection of services that may be sourced from a set of providers.

It is quite conceivable that we could see ISPs that have no direct role in carrying voice traffic wanting access to a country's E.164 number plant in order to provide various forms of ENUM services. Given that each element of an ENUM service collection can use URIs that refer to different ISP services, it is possible that the one ENUM record can be populated by URIs referring to numerous different service providers. This model of multi-agent access to such infrastructure resource records is a novel concept to many regulatory and operating regimes, where a single operator manages the entire associated infrastructure elements that are needed to deliver a service.

Some of the discussion about ENUM has been on more subtle aspects of this mapping. There's the choice of **e164.arpa** as the common DNS root for ENUM DNS entries. At an international level there's a lingering perception that "**arpa**" is too American and that a name root of "**int**" appears to be more neutral.

But there's something else lurking here, which has surfaced within the regulatory debate in the United States. North America has the .164 country code of "1," implying that under ENUM there is a single DNS domain for ENUM, namely **1.e164.arpa.** Single domains imply single operators, and single operators have an implication of a noncompetitive monopoly service regime. There has been a call for multiple E.164 DNS root locations for North America, allowing for two or more competing service operators using different DNS hierarchies to locate their ENUM services.

On the one side there is the view that such attempts to create multiple partially populated ENUM name hierarchies to support competitive service provision in ENUM-based services are no more than an incitement to address and service chaos. This chaos would, in turn, seriously hamper the uptake of ENUM services.

On the other hand, the competitive provision proponents of multiple DNS root domains argue that a regulatory-sanctioned monopoly is still a monopoly, and this monopoly situation will likely lead to high service prices for ENUM services. This escalated pricing structure would, in turn, seriously hamper the uptake of ENUM services.

As we have seen with the use of multiple services for an **`e164.arpa`** entry, the proponents of ENUM envisage a single telephone number as being an alias not only for your Internet phone service, but also for instant messaging, e-mail, your Web page, and any other service that is associated with you. One identifier is all that would be required to reach you, using a service protocol and service provider of your choice. The implication of such a use of a telephone number is, on a personal level, no more business cards cluttered with phone numbers, fax numbers, mobile numbers, e-mail addresses, Web addresses, and instant-messaging handles. Phone numbers are still the most widely used naming scheme in communications, and the use of these numbers as a universal locator has the advantage of being linguistically neutral as well as enjoying almost ubiquitous use. There are no international character set issues within this particular number space. All we need is just one ENUM address, or just one number, for all these services.

> "One number to rule them all, one number to find them, one number to bring them all and in the darkness bind them," is the ENUM version of Tolkien's saga[7].

But one person's ease of use is often another's opportunity to exploit. To be *Lord of the Numbers* would indeed be a powerful role if such uses of ENUM were to become widespread. In addition to the commercial opportunity in operating ENUM registries, ENUM can be seen as yet another erosion of personal privacy on the Internet. It can be viewed as one more step toward the use of single individual digital identity that could be used to track individuals within the Internet. On a more immediate and mundane level of concern it opens up the opportunity for spammers to use a wealth of new ways to drive you to complete distraction.

It appears that the technical components of ENUM are generally the most straightforward part. The regulatory and social implications of ENUM are more of a concern, and it is here that with ENUM we are entering into "the Land of Mordor where the shadows lie."

Further reading:

[1] List of ITU-T Recommendation E.164 Assigned Country Codes, available online at:
http://www.itu.int/itudoc/itu-t/ob-lists/icc/e164_717.pdf

[2] Malamud, C., and Rose, M., "Principles of Operation for the TPC.INT Subdomain: Remote Printing—Technical Procedures," RFC 1530, October 1993.

[3] Fälström, P., "E.164 Number and DNS," RFC 2916, September 2000.

[4] Mealling, M., and Daniel, R., "The Naming Authority Pointer (NAPTR) DNS Resource Record," RFC 2915, September 2000.

[5] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, August 1998.

[6] Handley, M., Schulzrinne, H., Schooler, E., and Rosenberg, J., "SIP: Session Initiation Protocol," RFC 2543, March 1999.

[7] Tolkien, J. R. R., *The Lord of the Rings,* George Allen and Unwin, London 1955.

[8] **http://www.enum.org** has a good overview of ENUM and its potential application as well as references to further ENUM resources.

[9] "Interim Approval for ENUM Provisioning," see the Fragments section in this issue of *The Internet Protocol Journal,* page 37.

GEOFF HUSTON holds a B.Sc. and a M.Sc. from the Australian National University. He has been closely involved with the development of the Internet for the past decade, particularly within Australia, where he was responsible for the initial build of the Internet within the Australian academic and research sector. Huston is currently the Chief Scientist in the Internet area for Telstra. He is also a member of the Internet Architecture Board, and is the Secretary of the APNIC Executive Committee. He is author of *The ISP Survival Guide,* ISBN 0-471-31499-4, *Internet Performance Survival Guide: QoS Strategies for Multiservice Networks,* ISBN 0471-378089, and coauthor of *Quality of Service: Delivering QoS on the Internet and in Corporate Networks,* ISBN 0-471-24358-2, a collaboration with Paul Ferguson. All three books are published by John Wiley & Sons.
E-mail: **gih@telstra.net**

One Byte at a Time

Bootstrapping with BOOTP and DHCP

by Douglas Comer, Purdue University

The process of starting a computer system is known as *bootstrapping*. In most systems, the initial bootstrap sequence begins with code in ROM, which the CPU executes. The ROM code only contains a first step—it merely loads an image into the computer's RAM and branches to the image. There are two approaches used to obtain an image:

- *Embedded system:* On a diskless computer, the ROM code contains sufficient support software to permit network communication. The ROM code uses the network support to locate and download an image.
- *Conventional computer:* On a computer that has secondary storage (for instance, a PC), the ROM code loads the image from a well-known place on disk. Typically, the loaded image consists of an operating system that then controls the computer.

In either case, the image loaded by ROM is not tailored to the specific physical hardware. Instead, an image is *generic,* which means that before it can be used, it must be configured for the local hardware. In particular, the image does not contain such networking details as the computer's IP address, address mask, or domain name. Each of these items must be supplied before applications can use TCP/IP.

Early in the history of TCP/IP, designers chose to provide a separate mechanism for each item of configuration information. Thus, the *Reverse Address Resolution Protocol* (RARP) only allowed a computer to obtain its IP address. When subnet masks were introduced, ICMP Address Mask messages were added to allow a computer to obtain a subnet mask. The chief advantage of such an approach lies in flexibility—a computer can decide which items to obtain from a local file on disk and which to obtain over the network. The chief disadvantage becomes apparent when one considers the network traffic and delay. A given computer must issue a series of small request messages. More important, each response returns a small value (for instance, a 4-octet IP address). Because networks enforce a minimum packet size, most of the space in each packet is wasted.

BOOTP

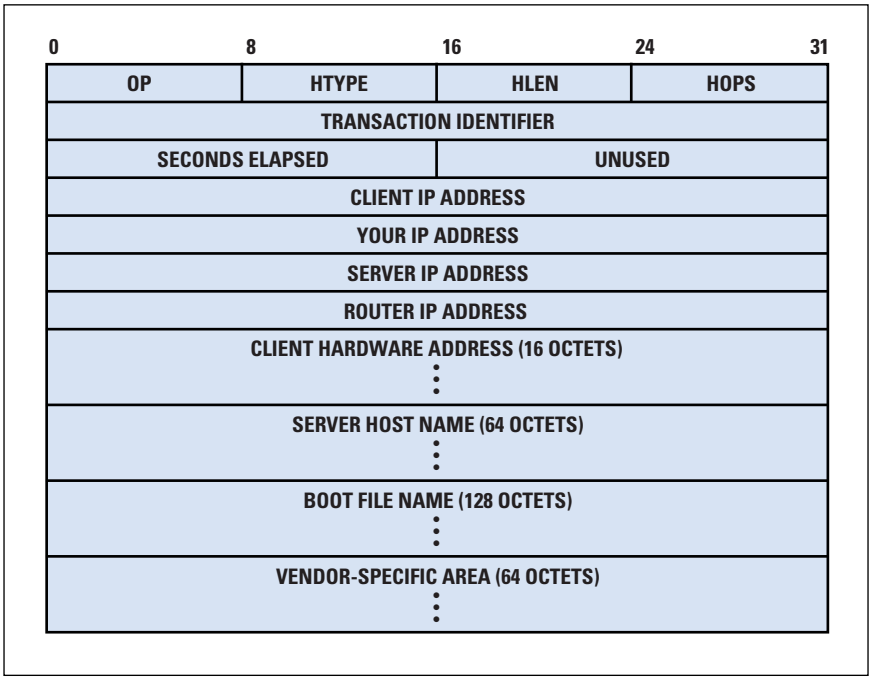
As the complexity of configuration grew, TCP/IP protocol designers observed that many of the configuration steps could be combined into a single step if a server was able to supply more than one item of configuration information. To provide such a service, the designers invented the *BOOTstrap Protocol* (BOOTP). To obtain configuration information, protocol software broadcasts a *BOOTP Request* message.

A BOOTP server that receives the request looks up several pieces of configuration information for the computer that issued the request, places the information in a single *BOOTP Response* message, and returns the reply to the requesting computer. Thus, in a single step, a computer can obtain information such as the computer's IP address, the server's name and IP address, and the IP address of a default router.

Like other protocols used to obtain configuration information, BOOTP broadcasts each request. Unlike other protocols used for configuration, BOOTP appears to use a protocol that has not been configured: BOOTP uses IP to send a request and receive a response. How can BOOTP send an IP datagram before a computer's IP address has been configured? The answer lies in a careful design that allows IP to broadcast a request and receive a response before all values have been configured. To send a BOOTP datagram, IP uses the all-1's limited broadcast address as a *DESTINATION ADDRESS*, and uses the all-0's address as a *SOURCE ADDRESS*. If a computer uses the all-0's address to send a request, a BOOTP server either uses broadcast to return the response or uses the hardware address on the incoming frame to send a response via unicast. (The server must be careful to avoid using ARP because a client that does not know its IP address cannot answer ARP requests.)

Thus, a computer that does not know its IP address can communicate with a BOOTP server. Figure 1 illustrates the BOOTP packet format. The message is sent using UDP, which is encapsulated in IP.

Figure 1: BOOTP Packet Format



Each field in a BOOTP message has a fixed size. The first seven fields contain information used to process the message. The *OP* field specifies whether the message is a *Request* or a *Response*, and the *HTYPE* and *HLEN* fields specify the network hardware type and the length of a hardware address. The *HOPS* field specifies how many servers forwarded the request, and the *TRANSACTION IDENTIFIER* field provides a value that a client can use to determine if an incoming response matches its request. The *SECONDS ELAPSED* field specifies how many seconds have elapsed since the computer began to boot. Finally, if a computer knows its IP address (for instance, the address was obtained using RARP), the computer fills in the *CLIENT IP ADDRESS* field in a request.

Later fields are used in a response message to carry information back to the computer that is booting. If a computer does not know its address, the server uses field *YOUR IP ADDRESS* to supply the value. In addition, the server uses fields *SERVER IP ADDRESS* and *SERVER HOST NAME* to give the computer information about the location of a computer that runs servers. Field *ROUTER IP ADDRESS* contains the IP address of a default router.

In addition to protocol configuration, BOOTP allows a computer to negotiate to find a boot image. To do so, the computer fills in field *BOOT FILE NAME* with a generic request (for instance, the computer can request the UNIX operating system). The BOOTP server does not send an image. Instead, the server determines which file contains the requested image, and uses field *BOOT FILE NAME* to send back the name of the file. Once a BOOTP response arrives, a computer must use a protocol like the *Trivial File Transfer Protocol* (TFTP) to obtain a copy of the image.

Automatic Address Assignment

Although it simplifies loading parameters into protocol software, BOOTP does not solve the configuration problem completely. When a BOOTP server receives a request, the server looks up the computer in its database of information. Thus, even a computer that uses BOOTP cannot boot on a new network until the administrator manually changes information in the database.

Can protocol software be devised that allows a computer to join a new network without manual intervention? Yes—several such protocols exist. For example, IPX and IPv6 can generate a protocol address from the computer's hardware address. To make automatic generation work correctly, the hardware address must be unique. Furthermore, if the hardware address and protocol address are not the same size, it must be possible to translate the hardware address into a protocol address that is also unique.

The AppleTalk protocols use a *bidding* scheme to allow a computer to join a new network. When a computer first boots, the computer chooses a random address. For example, suppose computer *C* chooses address *17*. To ensure that no other computer on the network is using the address, *C* broadcasts a request message and starts a timer. If no other computer is using address *17*, no reply will arrive before the timer expires; *C* can begin using address *17*. If another computer is using *17,* the computer replies, causing *C* to choose a different address and begin again.

Choosing an address at random works well for small networks and for computers that run client software. However, the scheme does not work well for servers. To understand why, recall that each server must be located at a well-known address. If a computer chooses an address at random when it boots, clients will not know which address to use when contacting a server on that computer. More important, because the address can change each time a computer boots, the address used to reach a server may not remain the same after a crash and reboot.

A bidding scheme also has the disadvantage that two computers can choose the same network address. In particular, assume that computer *B* sends a request for an address that another computer (for example, *A*) is already using. If *A* fails to respond to the request for any reason, both computers will attempt to use the same address, with disastrous results. In practice, such failures can occur for a variety of reasons. For example, a piece of network equipment such as a bridge can fail, a computer can be unplugged from the network when the request is sent, or a computer can be temporarily unavailable (for instance, in a hibernation mode designed to conserve power). Finally, a computer can fail to answer if the protocol software or operating system is not functioning correctly.

DHCP

To automate configuration, the *Internet Engineering Task Force* (IETF) devised the *Dynamic Host Configuration Protocol* (DHCP). Unlike BOOTP, DHCP does not require an administrator to add an entry for each computer to the database that a server uses. Instead, DHCP provides a mechanism that allows a computer to join a new network and obtain an IP address without manual intervention. The concept has been termed *plug-and-play networking*. More important, DHCP accommodates computers that run server software as well as computers that run client software:

- When a computer that runs client software is moved to a new network, the computer can use DHCP to obtain configuration information without manual intervention.
- DHCP allows nonmobile computers that run server software to be assigned a permanent address; the address will not change when the computer reboots.

To accommodate both types of computers, DHCP cannot use a bidding scheme. Instead, it uses a client-server approach. When a computer boots, the computer broadcasts a *DHCP Request* to which a server sends a *DHCP Reply*. (The reply is classified as a DHCP *offer* message that contains an address the server is offering to the client.)

An administrator can configure a DHCP server to have two types of addresses: permanent addresses that are assigned to server computers, and a pool of addresses to be allocated on demand. When a computer boots and sends a request to DHCP, the DHCP server consults its database to find configuration information.

If the database contains a specific entry for the computer, the server returns the information from the entry. If no entry exists for the computer, the server chooses the next IP address from the pool, and assigns the address to the computer.

In fact, addresses assigned on demand are not permanent. Instead, DHCP issues a *lease* on the address for a finite period of time. (When the administrator establishes a pool of addresses for DHCP to assign, the administrator must also specify the length of the lease for each address.)

When the lease expires, the computer must renegotiate with DHCP to extend the lease. Normally, DHCP will approve a lease extension. However, a site may choose an administrative policy that denies the extension. (For example, a university that has a network in a classroom might choose to deny extensions on leases at the end of a class period to allow the next class to reuse the same addresses.) If DHCP denies an extension request, the computer must stop using the address.

Optimizations in DHCP

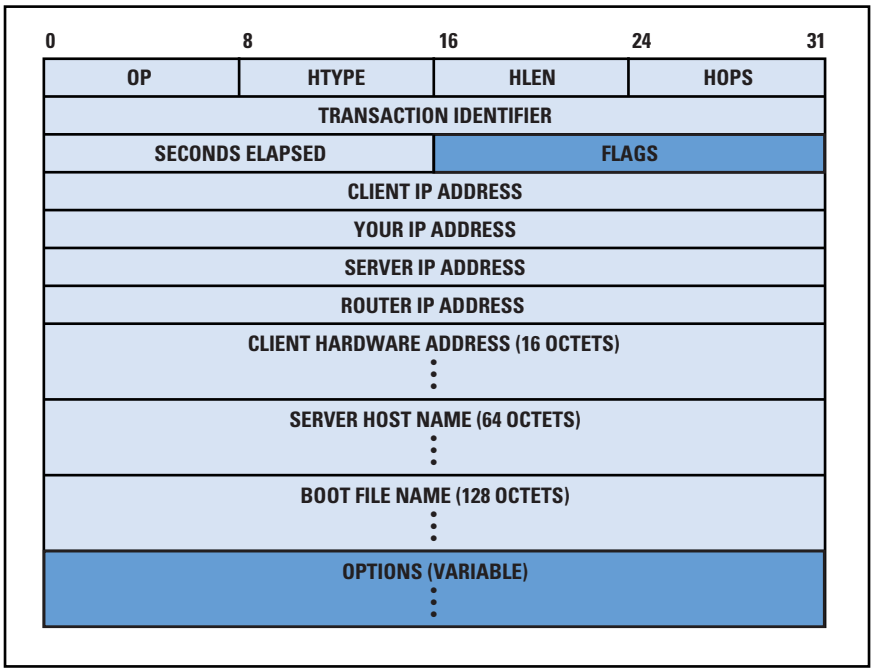
If the computers on a network use DHCP to obtain configuration information when they boot, an event that causes all computers to restart at the same time can cause the network or server to be flooded with requests. To avoid the problem, DHCP uses the same technique as BOOTP: each computer waits a random time before transmitting or retransmitting a request.

The DHCP protocol has two steps: one in which a computer broadcasts a *DHCP Discover* message to find a DHCP server, and another in which the computer selects one of the servers that responded to its message and sends a request to that server. To avoid having a computer repeat both steps each time it boots or each time it needs to extend the lease, DHCP uses *caching*. When a computer discovers a DHCP server, the computer saves the server's address in a cache on permanent storage (for example, a disk file). Similarly, once it obtains an IP address, the computer saves the IP address in a cache. When a computer reboots, it uses the cached information to revalidate its former address. Doing so saves time and reduces network traffic.

DHCP Message Format

Interestingly, DHCP is designed as an extension of BOOTP. As Figure 2 illustrates, DHCP uses a slightly modified version of the BOOTP message format.

Figure 2: DHCP Message Format



Most of the fields in a DHCP message have the same meaning as in BOOTP; DHCP replaces the 16-bit *UNUSED* field with a *FLAGS* field, and uses the *OPTIONS* field to encode additional information. For example, as in BOOTP, the *OP* field specifies either a *Request* or a *Response*. To distinguish among various messages that a client uses to discover servers or request an address, or that a server uses to acknowledge or deny a request, DHCP uses a *message type option*. That is, each message contains a code that identifies the message type.

DHCP and Domain Names

Although DHCP makes it possible for a computer to obtain an IP address without manual intervention, DHCP does not interact with the Domain Name System. As a result, a computer cannot keep its name when it changes addresses. Interestingly, the computer does not need to move to a new network to have its name change. For example, suppose a computer obtains IP address **`192.5.48.195`** from DHCP, and suppose the domain name system contains a record that binds the name **`x.y.z.com`** to the address. Now consider what happens if the owner turns off the computer and takes a two-month vacation during which the address lease expires. DHCP may assign the address to another computer. When the owner returns and turns on the computer, DHCP will deny the request to use the same address. Thus, the computer will obtain a new address. Unfortunately, the *Domain Name System* (DNS) continues to map the name **`x.y.z.com`** to the old address.

For several years, researchers have been considering how DHCP should interact with the DNS. Although a dynamic DNS update protocol has been defined, it has not been widely deployed. Thus, many sites that use DHCP do not have a mechanism to update a DNS database. From a user's perspective, the lack of communication between DHCP and DNS means that when a computer is assigned a new address, the computer's name changes.

Summary

The bootstrapping sequence loads a generic image into a computer, either from secondary storage or over the network. Before application software can use TCP/IP protocols, the image must be configured by supplying values for internal parameters such as the IP address and subnet mask, and for external parameters such as the address of a default router; the process is known as *configuration*. Initially, separate protocols were used to obtain each piece of configuration information. Later, the *BOOTstrap Protocol,* BOOTP, was invented to consolidate separate requests into a single protocol. A BOOTP response provides information such as the computer's IP address, the address of a default router, and the name of a file that contains a boot image.

The *Dynamic Host Configuration Protocol* (DHCP) extends BOOTP. In addition to permanent addresses assigned to computers that run a server, DHCP permits completely automated address assignment. That is, DHCP allows a computer to join a new network, obtain a valid IP address, and begin using the address without requiring an administrator to enter information about the computer in a server's database. When DHCP allocates an address automatically, the DHCP server does not assign the address forever. Instead, the server specifies a lease during which the address may be used. A computer must extend the lease, or stop using the address when the lease expires.

For Further Study

Details about BOOTP can be found in reference [1], which compares BOOTP to RARP and serves as the official protocol standard. Reference [2] tells how to interpret the vendor-specific area, and reference [3] recommends using the vendor-specific area to pass the subnet mask. Most uses of BOOTP have been replaced by DHCP. Reference [4] contains the specification for DHCP, including a detailed description of state transitions. A related document, [5], specifies the encoding of DHCP options and BOOTP vendor extensions. Finally, reference [6] discusses the interoperability of BOOTP and DHCP. The chair of the DHCP working group, Ralph Droms, and Ted Lemon have written a book about DHCP [7].

References

[1] W. J. Croft, J. Gilmore, "Bootstrap Protocol," RFC 951, September 1985.

[2] J. K. Reynolds, "BOOTP Vendor Information Extensions," RFC 1084, December 1988.

[3] R. Braden (ed), "Requirements for Internet Hosts—Application and Support," RFC 1123, October 1989.

[4] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, March 1997.

[5] S. Alexander, R. Droms, "DHCP Options and BOOTP Vendor Extensions," RFC 2132, March 1997.

[6] R. Droms, "Interoperation between DHCP and BOOTP," RFC 1534, October 1993.

[7] R. Droms and T. Lemon, *The DHCP Handbook: Understanding, Deploying, and Managing Automated Configuration Services,* ISBN 1578701376, MacMillian, 1999.

[This article is adapted from *Computer Networks and Internets, with Internet Applications, 3rd edition,* by Douglas Comer, with CD by Ralph Droms, ISBN 0130914495, Prentice Hall, 2001.]

Dr. DOUGLAS COMER is a professor of Computer Science at Purdue University, consultant to industry, and an internationally recognized authority on TCP/IP. He has written numerous research papers and textbooks, including the classic three-volume reference series *Internetworking with TCP/IP,* and currently heads research projects. He designed and implemented X25NET and Cypress networks, and the Xinu operating system. He was a principal on the CSNET project, is director of the Internetworking Research Group at Purdue, editor of the journal *Software—Practice and Experience,* a former member of the IAB, and a Fellow of the ACM.
E-mail: **comer@cs.purdue.edu**

Book Review

The Elements of Networking Style

The Elements of Networking Style, by M. A. Padlipsky, originally published by Prentice-Hall, 1985, ISBN 0132681110; now available from iUniverse, 2000, ISBN 0595088791.

Sometime in the autumn of 1986, I read Padlipsky on a flight from Boston to San Francisco, and about 15 minutes into it I began to get enraged. A few minutes later, I was snickering. By the time the attendants came around with profferings of alleged comestibles, I was laughing aloud, and a gentleman sitting near the window was grateful that there was a vacant seat between us.

Padlipsky brought together several strands that managed to result in the perfect chord for me over 15 years ago. I reread this slim volume (made up of a Foreword, 11 chapters (each a separate arrow from Padlipsky's quiver) and three appendixes (made up of half a dozen darts of various lengths and a sheaf of cartoons and slogans) several months ago, and have concluded that it is as acerbic and as important now as it was 15 years ago.

The instruments Padlipsky employs are a sharp wit (and a deep admiration for François Marie Arouet), a sincere detestation for the ISO Reference Model, a deep knowledge of the *Advanced Research Projects Agency Network* (ARPANET)/Internet, and wide reading in classic science fiction.

Arouet is better known by his pen name, Voltaire. He was a social rebel, a political agitator, and an acerbic satirist comparable to Swift. Isaiah Berlin, in a lecture published in Salmagundi 27 [1974], remarks:

> "Voltaire is the central figure of the Enlightenment, because he accepted its basic principles and used all his incomparable wit and energy and literary skill and brilliant malice to propagate the principles and spread havoc in the enemy's camp. Ridicule kills more surely than savage indignation..."

Padlipsky is pungent and sharp and witty ... and knowledgeable. His critiques of X.25, of the *International Organization for Standardization* (ISO) seven-layer cake, and of the standards process in general, are still relevant.

History

In the early 1970s, the CCITT (now the ITU), made up of PTTs and monolithic telcos, fixed upon a putative standard for a network interface protocol, X.25. First approved in 1976, and revised in 1977, 1980, 1984, 1988, and 1992, X.25 was unsatisfactory in its original form and remains less than effective.

One of the greatest drawbacks is that it is basically a store-and-forward mechanism, meaning that it has an intrinsic delay and (as noted by Sangoma Technologies) this delay is typically 0.6 seconds. It also requires a great deal of buffering space.

Padlipsky's "Critique of X.25" (Mitre Corporation Report, M82-50, September 1982; RFC 874 12 August 1983) is revised as Chapter 9 in *The Elements of Networking Style*. Padlipsky has restored, however, his original title: "Low Standards."

Flush with the failure of X.25, the *Consultative Committee for International Telegraph and Telephone* (CCITT) moved ahead.

In 1977, the British Standards Institute proposed to ISO that an architecture was needed to define the communications infrastructure. To me, this, as with *International Federation for Information Processing* (IFIP), CCITT, and similar efforts, shows how "the road to hell is paved with good intentions." Because X.25 was unsatisfactory, the IFIP Working Group was set up in the hope that that the technological community could forestall the highly political arena of ISO. (It didn't.)

ISO set up a technical committee [ISO/TC 97/SC 16]. The next year (1978), ISO published its "Provisional Model of Open Systems Architecture" [ISO/TC 97/SC 16 N 34]. This was labeled a "Reference Model," and referred to as the *Open Systems Interconnection Reference Model* (OSIRM or ISORM—pronounced "eye-sorm"—by Padlipsky).

In general, it was based on work done by Mike Canepa's group at Honeywell Information Systems, which came up with a seven-layered architecture, which itself owed a great deal to IBM's proprietary *Systems Network Architecture* (SNA). SNA had been announced in 1974, and its seven layers do not correspond exactly to OSI/ISORM's. TC 97/SC 16 turned over proposal development to the *American National Standards Institute* (ANSI), to which Canepa and his technical lead, Charlie Bachman, presented their layered model.

This, in turn, was the only proposal presented to the ISO subcommittee at a meeting in Washington in March 1978. It was accepted and published immediately. A "refined" version of the ANSI submission to ISO appeared in June 1979. This published version is nearly identical to Honeywell's of 1977.

Rage and Ridicule

While he eschews the history I've outlined here, Padlipsky is enraged by the standards process and its results. As Dave Walden and Alex McKenzie (both then at BBN, both now retired) pointed out in 1979, both virtual circuit and datagram services are valuable. "An international standard would do well to support both." [*IEEE Computer*, September 1979].

The 1977–1979 models were such that extant host-host protocols did not fit ISORM. ISO was trying to construct a set of geometric figures that would be a "tidy model." The ARPANET workers, of whom Padlipsky was one, were interested in getting things to actually work. They were into pushing bits around the system.

The irascible Padlipsky has described the OSI system as two high rises with parking garages. The two high-rises are seven-story buildings; the parking garages are the three-story X.25 structures.

John Quarterman once pointed out:

> "OSI specified before implementation. So specification took forever and implementation never happened, except for bits and pieces. In addition, heavy government backing (by the EC, now the EU, and various national governments) led some OSI participants to attempt to substitute official authority for technical capability. OSI and TCP/IP started at about the same time (1977). OSI wandered off into the weeds and TCP/IP won the race. Those governments that backed OSI bet on the wrong horse."

TCP/IP had clearly "won the race" by the early 1980s; it took till 1994 for the U.S. government to recognize the de facto standard by rescinding its *Federal Information Processing Standards* (FIPS). At that time, too, the *Defense Data Network* (DDN) was made up of IP router nets, not X.25-based nets.

In a totally different vein, there's Chapter 11: "An Architecture for Secure Packet-Switched Networks" (based on a presentation to the Third Berkeley Workshop on Distributed Data Management and Networking, August 1978). Here, Padlipsky suggests per-host processes. It was a really good notion.

Padlipsky's rants—and many of the chapters are just that—precede Quarterman's remarks by nearly a decade. But they are worth reading (and rereading).

I'm glad *The Elements of Networking Style* is available again.

—Peter H. Salus
peter@matrix.net

Call for Papers

The Internet Protocol Journal (IPJ) is published quarterly by Cisco Systems. The journal is not intended to promote any specific products or services, but rather is intended to serve as an informational and educational resource for engineering professionals involved in the design, development, and operation of public and private internets and intranets. The journal carries tutorial articles ("What is…?"), as well as implementation/operation articles ("How to…"). It provides readers with technology and standardization updates for all levels of the protocol stack and serves as a forum for discussion of all aspects of internetworking.

Topics include, but are not limited to:

- Access and infrastructure technologies such as: ISDN, Gigabit Ethernet, SONET, ATM, xDSL, cable, fiber optics, satellite, wireless, and dial systems
- Transport and interconnection functions such as: switching, routing, tunneling, protocol transition, multicast, and performance
- Network management, administration, and security issues, including: authentication, privacy, encryption, monitoring, firewalls, trouble-shooting, and mapping
- Value-added systems and services such as: Virtual Private Networks, resource location, caching, client/server systems, distributed systems, network computing, and Quality of Service
- Application and end-user issues such as: e-mail, Web authoring, server technologies and systems, electronic commerce, and application management
- Legal, policy, and regulatory topics such as: copyright, content control, content liability, settlement charges, "modem tax," and trademark disputes in the context of internetworking

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

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

Fragments

Stephen D. Crocker Receives 2002 IEEE Internet Award

The *Institute of Electrical and Electronics Engineers* (IEEE) has named Stephen D. Crocker, chief executive officer of Shinkuro, Inc. in Bethesda, Md., as recipient of the 2002 *IEEE Internet Award*. The award recognizes Crocker for his leadership in the creation of key Internet protocols. It will be presented on 19 June, at INET 2002, in Arlington, Va.

In the formative days of the Internet and its predecessor, the ARPANET, Crocker led the development of crucial technologies, processes and organizations that continue to support the Internet today. At the University of California at Los Angeles, Crocker and his team developed protocols for the ARPANET such as the *Network Control Protocol*. NCP laid the groundwork for today's *Transmission Control Protocol* (TCP). Crocker also founded and led the *Network Working Group* (NWG), which has evolved to become the *Internet Engineering Task Force* (IETF).

In organizing the notes from the first few meetings of NWG, Crocker was anxious to expand the community and invite further discussion and responses, and thus named the series *Requests for Comments*. RFCs remain a mainstay of Internet protocol publishing today, and have played a big part in creating the environment of open and evolving standards of the Internet.

"The Internet Society is honored that INET 2002 was chosen as the venue to present this year's prestigious IEEE Internet Award," said Lynn St. Amour, president and CEO of the Internet Society (ISOC). "Dr. Stephen Crocker is highly regarded throughout the international Internet community and we're pleased that his contributions will be recognized at INET 2002 in front of his peers."

Crocker's many contributions to the Internet also include extensive work organizing the standards process of the IETF, where he has served as area director of security and on the Internet Architecture Board. Crocker previously worked for the University of Southern California Information Sciences Institute in Marina del Rey, the Aerospace Corporation in El Segundo, Calif., and at Trusted Information Systems, Inc., in Glenwood, Md. In 1994, he co-founded CyberCash of Reston, Va., and served as its senior vice president for development and chief technology officer. He also has started other ventures including Steve Crocker Associates in Bethesda, Md.; Executive DSL in Bethesda, Md.; and Longitude Systems in Chantilly, Va.

He has served on the Council of Visitors at the Marine Biological Laboratory, as part of the National Research Council Study of Information Systems Trustworthiness and currently chairs the ICANN Security and Stability Advisory Committee and the ISOC 2002 Jonathan B. Postel Service Award Committee. The author of numerous papers, Crocker also holds patents in relation to his security and electronic commerce work.

He received his bachelor's degree in mathematics and doctoral degree in computer science, both from UCLA, he and studied artificial intelligence at the Massachusetts Institute of Technology.

The IEEE is the world's largest technical professional society with more than 377,000 members in approximately 150 countries. Through its members, the IEEE is a leading authority on areas ranging from aerospace, computers and telecommunications to biomedicine, electric power and consumer electronics. Additional information is available at **http://www.ieee.org**

The Internet Society **http://www.isoc.org/** is a non-profit, non-governmental, open membership organization whose worldwide individual and organization members make up a veritable "who's who" of the Internet industry. It provides leadership in technical and operational standards, policy issues, and education. ISOC is the organizational home of the International Engineering Task Force, the Internet Architecture Board, the Internet Engineering Steering Group, and the IETF—the standards setting and research arms of the Internet community. For information about INET 2002 please visit **http://www.inet2002.org**

Interim Approval for ENUM Provisioning

The *International Telecommunication Union* (ITU) and the *Internet Architecture Board* (IAB) recently announced interim approval for a single domain for ENUM, a technology that builds a bridge between the public switched telephone network and the Internet.

Voice on IP networks today operate by translating telephone numbers to IP addresses and placing an H.323 or SIP call to the device. The interchange format and translation record has not heretofore been standardized, limiting the possibility of deployment of multi-corporate and international Voice on IP services. Under the ENUM proposal, E.164 numbers can be represented as Internet Domain Names,providing a scalable and standard way to translate the numbers, and opening the way to such services. ITU has begun approving delegations for the purposes of trials. "The lack of an interoperable standard way to turn a telephone number into an IP Address has been one factor limiting the deployment of Voice on IP services internationally," said Leslie Daigle, Chair of the IAB.

If desk-mounted computers or servers are given telephone numbers as well as mnemonic names, this system further enables common telephone handsets to place Voice or Video on IP calls to such computers. This is a significant step towards integrating Internet-based services with the global telephone network, and the current agreements between IAB and ITU will allow trials to take place.

Patrik Fältström, member of the *Internet Engineering Steering Group* (IESG), said that "the integration of the desktop telephone and computer allows corporations to simplify their internal networks."

Roy Blane, Chair of ITU-T's Study Group 2, concurred, saying that "In the long term this protocol may facilitate many new internet services. In the short term, countries wishing to trial the system can begin work on developing it."

This interim approval is made possible due to cooperation between ITU, IAB and the IETF. As outlined in the ENUM specification document, RFC 2916, sub-domains from a single domain will be delegated after acceptance by the registries according to the existing assignment of country codes in the telephone address space. Information on how the ENUM registration requests will be processed can be found at: **`http://www.ripe.net/enum/`**

The IETF is an international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. The definition of the ENUM protocol, as proposed by the IETF can be found at **`http://www.ietf.org/rfc/rfc2916.txt`** The IETF is an organized activity of the Internet Society.

The ITU is a global organization where the public and private sectors cooperate for the development of telecommunications and the harmonization of national telecommunications policies. Study Group 2 of the *ITU Telecommunication Standardization Sector* (ITU-T), where work on ENUM is being carried out, is the Lead Study Group on Service definition, Numbering, Routing and Global Mobility and is responsible for the operational aspects of service provision, networks and performance. More information on the ENUM protocol, and the issues related to it, can be found at **`http://www.itu.int/ITU-T/worksem/enum/index.html`**

Committee on ICANN Evolution and Reform posts Recommendations

Following the publication in February of "President's Report: ICANN—The Case for Reform," by Stuart Lynn, President and CEO of *The Internet Corporation for Assigned Names and Numbers* (ICANN), a committee of the board has been examining the details of the restructuring proposal, receiving input from the community at large, and publishing several documents with recommendations. You can find pointers to all of these documents in the "Announcements" section at **`http://www.icann.org`**

Upcoming Events

INET 2002, the annual conference of the Internet Society, will be held June 18–21, 2002 at the Crystal Gateway Marriott, in Arlington, Virginia (5 minutes from downtown Washington, DC).
http://www.inet2002.org/

The *IETF* will be meeting in Yokohama, Japan, July 15–19, 2002 and in Atlanta, Georgia, USA, November 17–22, 2002.
http://www.ietf.org/meetings/meetings.html

ACM SIGCOMM 2002 is the annual conference of the *Special Interest Group on Data Communication* (SIGCOMM), a vital special interest group of the *Association for Computing Machinery* (ACM). This year, SIGCOMM will be held in Pittsburg, Pennsylvania, August 19–23.
http://www.acm.org/sigcomm/sigcomm2002/

ICANN will meet in Bucharest, Rumania, June 24–28, 2002 and in Shanghai, China, October 27–31, 2002.
http://www.icann.org/meetings/

The *Asia Pacific Network Information Centre* (APNIC) will hold its next Open Policy Meeting, September 3–6, 2002 in Kitakyushu, Japan. **http://www.apnic.net/meetings/index.html**

The next *Asia Pacific Regional Internet Conference on Operational Technologies* (APRICOT) will take place February 19–28 in Taipei, Taiwan. **http://www.apricot2003.net/**

Errata List

This is the 17th issue of *The Internet Protocol Journal*. Inevitably, some minor, and a few major errors have made their way into print since our June 1998 issue. We are planning to publish a list of correctionson our Web site in the near future. Since the online material is a reflection of the printed version, we feel it would be inappropriate to simply "silently" correct the online editions, thereby rewriting history. Instead, a list of the errors along with the corrections will be presented.

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.