KOBE – How It Works: DNS Fundamentals
Saturday, March 09, 2019 – 08:45 to 10:15 JST
ICANN64 | Kobe, Japan

CATHY PETERSEN: Good morning, everyone! Welcome to How It Works, DNS Fundamentals. This session will be presented by Matt Larson, our Vice President from the office of the CTO. Thanks.

MATT LARSON: Thank you, Cathy. Good morning, everyone. So, these slides are a brief introduction to, just as the title of the session says, how DNS works. So, let's get right into it.

Let me tell you a little bit about the motivation for DNS in the first place. IP addresses are easy for machines but hard for people. It's possible to remember IPv4 addresses. I know everybody knows what those look like. There's an example. It's essentially impossible for a normal human to remember IPv6 addresses, so IPv6 makes it even more important to have a way for humans to use names to refer to machines and other resources on the Internet, but the underlying devices need IP addresses and that's what DNS does for us.

So, in the early days of the Internet, when it came to naming devices, that was completely different than it was today. This was before domain names. There were no domain names. Names were simple. They're what we would now call single-label names. So, no dots in the name. The maximum length was 24 characters. And these were called host names.

The way it worked when a device, a computer, something, needed to translate a name into an IP address, that is what we call name resolution. And this was done in an extremely simple way using something called a host file. There was literally a file – and it was literally a plain text file that you could look at, human readable, you could edit it in a text editor and this was called host.text, and it has the same function but a slightly different format than what you would now know as the /etc/ host file on Unix or Linux.

This file literally had a line with the name and IP address of every device on the Internet at the time, and of course the Internet was much, much smaller so it was conceivable that you could have a file that had the name and address of every device on the Internet.

This file was centrally maintained by an organization called the NIC, the Network Information Center. They had a US government contract to do this. Network administrators from around the Internet, when they had a change that they needed to the file – namely, when they added a machine or removed a machine or changed the machine's name or IP address – they would send an e-mail to the NIC and say, "Hey, you need to make this change to the host file because I just made this change on my network."

Then, once a week, NIC released a new version of the file and people could use FTP, the file transfer protocol, to come in and grab a new version of the file whenever they felt that theirs might be out of date. So, this is an extremely manual, extremely low-tech solution to the update mechanism.

There were obvious problems to this, which you can imagine if you just think about it for a moment. The first of these was naming contention. So, the edits to this file were made by hand. There was no good way to prevent duplicates. There was no database in the backend. Nowadays, if you're going to do something like this, you'd have a database and if you needed a text version of the file, you'd have some program that would run that would extract the database and emit a text file. Well, that wasn't how it worked then. They were literally editing this file in a text editor and there was no good method to prevent duplicates.

And because we're talking about 24 characters, that's a limited, what we call a name space, for everyone to get along with on the entire Internet. You had to name your machines on your network in the same 24-character name space as everybody else on the Internet. So, duplicates began to be a problem.

Synchronization was also a problem. Nobody ever had the same version of this file. As the Internet grew and the file grew, you were never up to date. It was like printed phone books, right? We don't have printed phone books that much anymore, but when we did, the instant you got one it was already out of date because there were already people whose numbers that had changed.

Finally, the traffic in load, just to move this file around began to be significant. It took a significant portion of the bandwidth of the Internet at the time simply to move this monstrous growing file around. Clearly this just didn't make sense. Essentially maintained host file with the name and IP of every device on the Internet, it just didn't scale. It didn't make sense anymore.

So, discussion started in the Internet engineer community in the early 1980s on a replacement. There were a couple of goals. One was to address the scaling issues I've just talked about and another one was to simplify email routing and I'm going to talk a little bit about this later, so I'm going to hold off on that.

But until DNS, your e-mail address was tightly coupled to the physical machine that you read your e-mail on and DNS provides a level of indirection and lets you have an e-mail address that is independent of the physical machine where your mail is. Again, I'll talk about that coming up in a moment.

But the result of this discussion is we now know is the domain name system. That was the replacement for host.text.

This is my one slide, super high-level summary of the domain name system. Fundamentally, DNS is a distributed database, and in this distributed database, the data is distributed over at the entire world. It's the largest distributed database there is. It's been a fantastic success. It's scaled far, far beyond what it's creators I think could ever have imagined.

In this database, data is maintained locally, so everyone maintains their own named address mapping information. You maintain that yourself locally. But it's available globally. Anybody all over the Internet can look up the data that you're responsible for and you can look up data that other people are responsible for.

DNS file is the client server model and clients are called resolvers. The main thing to remember about a resolver is the resolver sends queries. That would be a query being a question, a DNS question. And name servers of the client side – and the main thing to remember about resolvers is that they answer queries.

Now, there are some important optimizations that make DNS work. One of these is caching. Caching improves performance. Because this is a distributed database, doing a lookup in this database can take a significant amount of time, even in human terms. Not just in computer terms, but in real human eyeball terms. And that's because we're talking about a database distributed over the entire world and the speed of light is only so fast, so you could imagine doing several lookups halfway around the world and a few milliseconds here, a few milliseconds there, pretty soon you're adding up to half a second, a second, and those are actual noticeable delays for real people.

So, it's important that DNS builds in caching, the ability to remember information that you've looked up previously, so you don't have to look it up again. And that includes not only the final answer for what you looked up but any intermediate answers along the way that helped you get there. So, that all speeds up the resolution process.

DNS also uses replication to provide redundancy and load distribution. What I mean by that is, in this distributed database, data is not located just in one place. You have multiple copies, multiple replicas, of the data. And this provides redundancy, right? If all your name to address information were in one place on one machine and somebody tripped over the power cord and that machine goes off the air, then nobody can look up your name to address information. But because DNS uses replication, there's by design multiple copies of that information so you can tolerate a failure like that.

And then it also spreads the load. If you run a popular domain and a lot of people are looking up your name to address information, then you would have a lot of DNS queries to answer, and if you have multiple replicas that can answer the queries, that helps with the performance.

So, here's a high-level picture. I think a picture at this point really helps put all these components together and explain how everything works together.

So, let's start at the lower left. There we have a device that happens to be a phone, but any device that uses the Internet that needs to turn domain names into IP addresses. Any device is going to be a DNS claim. And it has a very simple client in it called a stub resolver.

So, remember, resolvers are DNS clients and there's actually two kinds of resolvers, as I'm showing on this slide. And the very simplest kind that's everywhere is called a stub resolver. Literally everything connected to the Internet that is going to need DNS has one of these stub resolvers in it. They tend to be very, very simple pieces of code and they tend to be supplied by the operating system on the device. They're a service that the device needs to provide, applications that run on the device.

So, for example, on this slide, I'm showing that iPhone and I'm showing the icon for the Safari web browser. So, let's say that somebody types in a domain name in that web browser. Well, the web browser needs to turn that name into an IP address of a website so it can go contact it. So, the application calls the stub resolver and the orange there, the orangish color, that represents an API call. That represents a programmatic call. This is the web browser program talking to another piece of code on the machine, a stub resolver.

So, the stub resolver's job is very simple. Its job is to take an applications request for data, like an application like the web browser is saying, "Hey, here's a domain name. I need the IP address." The stub resolver takes that and turns it into a DNS query and it sends it over the network, and it sends it to something called a recursive resolver and that's what we have in the top center of the screen.

So, recursive resolver is more complicated. And this is actually part name server and part resolver, if you will. Remember, I said name servers answer queries. So, a recursive resolver has a name server component in that it answers queries from stub resolvers. But it also has this more complicated resolver component and this resolver knows how to go out and contact various different sources of DNS data on the Internet and look up a query, look up an answer. So, those are what we call authoritative name servers.

And authoritative name servers, their job is to hold DNS data and wait for resolvers to ask them questions and then answer the questions and that resolver, the recursive resolver in the center of the screen, it knows how to chase down the answer to a query. So, it might go to one authoritative server and say, "Hey, here's the piece of data I'm looking for," and that authoritative server might say, "Well, I don't have the exact answer you're looking for, but I can refer you to another set of authoritative servers that can help you out," and then the recursive resolver would contact them and it might get referred multiple times. And we'll talk about this in more detail how this works, but this is to try to show you a high-level picture of how that works.

So, eventually, that recursive resolver finds the answer it's looking for. See, I've got a cache shown there in the oval that represents the idea that the recursive resolver is remembering all this information that it's looked up to speed up future look-ups. Then, eventually, it returns that answer to the stub resolver and the stub resolver returns the answer to the web browser and we're off and running, looking up the web page.

So, these are all the components that work together to make DNS work from a technical perspective.

Let me define a couple of important terms for you here. The first of these is what we call the name space. So, I've said that DNS is a distributed database and the structure of this distributed database is an inverted tree. So, this is a computer science term. An inverted tree is the opposite of a regular tree. A regular, botanical tree you have the roots at the bottom and the branches grow upward. But in a computer scientist tree, an inverted tree, it's the reverse. The root is at the top and the branches grow downward.

So, this is the structure of the DNS database, what we call the namespace. If you contrast this to say if you know a little bit about databases, you may know about, say, relational databases and the structure of a relational database is you have multiple tables, and in each table you have rows, and there are different kinds of data in each row in what we call columns. So, that's the structure of a relational database.

But the structure of the DNS database is this inverted tree that we call the namespace. So, in this inverted tree, in the namespace, we have these nodes represented with the rectangular boxes here. Each node has a label which is just a string of characters that identifies that node and the root node at the very top is special. It has actually a null label. It has a label, but the label is that it has no label. That's a little mind-bending but that's how it works.

Every other label, every other node in the name space has a label and the legal characters for these labels are letters, digits, and the hyphen and the maximum length of these is 63 characters and case does not matter. Upper case and lower case are all compared equivalently.

I want to go back a slide because there's one point that I wanted to make and I didn't. If you look at the right, you see I show the different levels of nodes in the name space. We often talk about these nodes in relation to where they are compared to the root. So, the root is at the very top and immediately below the root are what we call nodes at the top level, because remember, the top of this tree is the root. And then below that we have the second level and third level and so on.

You also sometimes use parent/child terminology. You'll talk about one node being the parent of the other, like the root node is the parent of dot-UK in this diagram and UK is the child of the root.

So, that's the syntax for these labels. And this gets us to the point where we can talk about domain names. Every one of these nodes in the name space, it has a label name, but for example, we have several nodes on the screen that are labeled www. So, a node's label name by itself doesn't tell you where the node is in the name space. What you need is the full name for that node. And that's what we call the domain name and every node has a domain name and it's really straightforward to make that node's domain name. You just start at the node and you read off its label name and you put down a dot and then you go to its parent and you read that label name and you put down a dot and so on until you get to the root and that's how you construct the domain name for any node.

So, for example, the highlighted node there, as you can see, reading upward is www.example.com. Now, if you imagine you're writing label names and dots, you eventually get to the top-level label, in this case com, and you write a dot at the end and then you get the root zone label, but there is no – the root is just that. It's a null label. There's nothing to write for the root. So, you end up with a domain name that ends in a dot and a domain name ending in a dot is what we call a fully qualified domain name and that unambiguously identifies a node in the name space. So, there's only one fully qualified domain name for every node and every node has only one fully qualified domain name.

So, then, let's talk about domains. A domain is simply a node in the name space and everything below it. So, here I've highlighted the dot-com domain. Obviously, a tiny portion of the dot-com domain. But the idea here is that we have the node dot-com and that node and everything below it is the dot-com domain.

Now we get to a really important concept and that's the concept of a zone. The whole reason that we have the DNS in the first place is that centralized administration, putting everything in one place in that host file, that didn't work. We realized we have to distribute administration. Everybody has to be responsible for their own portion of the name space so that they can administer it on their own. In other words, administration is distributed. So, this name space, the database that makes up DNS, has to be divided up to allow distributed administration and different people are responsible for different portions of it and those divisions are called zones. And this process of delegation creates zones and we have – you delegate from a parent to a child. This is much easier to explain with a picture.

So, here we have another picture of the same name space that I've been showing this little excerpt of it. So, zones are administrative boundaries. So, I've shown here what the boundaries around some zones might be and there's no way to know what these boundaries are just by looking at the namespace. You have to look at the information in DNS to tell you where the zone boundaries are.

At the very top of the name space, we have the root zone and it delegates, we call it, two zones below it which are at the top level and those are called top-level zones. Then, below that, top-level zones delegate to second level zones and so on. So, the arrows I've got represent the delegation going from a parent to a child.

So, if we go back to name servers, remember I said the job of a name server is to answer queries and a name server that we call authoritative [for a] zone has complete knowledge about the zone. The idea is that every zone has a set of name servers that know about information in that zone and every zone should have multiple authoritative name servers and this is where you get redundancy and you spread the load, as I mentioned earlier, how DNS uses replication.

So, if you're going to have multiple authoritative servers for a zone, if you're going to have you zone's information in multiple places, you need a way to synchronize that. You need a way to keep all of the copies in synch. Fortunately, DNS provides that. There's something called a zone transfer. I have some details on the slide here that I won't read out but the point here is that DNS has built into it a replication protocol so that you can have your zone maintained in one place and have all of the copies stay synchronized.

So now let’s look inside a zone. I’ve talked about a zone at a high level, the idea that the whole DNS name space is divided up into these different regions that can be administered separately called zones, but what’s in a zone?

Well, every node has a domain name, remember, and we can associate different kinds of data with that domain name and this data is what we call – it’s stored in what we call research records and there are different types of research records for different kinds of data.

A zone, then, is nothing more than the sum of all its resource records and all the resource records in a zone are stored in what we call the zone file and every zone has at least one of these zone files.

Each resource record has five fields that I’ll talk about here. There’s actually a standard space way to write down this information in text format and I will go ahead and just give some examples here, again, rather than dwell on what’s on the slide.

So, these are some common types of resources records. There are actually – I can’t remember, do I have ….? Yes, I do have. Actually, as of late 2017, there were 84 different types of resource records. So, those were all the different types of data that you could put in DNS. These are the most common resource record types.

So, by far, the most common is what we call an address record or the Quad A record for four As and those are records that represent IPv4 and IPv6 addresses. So, this is arguably the main purpose for DNS, to map domain names to IP addresses. So, these two types of records, the A record and the Quad A record, they're the work horse of DNS. They're really main reason we have DNS.

But DNS isn't limited to mapping names to IP addresses. You can map names to any other types of information and you can create new record types if you can think of new thing that you want to store in DNS.

Now, there are other uses for DNS other than name to address mapping. There's nothing that's been as fantastically popular and successful as the main purpose of DNS, namely name to IP address mapping. But the point is that you can use it for other things.

So, as I said, here are the most common types of resource records and I'm going to go through and describe them one by one here.

So, here is the … There is an IANA registry that describes the different types of records. There's the domain name for it here if you wanted to go look at all the different types of resource records that are officially on the books.

But let's take these one at a time. So, the first of these would be the two kinds of address records and this is how you map a name to an IP address. So, here's the actual text-based representation for what they would look like in a zone file. You have a domain name and then the type of the record and then the actual data for that record. In this case, an IP address.

So, that first one simply says that the domain name example.com has the IPv4 address 192.0.2.7. So, that would be in a zone file somewhere. An authoritative server would load that zone file and know that that name maps to that address and a recursive resolver could look it up.

We also have the Quad A record which stores an IPv6 address and that second record shows that the example.com domain name maps to that particular IPv6 address.

Now, there's an analogy I like to use here. Most of the types in DNS are used by people who want to store information in DNS and look up that information but there are a few types that really are only used by DNS itself. So, some types are the whole reason we have DNS like A and Quad A, but then there are some types that just are needed to make DNS itself work. Nobody outside of DNS, nobody outside of clients, resolvers, and servers cares about these types but they've got to be there for everything to work.

so, if you compare this to a warehouse, imagine that you have a bunch of goods that you want to put in a warehouse, well, you don't just back your truck up and just throw everything into the warehouse. First, you have to get the warehouse ready. You have to set up, say, shelves. And once you get all the shelving in place, then you can take the goods that you care about that you want to store in the warehouse and you can put them on the shelves. And if we consider DNS this way, types like the NS type and the SOA type which I'm going to explain here in a moment, those are like the shelves of DNS. We don't really care about the shelves. They have to be there or nothing else works. But the things we care about would be like A records and Quad A records. That's the whole reason we have the warehouse, or in this case DNS, in the first place.

So, that being said, let me talk about one of these pieces of shelving, the name server record. Every zone has to have a place where we know where we say what the authoritative servers for the zone are and that's what this NS record – name server record – is for. So, these two records here, for example, they say that for the example.com zone, there are two authoritative name servers and they're named and it's example.com and there's two different example.coms.

So, on the left-hand side, then we have the name of a zone and on the right-hand side, we have the name of a name server.

Now, right away, this unfortunately gets kind of complicated because NS records actually appear in two places. The main purpose of the NS record is to mark a delegation, is to actually create a child zone. So, the NS records appear in the zone itself and then they also appear in that zone's parent zone and it's the NS records in the parent zone that actually delegate the child zone.

So, for example here, I'm showing the root zone and the dot-com zone and that red arrow. The red arrow is conceptually the delegation information. That's the information in the root zone that tells everyone that the dot-com zone exists.

So, in that box, those are the 13 NS records. Those are the actual 13 NS records for the dot-com zone and that just says that the dot-com zone has these 13 authoritative name servers.

So, those 13 NS records for dot-com, they actually appear in two places. They appear in the dot-com zone itself, but they also appear in the parent zone of the dot-com zone which in this case is the root – the parent is the one right above it in this name space tree.

This process works all the way down. I'm just showing that they're in both places, that the same set of records is in the parent, is in the child.

If we go down a level, let's for example look at the example.com zone on the lower right. So, the example.com zone might have the set of NS records that I show inside the box and that set of NS records is not only in the example.com zone, but it's also in that zone's parent, the dot-com zone and it's the existence of those NS records in the dot-com zone that tells everyone that the example.com zone exists and where to go to find information about it because the NS records, remember they tell us the names of the authoritative servers that we can go contact if we have a question we want to ask about example.com.

So, if you think about it, then, a zone like dot-com is really just full of NS records, because for every dot-com domain, there's a set of NS records that delegate that zone. So, the dot-com zone is basically filled with 130 million or whatever the number is these days, 130 million sets of NS records for each of the different domains underneath dot-com, each of what we call the second-level domains.

Then, every zone has one and only one of what we call these SOA or start of authority records. This type is rather complicated. If you look, so far I showed you A records and Quad A records that have just an IP address on the right side and NS records that have just a name server on the right side. But here we have a record that has seven different fields on the right side and most of these fields have to do with zone transfers and the zone transfers, that way that we synchronize information on all the authoritative servers.

So, I'm not going to go into all the information. I'm not going to go field by field into what all that means. I have it on the – oh, I don't have it on the next slide. I don't have an example. I've got to put that back. But again, most of these control relate to zone transfers.

One I will point out is that serial number. Every zone has a serial number which is kind of like a version number and that's how the different authoritative servers can track whether or not they have the most recent version of the zone or whether they're out of date because every time the zone changes, that serial number has to increase and that's how an authoritative server can tell, "Do I have the most recent version or do I have an older version and I need to get the most recent version?"

Now, another thing that DNS is used for is mail routing. And I mentioned this early on. One of the initial design goals for DNS, one was to handle all the scaling problems in host.text that we mentioned, but another one that I mentioned briefly was we need to simplify – needed to simplify mail routing.

So, here's the issue. At the time, e-mail addresses consisted of a username@ and a host name, because remember, back in the 80s, there were no domain names yet. So, your e-mail address was basically your user name @ and then the name of a physical machine. So, what that said was your e-mail went to the machine named in your e-mail address. So, that's important to point out. Your e-mail address literally consisted of your username@ and the name of a physical machine. That meant that you had to go to that machine. That's where your e-mail went.

Now, let's say that that corresponded to a department in a university. Let's say that the Computer Science Department had a particular machine that the Computer Science Department's e-mail went to. So, if you're in the Computer Science Department, your e-mail would go to that machine.

Well, let's say that you changed departments. Let's say that you changed to the Electrical Engineering Department and let's say that they have a different mail server and all the people in the Electrical Engineering Department have e-mail accounts on that server. Well, now, simply by changing departments, you're going to have to change your e-mail address, right? Because your e-mail address, the physical machine where your mail goes is part of your actual e-mail address. So, that was a real problem and people said we've got to have a level of indirection in here. We need to have a way for somebody's e-mail address to say where the e-mail goes, not exactly, not the physical machine, but have another way to specify the physical machine.

So, that's where mail routing and DNS comes in. So, what I've just described is what's on the slide. It described how, in the old days, you had no choice but to send the mail to the physical machine.

But, in DNS, we have what's called an MX or a mail exchange record and that lets us decouple the mail server from the actual e-mail address.

So, here's an example of that. Let's say we're talking about the mail for example.com. So, for any user at example.com, so if this is somebody's e-mail address, some user at example.com, in the old days, there would have had to have been one machine named example.com that all the mail would have gone to, but now with MX records, we can have this level of indirection. What we can say is these MX records say, well, for any mail to example.com, it should really go to a server named mail.example.com and MX records have this concept of a preference value. That's the number, the 10 and the 20, and counterintuitively lower is more preferable. But you can not only say here's where the mail goes but you can have a backup. You can say if the main place for my mail, for my domain, is not available, if that mail server is down or whatever for whatever reason not working, the mail can go somewhere else so that it can at least be delivered and be there until the primary mail server is back up.

So, nowadays we just take MX records for granted, but this was a big deal back when DNS was invented. This finally let you have an e-mail address that was independent of the physical machine where your mail went.

Another thing that I want to talk about is called reverse mapping. So, everybody, when they think of DNS and they think of host names and they think of name resolution, they probably think of mapping a domain name to an IP address. This is what we call forward mapping.

But there are times when you want to do the opposite, when you have an IP address and you want to know what's the domain name that corresponds to this IP address.

Now, by far – again, remember, the main thing that we worry about most of the time is name to IP. I want to go to the name of a website. I want to type in a URL that has a domain name in it for a website. I want to turn that domain name into an IP address so I can get to the website. I want to send somebody an e-mail, so I need to turn the e-mail address into a mail server name and ultimately into an IP address.

But there are times when you want to do the opposite, when you want to look at an IP address and know what's the corresponding domain name? An example of that would be what if you are … Everybody's maybe used the trace route program which lets you see the path that packets take across the network. The trace route packet will show you the IP address of every router that a packet hits as it moves across the network.

Well, wouldn't it be nice if instead of just seeing the IP address of the router, we could turn those IP addresses into domain names and then you could see the corresponding names of the routers which would tell you the different organizations that they belong to? That's just an example of how it's, in some cases, nice to be able to map IP addresses back to domain names. And that's what we call reverse mapping.

Now, if you think about how this would work if you had a host table, as we did originally with the host.text file, if you have a file that has every name and IP address in it, it's easy to do forward mapping and reverse mapping. They're the same, right?

If you have a name and you want to know the IP address, you can just look through the file until you find a name and then there's the IP address. And if you have an IP address and you want to know the name, you just look through the file until you find the IP address and then you know the name. It's easy. In both cases, it's just a search through the file.

But, in DNS, it's more complicated than that because the name space … Let me go back a few slides to get a picture of the name space. Here we go.

So, this inverted tree that we made, this is designed for looking up names. You can imagine – and I'm going to describe this in just a moment. If you want to look up a name here, if somebody tells you, "I'm looking for information about www.example.com," well you can just start at the top of the tree at the root and you can work your way down and you can eventually get to www.example.com. You can get to that node and you could find the information you're looking for. That's exactly how DNS name resolution works and I'm going to explain it in a moment.

So, you can imagine, given a name and this data structure, it's really easy to look up that name. But now imagine you have this data structure and somebody says, "Well, I'll give you an IP address, 192.0.2.1. I want to know what the domain name is for that." Well, how do you even start, right? Here we have a data structure that lets us look up names, but not IP addresses.

So, DNS handles this in a rather interesting way. What it says is, well, since you can only look up domain names in DNS, if I want to look up an IP address, I have to make a domain name for it. So, the way this works is that there is a domain name that corresponds to every possible IPv4 address and to every possible IPv6 address.

So, if you have an IP address that you want to know its domain name for, you look in this special portion of the name space where all the IP addresses are. And there's a special kind of record called the PTR or a pointer record. This is kind of conceptually the opposite of an A record and of a Quad A record.

So, remember, an A record maps a domain name to an IP address. Quad A maps a domain name to an IPv6 address. And a PTR record maps a domain name to another domain name, but in this case the domain name is the special domain name.

Here's an example right here. This 7.2.0.192 to [inaudible], that corresponds to the IP address 192.0.2.7. Let me show this picture. That makes it maybe a little … It's a complicated thing and hopefully that makes it slightly less complicated.

So, on the right, we have example.com, the portion of the name space that I've been talking about for the past few minutes. But then on the left, we have the [inaudible] or dot-arpa name space and this is where all of the IPv4 address space has a domain name.

The way this all works is it's just a convention that everybody obeys. If you have IP addresses, let's say if you own a particular block of IP addresses and you want to make those IP addresses, if you want them to be able to map back into domain names, you are entitled to a zone that corresponds to that IP address space and you fill it with PTR records, and if anyone wants to look up the corresponding IP address, they construct the [inaudible] dot-arpa domain name, this one here on the slide, and they look it up and then they find the domain name.

I realize this is kind of a complicated thing that's hard to explain briefly without a whiteboard and a bunch of examples, but I include this just to show that reverse mapping is a thing. It's something that people want to be able to do with DNS. They want to not only map from names to IP addresses, but also from IP addresses back to names.

So, there are many more types of resource records than the one that I've shown and I have just here a few examples for some of the other pieces of data that you can map domain names to, other than just IP addresses. This is, again, just an example to show that DNS is used for other things.

Here is an example zone file. This would be the zone file for the actual example.com zone. These are all records that I've showed so far as I went type by type, but if you put them all together, this would be what a zone file would look like and this is a very typical zone file. This is a small zone file, but it has enough information in it – enough DNS information in it – to support, to tell us where the web server for the zone is, the domain is, as well as how to get e-mail to that zone, to that domain name. If you think about it, that's what most domain names on the Internet are used for, e-mail and web server. That's the use case for most domain names on the Internet. This then is a simple DNS zone that supports that.

So, the authoritative server, the authoritative name server, for the example.com zone would literally have this information, this file, on it and it would read this file to know what the information is in the example.com zone.

So, let me wind up by describing the resolution process, to describe how the different types of DNS clients, the stub resolvers and recursive resolvers, how they cooperate with authoritative name servers, the server side of DNS, to look up data anywhere in DNS.

So, when you're going to look up information in DNS, you always need these three parameters – a domain name, a class, and a type. We didn't talk about class. I went over that. I skipped over that. But class is always going to be IN for the Internet class. So, what you really have then is a domain name and a type.

There are two different types of DNS queries. The very simple stub resolvers that are on every single device connected to the Internet. A stub resolver sends a really simple type of query called a recursive query and what a recursive query says is I need you to give me the exact answer to this question. I need you to either answer the question completely or I need you to send me an error back and tell me that you can't do it.

But recursive resolvers are more complicated and they send a type of query called a non-recursive query and what that says is you can either give me the final answer to my question if you have it or you can give me partial information and I will work with the partial information.

So, this resolution process starts at the root zone. So, there's a set of servers of that are authoritative for the root zone at the top of the name space and they're called root name servers. So, if you start by contacting a root server, you can follow the information that it gives you and ultimately find a name anywhere in the name space. So, the root name servers are pretty important in that regard.

So, it's important to be able to contact a root name server. But there's no way to discover them. They have to be configured. So, every recursive resolver on the Internet does have to have the names an IP addresses of the root servers configured. And this is usually done by whoever you got your recursive resolver software from. For example, if this is on Linux, whoever packaged up your Linux distribution, they would have included the current list of root name server names and IP addresses. That's what we called a root [inaudible] file. This is the actual current list of root name servers. There are actually 13 names for root name servers and there are 13 IPv4 addresses and there are 13 IPv6 addresses.

However, it's not quite as simple as that. As I'm going to show, there are many more than 13 physical root servers. Administration of the root zone is kind of complicated. There are two organizations that cooperate to administer what's in the root. One is ICANN through its role as what we call the IANA functions operator and the other is Verisign that operates as what's called the root zone maintainer. So, that's the content of the root zone itself. That's maintaining what's in the root zone.

But then there are 12 different organization that operate the authoritative name servers for the root zone. So, this is a little unusual. Usually, when you think of a company – let's say ICANN. So, ICANN's zone is the ICANN.org zone and ICANN is responsible for all of the authoritative servers for ICANN.org. But here we have the zone, the root zone, and there's not one organization responsible for the authoritative servers. There are 12 different organizations. And here's that list of 12 organizations.

So, there are 13 letters that correspond to the 13 root server identities. There are 12 organizations because Verisign runs two. For complicated historical reasons, they run A and J. So, this group of organizations runs root servers and there's some complicated Internet history here, but the really, really brief version is that back when DNS was invented, it was important that these root servers be on networks run by people who had good networks with a lot of bandwidth and who understood how to operate a root server. So, these organizations were basically picked in the early days of DNS and they've stayed that way almost the same ever since. So, that's the 13 organizations that run – or the 12 organizations that run the 13 root server identities.

However, there are many, many more than 13 physical machines because of a technique called Anycast. This is a routing technique that allows a single IP address to appear in multiple places physically on the Internet. So, there are literally over 1,000 root server instances we call them. So, there are more root servers than there are any other kind of name server for any zone anywhere on the Internet.

So, the important thing is not to think of the 13 root server identities. The important thing is to remember that each one of these identities is located in, in some cases, hundreds of places so that we have literally over 1,000 different root server instances all over the Internet.

At a real high level, this is how the root zone gets updated. When the manager of a top-level domain has a change they want to make, they contact the IANA functions operator which is run by ICANN's subsidiary PTI and then that change, after getting verified, goes to root zone maintainer which is Verisign and they ultimately produce a root zone file which then the 13 different organizations pick up and distribute to all the thousand instances that I mentioned.

So, this is obviously a very simplified – very, very simplified – version of the actual process.

But let's actually go now finally to the resolution process and talk how it actually works. So, let's step through. Let's go back to my phone here in the lower left and my web browser and let's say that I type into my phone that I want to go to www.example.com. So, how exactly does my phone turn that name. www.example.com, into an IP address that it can then use to contact the web server for www.example.com?

Well, the first thing the web browser does is it calls the stub resolver and it passes through. It says, "Alright. I have this domain name. I need you to find me the IP address." So, the stub resolver – remember, very simple. One thing that it does know about is the recursive resolver that it should send queries to. And usually the stub resolver discovers this by the same way that the device gets its IP address.

So, a device comes up on a network for the first time and it uses the DHCP protocol. It says, "Help. I'm new on the network. I need an IP address." And the network gives it an IP address, and not only an IP address but it gives it other important configuration information so that the device can work on that network and one of those pieces of configuration information is the IP address of the recursive resolver the device should use.

So, in this case, this device is on a network that says, “Alright. You should use the recursive resolver at 4.2.2.2.” So, the stub resolver, it just got asked, “Well, what’s the address for www.example.com?” So, it constructs a DNS query. It constructs an actual packet and it sends it off to that recursive resolver.

Now, the recursive resolver is going to do all the work here, as we’ll see. To make this example more interesting, we’re going to assume that this recursive resolver has just been turned on, so it has nothing in its cache. The only thing that it knows is how to get to the root name servers because that’s what you have to know to do DNS resolution. You have to know how to get to the root servers. Everything else you can discover along the way.

So, this recursive resolver gets that query, and because it doesn’t have anything else in its cache, it says, “Well, I’ve got to ask a root server.” So, there are 13 root server identities it could pick from and it picks one. Actually, how it chooses depends on who wrote the recursive resolver. Let’s say in this case it chooses randomly and it chooses one called L.root-servers-net. So, this recursive resolver then asks that root server. It says, “Do you know the IP address of www.example.com? So, note that it asks exactly what it’s looking for. It doesn’t ask a more general question. It doesn’t say, “Hey, do you know about dot-com?” It literally says, “Hey, I’m looking for www.example.com.”

So, in this case, the root server, while it doesn’t know the address for www.example.com and it doesn’t know anything about example.com, but it does have the delegation information for .com, because in the root zone are the NS records for dot-com that tells everyone where the authoritative servers for dot-com are.

So, what the root server can say is here are the authoritative name servers for dot-com. You should go ask one of them. So, the recursive resolver puts that information in its cache to remember it for future reference and then it does what we call follows that referral. It picks one of the authoritative name servers for dot-com – and again, let’s say it’s choosing randomly and it picks one, it picks the c.gtld-servers.net, that’s the name of a dot-com server, and it asks it the same question. It says, “Do you know the IP address for www.example.com?”

Well, in this case, we wouldn’t expect a dot-com authoritative server to know that, but it does know the name servers for the example.com zone. So, it can send what’s called a referral. It can say, “Well, here are the authoritative severs for example.com. Why don’t you go ask one of them?” So now the recursive resolver caches a list of example.com servers and it follows that referral and it goes to an authoritative server for example.com and it asks the same question a third time and it says, "Do you know the IP address for www.example.com?" And in this case, the authoritative server says, "Yes, I do know that. And here's what it is." And now the recursive resolver caches that and it can finally return the answer to the stub resolver which can return it to the application that's been waiting all this time and now the application knows the IP address of www.example.com shown there and it can contact that web server and download the page and away we go.

So, that is, at a very high level, a simple example of how resolution works. Just like most things in life, it can be more complicated than this. This is the simple example. But it shows the important concept which is that you start at the root and you work your way down.

Now, as I said, there's caching that speeds everything up. So, after that query that we just did, that recursive resolver now knows the names and IP addresses of all the dot-com servers, the names and IP addresses of all the example.com servers and it knows the IP addresses for www.example.com.

So now let's imagine that there's another query that immediately follows that. Let's say that somebody looks up ftp.example.com. So, that web browser sends that request to the stub resolver. The stub resolver creates a DNS query packet and sends it off to the recursive resolver. But this time the recursive resolver can say, well, ftp.example.com, it doesn't have to start at the root. It doesn't have to go to a dot-com server because it says, well, I know the authoritative servers for example.com. I can go immediately to an authoritative server for example.com and ask it for the IP address of ftp.example.com, it can get the response and it can return it.

So, that's an example that shows how much caching helps. That recursive resolver didn't have to go back to a root server. It didn't have to go back to a dot-com server. It could immediately go to an example.com server. And then that address gets returned to the web browser.

So, that is a very, very quick tour. I threw a lot of material that's very complicated and I know I talked fast and I know it's first thing on a Saturday morning but the slides are available on the schedule. You can look at them. That's my e-mail address. I'd be happy to answer any questions via e-mail and we have a lot of time left in the session right now and we have microphones, so I would be happy to answer any questions if you have them now. So, please, just come on up to the mic.

In that case, I guess we'll close the mic line that doesn't exist. Thank you for coming. I'm going to stick around for a few minutes.

I'd be happy to answer any questions that you have if you want to come up and talk to me. Again, thanks for coming to this session and I hope everyone has a good week. Thank you.

CATHY PETERSEN: Thank you, everyone. That was our first How It Works tutorial for today. Our next one will be the How It Works on Understanding DNS Abuse and that will be here in—

[END OF TRANSCRIPTION]