SAN JUAN – How It Works: DNS Fundamentals
Monday, March 12, 2018 – 17:00 to 18:30 AST
ICANN61 | San Juan, Puerto Rico

CATHY PETERSEN: Good afternoon everyone, welcome to How It Works DNS Fundamentals. We have Matt Larson, our presenter and vice president of Research for the Office of the Chief Technology Officer at ICANN. As you can see, there's a few people in the ballroom here, if you don't mind moving upfront and we will start now. Matt?

MATT LARSON: Good afternoon, everyone. Welcome to How It Works DNS Fundamentals. We have few enough of us here that please, raise your hand if you have any questions. We have plenty of time for the material, so we can certainly stop and take questions.

So IP addresses are easy for machines but hard for people and that is really the reason that we're here talking about DNS in the first place. When we had only IPv4 addresses there was some hope that humans could remember an address or two, but that's an example IPv6 address and that's one with relatively few characters, there can be more depending how many zeros are in the address and it's basically impossible to remember v6 addresses. The point here is that people need to use names. Computers and routers use numbers, people need names.

In the early days of the internet, names were simple. We had what we call 'single label names', these were names that had no dots in them, they weren't domain names because domain names hadn't been invented yet. Every name on the early internet had to fit into a 24-character name space. Every name of every computer everywhere on the internet, and these we called 'host names'. Host is just a fancy word for computer.

Mapping these names to IP addresses so that humans can use the names but then the program or computer or router or whatever can use the number, that's called 'name resolution'. Name resolution on the early internet, in the days before DNS, used a host file and it was called 'host.text'. You don't have to know that, that's just for historical interest. That has the same function but a slightly different format as the modern Esty host file, if you're familiar with that on UNIX of Lynx. This is basically just a file, a text file with names and addresses, so host names and the corresponding IP address.

This was centrally maintained by an organization called The Network Information Center or The NIC, and they had the government contract from the US Government to handle certain network administration tasks on the early internet and one of those tasks was maintaining the host file.

This is when the internet was much, much smaller, it wasn't even called the internet it was called the ARPA net, it was an experiment by the United States Government's Department of Defense and we're talking there were tens of thousands of hosts on the internet at this point and so it was reasonable to maintain this file centrally and the update mechanism was very low tech, it was done by email.

When a network administrator added a computer to the network, took a computer off the network, changed its name or changed its IP address, it sent an email to The Nic and said, "Hey, here's what I've done, change of the IP address of this computer to this." The Nic then maintained the master copy of the host.text file and they released a new one once a week. Network administrators throughout the network, whenever they decided that their file was out of date or might be out of date, they would download a new. It didn't change that fast, you could maybe go a little while without updating. You downloaded it via FTP.

This was all extremely a low technology solution. There were obviously problems with this which you could predict if you thought about it for a while. One of these was naming contention. If you've only got 24 characters to name a computer in and the size of the network is growing, the more devices you have on the network, the more contention there is for names and the harder it began to be to avoid duplicates and to make matters worse, this file was maintained in a really simple way, they literally at The Nic they edited it in a text editor, there was no database behind the scenes it was just file and a text editor. There was no good method to prevent duplicates, every now and then duplicates crept in so that was problem.

Another problem, an obvious problem, was symbolization, nobody ever had the same version of the file, you were always behind. Then traffic and load was an issue. Toward the end of the time that host.text was used, the file began to be so big, that it took a significant amount of bandwidth on the internet just to download the file. This is a time when a 64 kilobit per second connection was a fast connection and I'm told, this is a little bit before my time but I'm told that near the end of host.text lifetime it took longer to download the file then the update period for the file.

In other words, you could never possibly get the latest file because by the time your file download had finished there was already a new file to download. Clearly maintaining this host file centrally, it did not scale, something had to be done.

Discussions started in the early 1980's on a replacement for host.text and there were two primary goals. One was to address the scaling issues that I've just described and the other was to simplify email routing and I'm going to talk a little bit about this. People when they think of DNS and the motivation for DNS, they tend to remember the first one, the scaling issues but they maybe forget or never were aware of the email routing issues but that was an issue as well. The result, not surprisingly, the reason we're here, is the domain name system.

This is my one slide summary of DNS. If you had one slide to summarize DNS, this would do it. Fundamentally DNS is a distributed database and in this database the data is maintained locally, so everybody has their own portion of the database and they get to maintain their data but this data is available globally because the database itself is distributed over the entire world, over the entire internet. You maintain your own data locally but you can look up and see anybody else's data.

DNS follows a client server model. Resolvers are the client side and if there's one thing to remember about resolvers, it's that they send queries. Name servers are the servers side and if there's one thing to remember about name servers, it's that they answer queries. Resolvers send the queries that the name servers answer.

There's some optimizations, DNS uses caching to improve performance and what I mean by that is, since we're talking about a database distributed over the entire world, the speed of light is only so fast and so when you do a look up on this database you might have to make several queries that cross the entire the world and come back again and that takes time. It's very helpful, in fact it's vital that you remember not only the final result of a look up that you do but all the intermediate results as well and remember them and you cache them, that speeds up the process the next time.

DNS also uses replication to provide redundancy and load distribution. What I mean by that is that I said everybody maintains their own local copy of their data but replication is involved, they don't just have one copy, they have multiple copies of their data and that provides redundancy. In other words, if you only had one copy of your data and something happen to it, then nobody could look up anything but if you have multiple copies then you have redundancy. It also spreads the load. If a lot of people are doing look ups and you have multiple copies, that query load can be distributed among the multiple copies.

Here are the various components that make up DNS, illustrated all in a picture here on one slide. We're going to talk about these in more detail as we go through the session but I think it helps to show them all at once to show you where we're headed. Let's start in the lower left and work our way across the slide. On the lower left we have a device connected to the internet, in this case a phone but really any device connected to the internet, that needs to turn names into addresses, in other words any device that uses DNS, it's going to have a simple DNS client in it called a 'stub resolver'.

The stub resolvers job is to be a bridge between an application, like in this case I have a web browser icon, the stub resolvers job is to be a bridge between an application, like a browser and the whole rest of DNS. The stub resolver takes an applications request, for example to turn a name into an address and then it turns that into a DNS query that it sends off and it send that to something called a 'recursive resolver'.

The stub resolver is very simple, all it knows how to do is accept a request from an application, turn it into a DNS query, send that query to the recursive resolver and then wait for a response. The recursive resolver on the other hand is considerably more complicated. It knows how to contact various what we call 'authoritative name server' that's where the data in DNS is store that gets looked up.

The recursive resolver might have to contact multiple authoritative servers to find its way to the answer, it might contact one and that authoritative server might say, "No, I don't have the final answer you're looking for but I can refer you to another name server." Then it would contact that other name and that would say, "I don't have the answer but I can refer you somewhere else even closer."

And so, the recursive resolver is smart enough to navigate these authoritative servers and find the answer. If we look inside the box of a recursive resolver, you'll see it's actually made up of a name server and a resolver. Remember, name servers answer queries, so the name server component of the recursive resolver answers queries from the stub resolver but then the resolver portion, remember resolvers send queries, the resolver portion is what sends queries to authoritative name servers.

Then I've also shown the cache there, everything that the resolver receives, every response it receives from authoritative servers it puts in its cache so it can use that to answer future queries. That is the DNS ecosystem if you will at high level.

Let me define some important DNS terms and concepts here. The first of these is what we call the 'name space.' I said DNS is distributed database and the structure of that database is what we call the name space. You may be familiar, when I say database you may be thinking of a relational database if you're familiar with those and the structure of a relational database is you have multiple tables and the tables have rows in them. Within each row you have columns and information. That's how a relational database is structured.

The DNS data, the name space, it's structured something completely differently, it's what we call an 'inverted tree.' I have an example of very small portion of the DNS name space on the slide. In an inverted tree the root is at the top and branches grow down.

This is a computer scientist tree, it should not surprise you that a computer scientist tress is upside down, root at the top, branches grown downward. Every node in this tree, so every box on the slide, every node, has a name, has a label. The root node is special. The root node at the very top of the tree, it actually doesn't have a label or rather its label is the 'null label', its label is nothing. Sometimes you see that represented like I have on the slide as a dot in quotation marks, just to show that there's nothing there. That's the name space.

We often refer to these nodes in the names space, their position relative to the root. The root is at the top and then immediately below the root, if you look on the left, we have what we call 'top level nodes', they're immediately below. A level below the top level you have second level nodes and so on, this goes down the name space as we move down the tree. Sometimes we refer to nodes using family terms like parent and child.

In this example here, the root is the parent of .com and .com is the parent of example but example is the child of .com, you talk about parent child relationships. Each of these labels has a limited set of characters that can be used, we call it 'LTH' for letters, digits and hyphen and those are the only characters that are legal in DNS label names and the maximum length of a label is 63 characters. Another important point to know is that labels names are not case sensitive, you can mix uppercase and lowercase and they're equivalent.

Every one of these nodes has a domain name and the purpose of a domain name is to tell you where the node is in the name space. The definition of a domain name is really straight forward, you just start at a node and you work your way up toward the root and you write down labels and you put a dot in between. You can see the note highlighted here at the bottom, we write www and put a dot, then we go up to its parent, that's example and we put example and put a dot and then com and a dot and we're at the root.

A special kind of domain name is what we call 'fully qualified domain name' and this one that's not relative to any other domain. A fully qualified domain name unambiguously tells you where that node is in the name space and FQDN, fully qualified domain name ends in a dot and the dot is actually the separator between the TLD, in this case dot come and the root. You get to the TLD and you put a dot and the you put the roots label but the root doesn't have label, the roots label is the null label, that means the domain name ends in a dot.

If all this looks a little familiar you may be familiar with the Unix file system or for that matter the Windows file system, that is an example of a data structure that can also be represented as a inverted tree. In a file system, the nodes represent files or directories and you have instead of a domain name you have a path name, a files path name tells you where that file or directory is in the file system, just like a domain name tells where a node is in the name space.

We have another important term here, 'domain.' The definition of a domain is really simple, it's simply a node in the name space and everything below it. For example, I've got highlighted the domain dot com, dot com would be the node dot come and everything below it. I'm showing three dot com names here, there are actually 131 million, obviously a few are missing from the slide. The dot com domain is huge, it's everything below dot com, any domain name that ends in dot com, is the in the dot com domain.

Let me contrast that with the term zone and this is a really important term because this is something that you hear thrown around and it's important to understand. Remember, the reason we go here, the reason we have DNS is that we needed distributed administration, centrally maintained information about host names and IP addresses didn't scale it didn't work, so the idea was we had to distributed, we had to have everybody maintain their own information about their own host names and addresses.

The name space therefore is divided up to allow distributed administration and these divisions, these administrative divisions are called 'zones.' Everyone get's their own zone, it's like a little sandbox that they can play in. They can make all the changes in their zone and not affect anyone else, they're in charge of their zone, they maintain it.

Zones are created by delegation. You delegate from higher in the name space to lower in the name space. The delegating zone we call a 'parent' and the zone that's created we call 'the child' and this process starts at the root and works its way down.

Let me give an example of this. Here's the name space again and if you just look at the name space like this, we don't have enough information to know where the zone boundaries are. The name space, just looking at it, we don't know how it is divided up for administrative purposes.

An example I give of this is imagine you're looking down from a satellite or the space station and see the content of North America. When you're just looking at North America you can't tell that in reality there are three countries, there are Canada, United States and Mexico, you don't know that.

You can look at North America all day and never know the administrative boundaries, the political boundaries because they just don't show up, you have to have that extra information. It's the same way with zones. You can look at the name space but unless you know where the delegation happens, you don't know where the zone boundaries are.

Let me draw some zone boundaries. I happen to know where the delegations are in the name space so that's why I can put where these zone boundaries are. At the very top of the name space we have the root zone and the root zone then delegates to zones at the top level and the way to think of this is that a zone has this delegation information that creates or that points to a zone that it delegates and this delegation process continues down the name space.

The root zone delegates to dot com in this example and then the dot com zone delegates to in this case example dot com and bar.com and foo.com. If we think about the relative size of these zones, let's start with the root zone. Right now, last time I looked there were 1543 top level domain zones and so the root zone is relatively small, it only has to have delegation information in it for 1543 zones below it.

Let's move down to dot com, dot com on the other hand, if you looked at the material that Verisign included in your bag that you got when you registered, their latest report said that there are about 131 million names in dot com. The com zone is huge, it's the biggest zone there is, it has to have information in the dot com zone, delegating to the 131 million dot com zones, second level zones below it. Delegation can continue below the second level, I don't have an example on this slide of that but it certainly can, delegation can happen arbitrarily deep into the name space.

You remember I said just a moment ago that DNS uses replication to achieve redundancy and improve performance, that replications what we're talking now. Remember, name servers answer queries and we say a name server is authoritative for a zone if it has complete knowledge of that zone.

The idea is an authoritative name server, it knows what's in the zone and it can answer definitively if someone asks it about it, it can say, "Oh yes, you asked about something in the zone, here's the information." Or it can say, "You asked about something and that name doesn't exist, it's not in the zone, so I can tell that it doesn't exist."

Zones should have multiple authoritative servers, that's the replication and as I said, it provides redundancy and it spreads the load. For every zone, it has to have at least one authoritative name server and in reality, it's going to have more than one, it's a best practice to have at least two authoritative servers because if you had only one authoritative server and something happen to it, nobody could query your zone.

If you're going to have multiple authoritative servers, you have to keep the information synchronized on them. The idea is that the information about a zone should be the same on all the authoritative servers. How do you do that? The good news is that the DNS protocol has a way to do that built in.

Synchronization of zone data across authoritative servers is built in and there's a process call a 'zone transfer' that lets you move a zones data around. You designate one authoritative server as what we call a ‘primary’ and that’s where you make changes to the zone and then you have other authoritative servers that are called ‘secondaries or slaves’ same term and those authoritative servers load the zone data from the primary, they contact the primary and they do something called a zone transfer and they copy the zone from the primary to the secondary.

It’s important to point out that all servers for a zone, all authoritative servers, they’re equal, they’ve all got the same data. The only difference between primary and secondary is where do they get the data? The primary loads the data for a zone off its disk and the secondary loads the data for a zone from the primary but the data on the primary and the secondary is all the same.

Of course, they can be out of sync briefly because you’ll make a change on the primary and then it will have more up to date information but then that propagates to the secondaries and then they get synchronized. This is nice that this is built into the DNS protocol, that you don’t have to worry about keeping name servers in sync on your own, it just happens.

Now what I want to do is move down a level. We’ve talked about zones, now let’s look inside a zone and let’s talk about the data in a zone. Remember, every node in the name space, every box on the diagrams that I showed, every one of those has a domain name that corresponds to it and the way to think of this is, is that a given domain name can have different types of information, different types of data associated with it.

The most common kind is an IP address. We might associate an IP address with a domain name. This kind of information that goes with a domain, we call those ‘resource records’. Resource records are the data in DNS and there are different types of resource records to store different types of data. The most common types of resource records are the ones that store IP addresses. IP version and IP version 6, they’re different types of records for those but they’re resource records to store other types of data.

A zone simply consists of a bunch of resource records and all the resource records for a zone are put in a file and we call that ‘zone file.’ There’s a zone file for every zone and you never mix records from multiple zones in one file. A zone is nothing more than the collection of its resource records.

Let me show you what these resource records look like. There’s actually a way to write them down, a standard way to write them down in text. Resources records have five fields, we don’t need to go over all of this. The important thing to take away here is that they are resource records of different types and then they carry data of that type. These are some of the most common types of resource records.

I already mentioned that we have a type of record for IPv4 addresses that's called an 'A record or an address record' and then we have a type of records that stored IPv6 addresses and we call that a 'quad A record' for the 4 A's. Then we have a few other types here that I'm going to briefly talk about. This list is the most common type but there are actually many other types of resource records.

The last time I looked there were 84 different types and there is an IANA registry that called, well I have it on the slide I won't read it to you and the URL for it, if you go to that webpage this is what that looks like. Based on the size of the resource record field, we can have up to 65,000 of them and we're nowhere near that, we have 84.

If you think of a new thing that you want to store in DNS, you can go to the IFT and you can write an internet draft and convince people that your idea should become a new DNS type and you can get created and get a type allocated in this registry and then you'll have new things you can put into DNS and people are doing that all the time. Not that frequently, we've only 84 of them but people do think of new things that they want to put in

DNS and they create a new type to store new types of data. But by far the most common type of data in DNS is addresses.

The most common use of DNS is to map domain names to addresses and here again are these two address type records that I showed. Here's an example of the actual text representation of what an address record and a quad A record looks like. We have a domain name on the left, example dot com and then we have the type, which in the first one is A for address and then we have the actual address.

This is an example of a resource record that would be in a zone file for the example dot com zone and this resource record simple says, example dot com has this IP address. Below that is a quad A record and that's who you would say, example dot com has this IPv6 address. Most of DNS consists of A and quad A records because as I keep saying, that's the main purpose of DNS, to map domain names to IP addresses.

There are other types and what I think is kind of interesting is that most of these types are used by people who consume DNS information, who are looking information up in DNS because they need to do something like connect to a webserver, so their web browser needs to look up a name to address mapping.

But, some types are used, mostly by DNS itself and the primary examples of those are the NS record and the SOA record which we're going to talk about very briefly and those are types that nothing cares about accept DNS itself. What's kind of interesting about this is that those are types just like other types and I like to think of this like a warehouse, if you can compare it to a warehouse.

Let's say you rent a warehouse and you have a bunch of stuff that you want to put in the warehouse. You don't just back your truck up and start throwing boxes into the warehouse, in order to use the warehouse, you need to build some shelves and once you have the shelves up then you can take the boxes, the stuff you really care about, the boxes, the goods in the boxes and you put those on the shelves.

The warehouse isn't any good without the shelves and DNS is similar in that you can think of these NS records and SOA records are the shelves, they have to exist in order for DNS to work but nobody outside DNS actually cares about these, they care about all the other types like A and quad A.

Let me talk about these NA records. These are how you say what the authoritative name servers are for a zone. This example here shows two NS records and what these NS records say is that example.com, that zone has two authoritative name servers, one is called NS1.EXAMPLE.COM and the others called NS2.EXAMPLE.COM. The left side is the name of the zone and the right side is the name of a name server.

Now, NS records get a little bit complicated because they actually appear in two places. They appear in the parent zone and they appear in the child zone. In the box here, I have the actual list of NS records for the dot com zone, there are actually 13 authoritative name servers for the dot com zone and they are named, as you can see, look on the right, A.gTLD-SERVERS.NET through M.gTLD-SERVERS.NET, that list of NS records, those 13 records, they appear in two places.

Let me go here to zoom a little bit. They appear in the root zone, in this case the root zone is the parent zone and these records, these NS records in the parent zone are actually what does the delegation, that's what tells the rest of DNS that below the root zone is the dot com zone and then list of NS records they also appear again in the zone that's named, in this case dot com. The dot com NS records appear in root, in the parent and then they appear in dot com itself.

When we talk about how you look data up in DNS, you'll see how important it is that these NS records appear in the parent because I will go ahead mention it in advance that the way resolution works in DNS, the way you look up things in DNS is that you start in the root and you follow these delegation pointers, you follow the NS records. If you were looking up something below dot com you'd start at the root and in the root zone you'd see the delegation to dot com and then you could go to dot com name server and you could find a delegation below that and so on but more on that in just a moment.

The NS records include just names, if you look at the example that I gave, we say that example dot com, one of the name servers is NS1.EXAMPLE.COM but that alone is not enough because you need then the IP address of NS1.EXAMPLE.COM if you're actually going to contact it. The delegation information also needs to include address records in some cases, we call those 'glue records'. If you ever hear people talk about glue records, it's the address record for a name server.

There's another record that is in every zone called an 'SOA record' and I don't want to talk a whole lot about the details of this but I just want to point out that it exists. There's one of these SOA records for every zone and it's at the top or what we call the 'apex' of the zone. Most of the values here relate to that zone transfer that I mentioned earlier. They tell authoritative servers how to synchronize, how often to synchronize the zone.

Let's go back to the second goal for DNS, the second problem that DNS was mean to solve and that was helping with email routing. The issue that DNS had to solve was how do you deliver mail based on an email address? In early days, before DNS, you'd have email addresses that would be some user @ and the host name and it would be one of those maximum 24-character names.

That was your email address and what that meant, before DNS, was that your email was going to go to host named on the right side of the email address. There was no way to say, my email address is MATT@FOO but you should really send that message to a machine called Bar, somewhere else. No, if you're email address was MATT@FOO, you're email went to the machine called FOO and that's where you had to read it.

One of the goals of DNS was to decuple that, to be able to say, "This is my email address and here is where that mail should go, you should send it somewhere else." DNS offers the flexibility of that. We have a record called the mail exchange record that tells you where email for a domain should go and here's an example. These MX records they're called for the example dot com domain name, they say where mail should go.

For any user name @ example dot com, these MX records say you should send it to a machine named MAIL.EXAMPLE.COM. There's this preference value, it's called the number 10 and the number 20 and that's a little counter intuitive because the lower the preference the more desirable the mail server. These two MX records, what they say is, for any email addressed to any one at example dot com, you should send that to MAIL.EXAMPLE.COM but if you can't for some reason, then you should try MAIL-BACKUP.EXAMPLE.COM.

Any mail server that's going to deliver mail has to be able to look up MX records for domain names. There's a very, very tight coupling between DNS and SMTP based email. A mail server when it's get a message to deliver, it looks up the MX records for the email address and that's how it knows where to send the message.

So far, what we've been talking about is mapping names to IP addresses, which is an important task. Sometimes you want to the opposite. Name to IP we call 'forward mapping' but what if you want to do from an IP and know what its name is? There are some times when you want to do this.

Imagine for example there's a network troubleshooting tool called Trace Route, that lets you show every router between you and IP address you want to get to. When you show the IP address of every router, you may be interested in the IP address but you might be just as interested to know where is that IP address? Who operates it? What's its name? That's an example of reverse mapping, which is taking an IP address and finding the corresponding name.

Think back to having just a host table. If you have just a file that has a list of names and IP addresses and you want to do forward mapping, it's really easy, you have a name and you want to map it to an IP you look through that file until you find the name and there's the IP and you're done. What if you want to do reverse mapping? That's just as easy. You look through the file until you find the IP and then there's the name. That's works great with a host table. What do you do with DNS?

Let me go back to an example of the name space here. The way the name space is structured, it makes it very easy to look up domain names but in this case it's impossible to look up IP addresses. If I want to look up a domain name, like say WWW.EXAMPLE.COM you can see how I start at the root and work my way down until I get to WWW. But what if I start with an IP address, what do I do? The answer is you can't do that in DNS as far as I've shown you so far because you can't look up IP address.

What had to happen is, we had to have a way to turn IP addresses into domain names, so that you can then look them up as domain names, so that's in fact exactly what we have. There's another record type called 'PTR' for pointer and IP addresses go into special domain names where you PTR records in and this lets you look up IP address as domain names and then find out the name that they correspond to. IPv4 addresses go under a domain name called 'IN-ADDER' for internet address dot arpa and then IPv6 addresses go under a domain called IP6.ARPA.

Let me show you an example of this. Here is the name space tree. I'm just showing you a new portion of it, that you may not have been aware existed. Over on the right we have EXAMPLE.COM, which we know about it but then here we have the INADDER.ARPA domain and in this case if you look way at the bottom, that is where a PTR record for EXAMPLE.COM zone goes. You can see on the right we have an address record that says, the IP address for EXAMPLE.COM is 192.0.2.7, if you want to look up EXAMPLE.COM you see the address record at EXAMPLE.COM and you know the address.

What if I know what is the name if the domain name that corresponds to the IP address 192.0.2.7? What you have to do is you have to turn that IP address into a domain name and what you do is you take the IP address and you flip it around and you add IN-ADDER.ARPA to it and then you look up the PTR record. The only reason this works is that everybody understands the rules. Everybody understands what I'm describing to you. If you have IP address space assigned to you, the regional internet registries, the RIR's they cooperate to manage the IN-ADDER.ARPA domain and you can get the zone that corresponds to your IP addresses delegated to you.

Let's for example that you had assigned to you 192.0.2/24 everything that begins with 192.0.2 you would have the 2.0.1. 192.INADDER.ARPA domain assigned to you and you would have to put PTR records in there if you wanted people to be able to do the reverse mapping of your IP addresses. It works, it's a little awkward but it works.

I think most people would agree the reverse mapping is frankly not as important as forward mapping. Forward mapping is what let's you type in a domain name in browser and get to the website. Reverse mapping tends to be used more for diagnostic and troubleshooting uses. No normal people other than network engineers or system administrators probably care much about reverse mapping.

As I said, there are many more types of resource records, here's an example of still more just to give you a sense for the other kinds of data that people have thought of to put into DNS. Here then is an example of what zone file, for a very small zone might look like.

Here's a zone file for our hypothetical EXAMPLE.COM zone and I know I haven't talked about everyone of these records in detail but this is a small zone that is similar to what most every zone on the internet is like because if you think about it, most domain names on the internet, you probably want to do two things. You want to have a web server, you want to have a website and you want to accept email.

Now, there are obviously domains that have many, many more things in them than that, all kinds of names but a lot of domain names, like my personal domain name for example, that I use for email, I care about email and I have a small website. My personal zone for my domain name looks something like this, this is all you need to support those applications.

We have an IP address for EXAMPLE.COM, which is where our web server would be. You could look at this zone file and you could make a guess that 192.0.2.7 is the IP address of the website for EXAMPLE.COM because we have a record that maps EXAMPLE.COM to the IP and then you see some MS records there that say where to send mail for EXAMPLE.COM.

Now I want to talk about the resolution process. This is how you look up things in DNS. The DNS components that I showed at the beginning of this session on that picture, the stub resolver, recursive resolvers and authoritative name servers, they all cooperate to look up data in the name space.

An important thing to know is that a DNS query always has three parameters, it has a domain name, like WWW.EXAMPLE.COM, it has the type of data you're looking for, in this case A for address, there's also this value called the 'class' that I skipped over and didn't talk about. Class was a way that people thought early on they might be able to use to extend DNS to other types of network, it really hasn't been used but it's baked into DNS, we're stuck with it.

You can in this case, class is always going to be something called 'internet class' and you don't ever have to worry about it. In this case it's really just the domain name and the type that are important. If you're going to ask a DNS query, if you're going to ask a name server a question, you always have to specify the domain name and the type.

There are two kinds of these queries, stub resolvers and remember stub resolver is in things like your phone, your toaster, your refrigerator, your laptop anything that connects to the internet and needs to turn names into addresses or other pieces of information, all of those devices have a stub resolver. Stub resolves send what are called recursive queries and a recursive query is the signal to the recursive resolver that says, "Hey, I'm a stub resolver and I just need you to give me the answer or an error. I can't take anything else. I can't take a partial answer, I need the full answer to what I asked."

Recursive resolvers on the other hand, they're smarter and they can accept these partial answers that are referrals. They send a type of query that indicates that, that's the case, that they can take a referral in response. As I already said, if you're going to look something up in DNS, you start at the root zone and you work your way down, you follow delegation pointers.

There are authoritative servers that are authoritative for the root zone, they have all the information in the root zone, that's what authoritative means and we call these 'root name servers.' If you're going to start resolution at the root zone you need to be able to contact a root name server, so how do you find who the root name servers are? The answer is, they have to be configured, there's not a way to discover them, they have to be configured on every recursive name server. This is different than say other network parameters.

When my phone came up on the WIFI network, on the ICANN WIFI network in the conference center, it used a protocol called the 'dynamic host configuration protocol.' DHCP and my phone asked the network, it said, "Hey, I'm a new device on the network, I need an IP address." And the network said -- my phone knew nothing about this network and the network said, "Okay, here's your IP address and here are some other configuration parameters that you need to know about, including actually the IP address of a recursive name server to use."

That's an example where a device can know nothing and the network tells it everything it needs. That's not the case with a recursive name server, you can't just turn a recursive name server on with no configuration, it has to know who the root name servers are and what their IP addresses are. There is a special file that every recursive name server needs called the 'root hints file' and that has the name and IP addresses of the root name servers.

The good news is if you're installing a recursive name server let's say on a Lenox machine, somebody has already done the work to package, when they packaged up the recursive name server software, they included the root hints file and some recursive resolvers even have the names and IP addresses of the root servers as part of the software code itself. But that is the URL where you can get the root hints file and this is what it looks like.

There are 13 root name servers. There are 13 servers, authoritative for the root zone. The dot on the left there means the root zone and then we have 13 NS records that say these are the names of the name servers for the root zone and you can see there are called A.ROOT-SERVERS.NET through M.ROOT.SERVERS.NET, they have those names and then you can see below the IPv4 addresses for them and the IPv6 addresses. Every root name server has an IPv4 address and IPv6 address. A recursive name server, in order to do any resolution, it has to have this information, it has to know the names and IP addresses of the root name servers.

Let me take a brief detour and talk about the root zone and how information gets into the root zone. Remember, what do we have in the root zone? We have information about the top-level domain zones. We have NS records for the TLDs and administering the root zone is kind of complicated. There are two organizations that work together, ICANN has a role called the "IANA Functions Operator' and ICANN subsidiary PTI handles that role and then Verisign is the other organization and they have a role called the 'root zone maintainer'.

This arrangement is actually quite old, it dates to the early 1990's back when Verisign -- it was actually called Network Solutions that Verisign bought in the year 2000. The IANA Functions Operator was a role that was performed by the University of Southern California before ICANN was created.

This is a very old historical arrangement and it's somewhat complicated but that's the way the root zone is.

Those two organizations, ICANN and Verisign, they cooperated to put together the data in the root zone, to make the root zone file, then we need authoritative servers for the root zone, the root name servers. There are 12 different organizations that operate the authoritative name servers for the roots zone. This is kind of unusual, for most zones you have one organization that operates all the authoritative servers.

Let's take dot com as an example. I just to work at Verisign so I know a little bit about how that works, Verisign operates all the authoritative servers for dot com. You take another zone, what a lot of companies do is they'll maybe operate some of the authoritative servers themselves or they'll outsource them to a third-party provider or maybe multiple third party providers for redundancy but not 12 different organizations, that's unusual.

Here are the 13 root name server letters. Even though it's called A.ROOT-SERVERS.NET for short hand people just call it AROOT through MROOT and these are the organizations that operate A through M. It's an interesting group of organizations and they really have nothing in common other than that they operate a root name server. If you look at the list you can see commercial organizations, educational institutions, nonprofits, ISP's, departments of the US Government, you have a little bit of everything here and this again, goes back a long time, we're talking 20 years ago.

This list of operators has been static, has been the same for 20 years and there's a whole bunch of complicated stories around that as well that we don't have time to go into. We do have 13 root servers and 12 organizations. It's 12 because Verisign operates two, they operate AROOT and JROOT. That's the root zone, the root name servers, the root operators.

If you want to know more about the root operators and the root servers you can to the ROOT-SERVERS.ORG website and it tells you about where all the root servers are located and a little bit of information about them.

At a real high level let me give you a sense of how the change process for the root zone works. All the information in the root zone relates to TLDs. If a TLD manager wants to make a change, so they would want to add an authoritative server for their TLD, remove an authoritative servers for their TLD or change the name or IP address of one of their servers, they would submit a change to the IANA Functions Operator, which is PTI owned by ICANN and PTI does several checks and they update the root zone database that they have and then they send that request to the root zone maintainer, which is Verisign.

Verisign does more checks, Verisign updates its root zone database, creates a root zone file and then makes it available and then the 13 root servers download that file and they make it available. This is a very, very high-level version of the process. I'm just illustrating it to show how the different organizations cooperate to make this work but there's a lot more to the process then I'm showing here. That's a short detour into the root zone.

Let's actually talk about how resolution works. Let's say that we have a phone in the lower left and someone has opened the web browser on their phone and they type in WWW.EXAMPLE.COM the web browser call the stub resolver and the stub resolver is just a very simple piece of code, so the orange here represents that it's actually an API call, it's one portion of the web browser program calling the stub resolver, another program or portion of the program, sometimes it's just a function call and it says, "I need the address for WWW.EXAMPLE.COM."

Remember, the stub resolver is very simple, all the stub resolver knows is the IP address of a recursive server or maybe several recursive servers, that is should send its query to. The stub resolver turns that into a DNS query, that request from the web browser and then it sends that DNS query to the configured recursive name server, which in this case is 4.2.2.2, that's actually the real IP address of a recursive server name server operated by a ISP called level 3 and this recursive servers is what they call open, so anybody can send a query to this recursive server if we want, it's just a good IP address to use as an example.

The stub resolver asks the recursive resolver to resolve, ask it for the address of WWW.EXAMPLE.COM now to make this example more interesting, I've said remember that recursive resolvers have a cache but to make this interesting we're going to assume this recursive resolver has just been turned on, so it has nothing in its cache, all it knows are the names and IP addresses of the root servers. The recursive resolver picks one of the root servers, let's say that it picks LROOT and it asks it the same question that it just received from the stub resolver which is, what's the IP address of WWW.EXAMPLE.COM?

Now the root server, it can't answer the query directly, the root server does not know the IP address of WWW. EXAMPLE.COM, it doesn't know anything about EXAMPLE.COM but it does know something about dot com because the root zone contains the delegation to dot com. The root server can return what we call a referral to dot com, it can say, "Here are the name servers and their IP address for dot com."

Now the recursive resolver it caches that respond so it can use that information in the future and then it does what we call 'following the referral' it picks one of the dot come servers and it sends it the same query. An actual dot com server is called C.gTLD-SERVERS.NET, Verisign chose to name the dot com servers similar to how the root servers are name.

Just as there are A through M ROOT-SERVERS.NET there are A through M gTLD-SERVERS.NET but those are the names of the dot com severs. Our recursive resolver has picked one of the dot com severs, C.gTLD-SERVERS.NET and it asks it the same question that it received from the stub resolver and that it asked the root server, it says, "What's the IP address of WWW.EXAMPLE.COM?"

Now, the dot com server, it does not know the IP address of WWW.EXAMPLE.COM either but it does know who the authoritative name servers for example dot com are so it can send back that list, it sends back a referral to example dot com.

Now our recursive resolver caches that and it follows that referral and it sends the same query for a third time to one of the authoritative servers for example dot com which is NS1.EXAMPLE.COM.

Now, NS1.EXAMPLE.COM is authoritative for the example dot com so it does know the IP address of WWW.EXAMPLE.COM and so it can return it to the recursive resolver which caches it, sends that back to the stub resolver, which passes back to the application and now that application knows the IP address of the web server and it can contact it, download a webpage and keep going. That's the resolution process, that's the most simple version. Like most things with DNS, it can get more complicated than that but that is the simple version. The important point here is that you start from the root and work your way down.

You don't always have to start from the root because of caching. I highlighted all the caching that happened as I described that. What happens if somebody comes along and immediately thereafter wants to go to FTP.EXAMPLE.COM?

The stub resolver assembles a DNS query and sends it to the recursive resolver as before but now recall everything that the recursive resolver has cached, it knows about the dot com severs, it knows about the example dot com servers, so it doesn't have to start at the root, it doesn't have to go to dot com, it can go straight to an example dot com authoritative server and ask it's question, get the response, now cache this as well and return that to the stub resolver, which returns it to the application. You can see how caching speeds things up greatly. If it weren't for caching everything would be a lot slower on the internet.

The last slide I have then is to give you a high-level picture to connect, we've been talking about just DNS but when we think about domain names in a ICANN context, we think of more than just the authoritative DNS side, we think of the bigger picture that includes registrants, registrars and registries.

I'm just showing this bigger picture to show where the other actors in the naming world go. We have registrants who communicate with registrars, usually via a website to buy domain names or make changes. The registrar then communicates with registries. The main part of a registry is the database, which domain names are registered and information about them. The registry has to make the information in the database that's related to DNS, it has to make that available on authoritative name servers and then those are what recursive resolvers query. Hopefully this is a little helpful to show how this fits in to the larger picture.

That is all the slides that I have. We have some time left. I would be happy to answer any questions if you have any?

CATHY PETERSEN: As a reminder, please give your name and your affiliation if you have any, thank you.

MATT LARSON: Anything in the Adobe room? Okay. Alright, everyone's hungry. I'm kind of hungry actually.

MALISA RICHARDS: Actually I do have a question. Malisa Richards, fellow from [inaudible]. What's the criteria you use when installing a root zone server?

MATT LARSON: It's different for different operators, each operator has their own policies. I didn't talk about this but all of the root servers at this point are any cast, which means that there's not just a single server for a given root server IP address but there are multiple servers and any cast technique uses the underlying routing system of the internet, that allows you to have a server with the same IP answering in multiple locations across the network.

All of the operators have used the any case technique to have multiple instances we call them, of their particular root server letter and the different operators have different policies. I know for ICANN, for LROOT, just about anyone we're happy to have operate an LROOT instance, it's literally a single server and what we ask is that you buy the server and then we -- you buy it, put it in your network and supply rack space and power and connectivity, bandwidth and then we operate it and operates as the rest of LROOT. Different root operators have different polices.

MALISA RICHARDS: One more question. I'm looking at the website presently that you recommended and I'm noticing that in the Caribbean particularly there aren't much root servers compared to South America, North America. Can you probably give a reason why that would be?

MARR LARSON: I don't know for the other operators, it just could be that the operators of those various networks have not approached root operators to try to get root servers installed there. At least for ICANN the bar is very low, if you're willing to buy a server, we're happy to put one in your network. Thank you, everyone. Oh, yup, go ahead.

NORMAN WARPUT: I'm Norman from Vanuatu, Icann fellow. Thank you for the presentation. I have one question regarding the root zone change process. Does that apply to the ccTLD top level domains?

MATT LARSON: Maybe I didn't understand your question.

NORMAN WARPUT: The process to change the root zone, does that apply to the county code top level domains?

MATT LARSON: Oh yes, absolutely. From DNS perspective a TLD is a TLD. We've made categories of TLDs like ccTLDs and gTLDs and then within gTLDs sponsored and unsponsored but that's all additional categories that we've put on things. From DNS perspective, a TLD is a TLD regardless of what we call it, whether it's a ccTLD or gTLD or whatever. Any other questions?

ANDREW FRASER: Where is it outlined where you would go or how you would point to a recursive server? At what level or is that through your ISP or? Cause there's a number of choices for recursive servers.

MATT LARSON: Right, so ultimately, it's configured on every device and when you're device connects to the network, the network has to tell you. Every network is going to tell in addition to when you get your IP address it's going to give you the IP address of a recursive server. Anybody operating a network then is going to have to have a recursive server.

ANDREW FRASER: So, my phone would be preconfigured based on my provider?

MATT LARSON: Well, I wouldn't say preconfigured I would say that when it connects to the network of your provider, your provider would tell it, "Here's your IP address, here's some other configuration including a recursive server to use." That they operate. However, you could if you wanted, you could override that and chose another. There are what we call third party recursive DNS providers or sometimes public DNS providers or open resolvers, there's multiple names for them and these are companies that operate recursive servers that anybody can use.

The first company that did this that I'm aware of on any scale is Open DNS and I remember when they did that, my reaction as well as a lot of peoples was, “Why would you possibly want that?” Recursive service feels like something like a core network service, why would I take something so important that I’m dependent on to do anything on the network and why would I rely on a name server outside my network and Open DNS value proposition was, well we can do extra services, we can to do content filtering based on names.

You could tell us, I don’t want to resolve names that correspond to say gambling or adult content or things like that and they also could do things like, we won’t resolve domain names that we know are sites that host malware or are otherwise bad from a security perspective. I remember thinking, Open DNS that that was a ridiculous idea but then Open DNS got by Cisco for over 600 million dollars, we know who -- I guess they get the last laugh. But after Open DNS, there’s Google Public DNS for example, Google’s stated aim for that was to have a very reliable recursive service that anyone could use that did DNS Sec validation.

I didn’t at all about DNS Sec but that’s adding cryptographic authentication to DNS. Google felt strongly, feels strongly about DNS Sec and so they’ve said, “Here’s a service that we’re offering that is free to use and you know that you’ll get the extra DNS Sec protection.” And there are other public DNS services, Verisign has one, there's Quad 9. If you want, you can override your devices configuration and use those.

Any other questions? Alright, thank you everyone.

CATHY PETERSEN: Thank you, everyone. Thank you, Matt. Thank you to our scribes and interpreters and to our tech team, excellent work, thank you again.

[END OF TRANSCRIPTION]