COPENHAGEN – How It Works: DNS Fundamentals
Sunday, March 12, 2017 – 11:00 to 12:30 CET
ICANN58 | Copenhagen, Denmark

STEVE CONTE: Alright, we're going to go ahead and get started. Thank you all for coming today. I'm Steve Conte, I'm the Programs Director for the office of the CTO for ICANN. Again, this is a DNS fundamentals class, so if you run a DNS server you're in the wrong room. If you're here and you want to catch up and get a refresher on DNS? You're in the right room. If you never touched DNS before, you're in the right room.

We try to do this – we call it the How It Works sessions. Throughout today and tomorrow, we'll be doing primer sessions on various pieces of technology that comes across to the ICANN sphere of the Internet system. This week, we're doing one on DNS fundamentals, which you guys are about to join.

We also have one on Internet networking; we have Alain Durand from the office of the CTO team as well as Jeff Houston from APNIC presenting that.

We have Understanding DNS Abuse, which is our Security Officer John Crain from ICANN talking about how DNS is used in abuse and how law enforcement interacts with us to understand better on how to fight against that abuse.

And then finally, we have the Root Server System Advisory Committee, the RSSAC speaking this afternoon about being a root server operator and what the root server system is, and what the role of the root server operators are as well.

Always an interesting session. I invite you all to do it. If you can't make any of the sessions today and yet they still have interest for you, we do a complete mirror of these sessions tomorrow at 2:00. So if you're looking at tomorrow's schedule and you see these, they're the same sessions. Anything said on How It Works today, it's the same session tomorrow. But if you can't make it today, look at the schedule and see if you can make our sessions for tomorrow. So with that, we're going to jump into the DNS fundamentals.

The core concept of DNS is that numbering is hard to remember. It's fairly easy to remember a phone number or maybe three phone numbers, but when you start looking at your entire phonebook and trying to remember all those phone numbers, it becomes increasingly difficult to remember them. The same with IP addressing too. There are 3.4 billion IPv4 addresses out there on the v4 space.

There are hundreds of duodecillion addresses in v6 space. It becomes increasingly difficult to remember the IP address of a specific server. So the DNS, the Domain Name System was

developed in order to attach human words, to associate them to IP addresses on the Internet. We use names, we remember names much easier than we do numbers.

So back in the early days, names were simple. There were only a handful of servers on the Internet at that point so it was quite easy to remember those, and they were single label. There wasn't a dot at the end, it was just a name, and that was it. They're referred to as host names. The actual concept of mapping the name to an IP address is called name resolution. Name resolution in the early days came down to a file on the computer called a host file, and every computer connected to the Internet had a host file.

That host file had basically a directory of names to IP addresses of the other servers on the Internet, so it was easy to get to, easy to look at. Again, it was just a handful of servers so it was easy to manage. There's a host file in this day and age too. It still lives; it's still the core of it. The host file system has slightly changed because now it's developed more as a pointer rather than a directory.

It was centrally maintained by the NIT, Network Information Center at Stanford. It was basically a place where if you had a machine, you put it onto the Internet, you would submit that information into SRI, they would put it into the host file, then

you would download it every once in a while to get it. They sent their updates via e-mail, and a new host file was released roughly once a week and you downloaded it through file transfer protocol, FTP.

So that was cool. It worked great for a while, and then the Internet started to get bigger and bigger and we started getting a couple of issues with that. First of all, we had naming contention. Back then, if you wanted a name, pretty much if it wasn't there you just said, "Okay, I'm going to call the server Steve and we'll move on and be good." But the more and more servers that came, there was the concept of people wanting the same server name. So there was a contention on that that had to be resolved. There was no good method to prevent duplicates, so we started looking at the concept of DNS and the concept of the Internet is that it's a unique system of identifiers. This broke that model almost immediately.

We also had synchronization issues. Because it was an opt in to download the host file, the network manager from whatever organization had to go to SRI and decided they were going to download the latest, greatest file. So there was no synchronization on that file. There was a good chance that every file was different on the various client machines, so there was no synchronization of the data itself, which means that some people who might not have downloaded their file, their update

in a while, they might have very old information and maybe those servers have changed or the new servers might not have been put onto the file there, so they didn't have the complete data.

It's kind of funny, just the traffic and load. We're in an age now where we have fiber optics to the house and huge amounts of bandwidth, but back then we were talking about dialup model and BOD rates of 200 to 1200 BPS, which was wonderful because we had this Internet, but it was excruciatingly slow. So downloading this file had a very high overhead, not just at the user, but also at SRI because they had to facilitate a lot of different downloads and stuff like that.

Essentially, maintain host file just didn't scale. Having a single file as more and more hosts joined the Internet, it became more and more difficult to have a single place for all this data to live. So they started discussion in the early '80s on a replacement. The goals on this were to address the scaling issue of the host.txt file – I'll stand back a little bit so I don't have to look backwards – and to simplify some of the e-mail routings.

E-mail was becoming more and more prevalent in the Internet in the early days as well, so how to handle different mechanisms to route that e-mail was part of the discussion on how to change the methodology in which hosts are resolved. The result of this

through discussions and engineering was the Domain Name System. It required a whole lot of documents. These were done in the IETF, the Internet Engineering Task Force. The IETF produces open standards and they do that through methodology called RFCs. They're standards documentation that they release. So here we have a couple, 799 and 819 were some of the early RFCs, early standards that described what the Domain Name System was.

Instead of having a centrally located data source in the old model with SRI, it was discussed that as distributed database was a better model. It was more adaptable and scalable and more manageable too, because it put the power of management of the various zones – and we'll go in through resolution in a little bit. It put the power of management with the zone operator or the zone owner.

So, it was easier to maintain that largely. It required less of a single point of failure because there were more people who were able to modify their own zones without having to rely on a single entity to make those changes.

They introduced caching which improves performance. I have a resolution diagram and we'll talk about caching in a little bit. And it provided replication which helped facilitate the redundancy and load distribution on the service as well.

So DNS at a glance, we're talking about a couple of different things. We're going to start right and move left. We have our authoritative nameservers. If we look at it from a TLD perspective, these are your .com, your .net, your .uk, .dk, any of that. And if you're a domain holder – so for instance, I have conte.net, I have certain servers on the Internet that are my authoritative nameserver. They could also be authoritative for conte.net as well.

We have recursive nameservers. Those usually live within ISPs, of if you have a business network, you might have a recursive server there. Those are your proxies for the most part. Those are the servers that will act on the individual's behalf when they go looking for DNS information.

We have your stub resolvers. We show a telephone here. It could be anything. It could be your laptop, it could be your telephone, it could be whatever device that's doing a DNS call or query will also be a stub resolver because it has to understand how to make the query but also how to parse the answer as well. And then it also has a level of caching.

All of these, the recursive and the stub have a certain level of cache. Most of the cache happens at the recursive level, and we'll talk about why. But these are the types of servers that we'll be talking about within the DNS structure.

The DNS database is an inverted tree. We call it namespace, and what we look at is if we turn it over, the new see the root is the branch or the main part, and then it starts branching out from there. This is done purposely, and each section on these is called a node. Each node is specific to this, and we do it this hierarchical way because it adds and takes away from that single distribution point.

I think the next slide – okay. So within these nodes, we're restricted to LDH (letters, digits and hyphen). Those are the only legal characters, ASCII letters, digits and hyphen. Those are the only legal characters that can reside within the DNS space. So even though we have IDN these days, Internationalized Domain Names, when it gets to the DNS protocol itself, it translates out to LDH format.

Actually, we have an IDN example here on the top left. You'll see the xn-j6w193g. That is a translation mechanism that IDN and Punycode take into an account. So that is some language other than ASCII that's ascribing a domain name. I honestly don't know which one that is off hand, but that's what the DNS sees. So if you're doing IDN in Arabic or in any other non-ASCII script, the DNS sees it as an xn- string value.

Labels are a maximum of 63 characters. Can't have more than that. From a human's perspective, you really don't want more

than that because the purpose of the DNS is to be something that's memorable. And if you try to remember a domain that's 63 characters long, that's going to be more difficult anyway. And they're not case sensitive either.

So within every node, we have a domain name. Here, the example, as we look at the different labels on this and go down, we start building a domain name. When we build a domain name, we build it from the bottom up. So here we have www.example.com, and then we'll talk about the hidden dot or the fully qualified domain in a little bit.

So with any domain that you see on the internet, if you take it, like example.com and you tilt it on its side, you'll see this exact same structure. So you can look at what the structure of a Domain Name System is and how it relates to the hierarchy of the domain tree just by tilting it on its side, and you can see the structure there.

Now, a fully qualified domain name means unambiguously identify as the node. So it's not relative in any other domain. For this example, we have www.example.com, which no one types the dot at the end, but in this example we put the dot. And that means to machines, "This is fully qualified. I don't want any uncertainty as to what this will be." So the Domain Name System recognizes that ending dot as the root, so it says, "Okay,

this is fully qualified, it can't be anything else other than exactly what you're typing here." Most things are unambiguous anyways, so we tend to leave the trailing dot off. But it's always there inherently. The DNS system just kind of inserts it if you don't.

So a domain is a node everything below it. For instance, the .com which is a top level domain, everything below the .com is in the apex of that domain. It means it's all part of it. So the very top piece of that apex, the top piece of the node has to have some kind of pointers and management to the rest of it. And it might not be a full management or a full pointer to it, but it'll at least help you give hints on how to get to the next step or how to get to the next level. We'll go through that when we do an example of resolution as well.

If we take this to the top, top level, to the root, the root is the apex of the entire DNS system. If we take it to a ccTLD, .uk, there are domains under UK, so that's the apex of that domain system as well.

We divide the namespace into distributed model. This distributed model allows for separate entities to manage the zone file, the node. We call those zones, and the distribution of that allows for reduction of a single point of management. But also, it allows for more rapid response on editing too.

So if you're an organization and you're example.com and you're putting in a bunch of servers, instead of sending an e-mail to your registry, you're the administrator for example.com and you would go in and you would make these changes to your own zone, and that would start propagating throughout the Internet. So you can make changes faster this way as well.

The entity that delegates a domain out is called the parent. So for my purpose, again I have conte.net. The delegating party in that relationship is .net. The child part of that relationship is myself who manages the conte.net domain. So there's always going to be a parent and child relationship up until you get to the root, because the root is the parent and there is no parent above the root.

So another look at the namespace, and here we look at the administrative boundary. With the distributed model of this, every level, there's going to be a different point of distribution on who can manage those domains or those zones. Starting at the top, we have the root zone. The IANA and PTI manages the root zone, and then with the collaboration with the root zone managers of VeriSign as well. We have the top level domains, TLDs. Those can include country code TLDs, ccTLDs, generic top level domains, gTLDs, any of the other top level TLD type entities out there.

And then beneath that, in this example we have example foo and bar. Those are all domains, and those are being managed by the registrant of those names. And then beneath that, we have other entries. It could be a server, it could be a subdomain, but those are all still managed within the administrative boundary of those domain names. This kind of shows the parent/child and the delegations.

Alright, so nameservers, they answer queries. A query comes from an application or from another nameserver or another resolver. We saw stub resolvers, we saw caching nameservers, and a nameserver can be both. It can be a resolver plus it can also be a nameserver. An authoritative nameserver has zone information for that domain. It's the authoritative source of that domain. It has complete knowledge of that zone, and that's where the zone manager will be putting that data in.

So it can be a definitive answer the queries. It should have multiple authoritative servers. That's mostly for redundancy sake. We'll talk about the relationship I think in the next slide on that. And it spreads the query load. The more servers you have – up to a point – for a zone, the better. The more distributed it is topographically on the network, the better the load distribution it can handle.

In that form, you typically have a primary nameserver. Sometimes it used to be called the master server. And then you have secondary or slave servers. The primary server is typically where the changes to the zone data are being made by the zone operator. So if I'm adding a domain or if I'm changing an IP address or if I'm doing stuff like that, I'll make those changes and submit that to the primary server.

My secondary servers, although they will replicate exactly what the primary does, they'll do that through a query and response process. The secondary servers will be going to the primary to get their updates, and then when they're queried by resolvers, they will be providing the same information as the primary server.

So once the zone data is distributed through the various authoritative servers, it's the same data. The resolver doesn't know or doesn't care which one is the primary nameserver on that. They're all authoritative.

The zone transfer is initiated by the secondary. Basically, a secondary server. You program it in, you authorize at the primary level, you authorize a server to be – you identify it and say this is an authorized secondary server.

At the secondary server, you point to the primary server, and then every once in a while through a job it will go and query the

primary server and ask if there are any zone updates. If there are, it will initiate a synchronization, and if there's not, it'll just get a new update and continue on as normal.

So the DNS standard specifies the format of the DNS packets sent over the network. Within that, the standard also specifies that the zone file is in a text based format. It's the master file format.

So for most zone files, you can look at the primary server or any of the secondary servers. If you have the ability to look at the servers within the servers themselves, you can look at the zone file and it's just a plaintext file. It could be just a couple of lines. For my domain, I've only got a couple of entries on there so it's quite easy to see, or it could be something like .com which has millions and millions of entries because of all the different domains that it's pointing to.

So it can be a very simple file or a very complex – as far as size – file goes. Both simple in what they're serving, and we'll talk about that in a second, but it's mostly about size too. Because of this, it's fairly easy to read. It's very easy to parse, and it was put in in this format during the specifications stage when they did standards for the DNS.

So if we remember, every node has a domain name. A domain name can have different kinds of data associated with it. We call

those resource records or RR types. We'll look at a couple of different ones in the next slide, I believe. A zone can consist of multiple resource record types, so you can have pointers that say, "My e-mail goes to these servers, www goes to this server, ftp goes to this server." DNSSEC has a resource record.

There are different types of RRs, resource records that you can put into a zone file, which allow the machines, the queries to come in to be more specific when they get a response on how to direct that traffic and how to handle that traffic that's going to ultimately go to that server. You can never mix resource records from multiple zones into the same file, so a zone file is entirely and completely speaking about one domain name, one zone.

If you're manager of multiple domains and you have the same nameserver for these domains, you can do that, but you have to have separate files for each domain. That way, it can manage that data better. So our resource record types, some of the most common ones. We have the owner, it shows the domain name that the owner is associated with. We have a time to live. This is about caching. We'll talk about caching after we talked about resolution, but you can specify the life of the information within that domain.

So if you're doing multiple changes on a specific server, you want to keep that caching low because that way it'll propagate

new changes quicker to the Internet. But if you have something stable, you want to keep it long but not too long so that the caching servers don't query it too often, but you do want that data to expire at some point, because at some point you might have a change and you want to make sure that the nameservers and resolvers will get that change eventually. We'll talk about default times and preferred times and things like that coming up.

We have classes. This is not really used these days, but there are different types of classes. Most that you'll see these days if you look into a zone file is called IN, Internet Class, but there were other ones developed. We have resource record types. These are different ways that we can associate data. We have MX records, we have A records, we have AAAA records. We'll go into a little bit more detail about that, but that's really the meat and potatoes of what's in a zone file, is the type, and that's where a lot of the resolution between naming and numbering takes place, is within the resource record type.

And then we have RData. It types the data of – the type specified that the record carries.

So in this, this is typically the syntax. Anything in brackets you can leave out. If you're submitting or if you're modifying a domain zone file, anything in brackets you can leave out and it

will inherit default values when it looks at that. But typically, we have owner, we have TTL, we have class, then we have type and then the data itself. Again, class is mostly unused. We see it as an IN class typically in zone files. And TTL, if you don't specify a specific time to live, it will inherit it form the master record, SOA record on that file.

You have to always put in a type and data, that's where your resolution takes place. So you would have – we'll show an example in a second, but for instance – www as type, and then the data itself would be the IP address of that webserver.

So common resource record types we have. As I mentioned, we have A records, those are called anchor. Anchors will resolve the name to an IPv4 address. We have what we call AAAA records that will resolve a name to an IPv6 address.

We have NS, which are a listing of the authoritative nameservers that are related to that domain.

We have the SOA, the start of authority. This is the first record type you'll see inside of a zone file, and this is a descriptive line and it'll tell a bunch of thing: who the operator is, it'll do a default time to live, it serves a couple purposes on that and goes from there.

We have a CNAME. This is an alias. Basically, if you have an A record, if I have www and I'm pointing to 192.168.1.3 and I also want that to be an FTP server, instead of putting in an A record on that, I can put a CNAME for FTP and I can just say CNAME FTW www, and it'll create an alias for that as well and it'll inherit the IP address from that A record.

MX records are mail exchange server records. They're typically used for when you want to send an e-mail to a destination. Before that e-mail is sent, it has to know where that destination is. So the e-mail server will do a query through the different layers [of the] DNS to find out where that destination mail server is. The way it does that is it says, "Give me your MX records, your mail exchange records." And the response on that, you'll get the list of IP addresses and domain names associated with the mail records for that domain.

And then you have not in forward domains, but you have a PTR record, and what we do in a typical domain is we map a name to a number. A PTR record is meant for reverse lookups, so you would map a number to a name so you can look up a hostname by doing a command and then typing in the IP address and if it's managed and put into that zone, you'll get a hostname out of that. Used somewhat, but not typically by an end user.

Currently, there are 84 record types as of August of last year. Most of them are special case. What we saw on the last slide is mostly what you'll see if you're looking at a zone file. So although there are a ton of different types, they are special case and not always used. We're starting to see more and more, with the coming of DNSSEC there are more record types that come in form there as well. But for the most part, you'll get maybe two handfuls of resource record types that as a common user or even a common application would use on that.

IANA has a resource record type registry. There's a URL here. This slide deck will be available in the schedule. I'll have this uploaded at the end of the session, if you're interested in not typing or scribing that very quickly you can go grab this presentation and just grab it from there.

As we know, IANA, one of its jobs is to be the location where unique identifiers are stored and kept. So even though the IETF developed the DNS and maintains the RFCs, the protocol parameters for this and the standards for DNS, there needs to be a place where things like resource record types can be listed. IANA is the keeper of that, so IANA actually holds a lot of registries. This is only one of very many on that.

This is an example of the IANA page that has the resource record types, and you'll see at the bottom down here the different

types. We talked about the A record, we talked about the NS record. We didn't talk about an MD or an MF. There are 84 different ones in there, we're not going to go into super detail on that.

Most common use of DNS is mapping a name to a number, as we said. You do that, either map the name to a v4 address, in this case 192.0.2.7, or we map it to a AAAA address, IPv6 address, and you'll get that. So you can actually have – in this case you see it's the same query – the same hostname going to both a v4 address and a v6 address as long as you have the A record and a AAAA record associated with that hostname.

Nameservers, you have to have a nameserver record on every zone. It needs to be in the parent zone, and we'll talk about that when we do resolution, but the parent zone is if you remember example.com, the parent on example.com is .com, and then the child is example.com. The only information that the parent has about the .com domain and subsequent zone is the nameserver records and any DNSSEC information that it might need.

And that's it, so anything else, the query has to go to that zone to get the information. But if you don't have a pointer from the parent to give to the child, then the query doesn't know where to go to get that information. So you have to have NS records at

the parent, but you also have them in the child zone too to show where the other servers are.

You'll see here that it's not an IP address. The NS record is pointing to other domains or other fully qualified domain names. You'll see the dot on the end actually on that one.

So as I just said, the parent – in this case the root – has pointed records to the child – in this case .com – and in the root you have listings for probably 13 gtld-servers.net. So .com in this case has 13 authoritative nameservers associated with .com, and so the root points a query to go look at those.

But there's a chicken and an egg problem here, because if you've never done the name resolution and you're pointing to a domain name and you have never done a name resolution and pointing to a domain name, how do you know how to get there? So you also have something called glue. I'm hoping that that's the next slide. Almost. Here we go. You have something called glue. Glue puts that together. Glue makes the assumption that you haven't gone to a specific domain name yet so you can't go and get the actual mapping data from it.

So what it says is "You need to go to these nameservers, but since there's a good chance you don't know how to get to those nameservers, I'm going to put this glue in – which are A records, anchor records – into the parent, and that glue is going to tell

you the IP address of those names that I just referred to." So here, we're saying example.com goes to ns.example.com which is kind of cyclic. It circles around itself, so the .com server won't know how to get to example.com.

In that case, we put the glue in to say ns1.example.com is this IP address, and ns2.example.com is that IP address. That allows the resolution to take place, the query and resolve model to take place without having gone to that server before, because it won't know how to get to that server without that glue.

Glue could be an A record or a AAAA record. If you have nameservers in IPv6 space, you have to have glue in that IPv6 space as well. You don't necessarily have to have a DNS server in v6 space to have AAAA records, but it's probably pretty helpful if you're expecting people to be using v6 and you've got v6 servers, you probably want to have at least one machine in the v6 network as well. So you would have a AAAA NS pointer on that swell.

So we talked about the stat of authority and that it kind of builds the case for the rest of the domain and it [hands] the information for a lot of the inherited information that if you leave out onto the other stuff. So here we have it's showing example.com, it's showing a resource record type of SOA, it's

showing the primary nameserver on that one, and then it says hostmaster.example.com.

That is actually an e-mail address, but the @ sign means different things in different places, so we can't have that @ sign in this piece here. So what we do is we swap out the @ sign with a dot, so hostmaster.example.com is actually hostmaster@example.com. So if you wanted to reach whoever is managing that domain, you would just swap that dot out with the @ sign and hopefully get to the zone administrator. It's not always the case these days, but that's how it was designed and developed.

We have a serial number. This allows you as the zone manager to understand when the last time you made a change was. Most serial numbers are in the form of a date and a sequence. Like in this example, we're looking at 2016 of May the 1st, or in whatever format you use. It could be the 5th of January, depending on what your date format preferred is.

And then the 00 is a sequence number. So if you make multiple changes throughout that day, instead of coming up with a new – a serial number has to be higher than the last one in order to reflect a change, so in order coming up with a brand new sequence number entirely, you would just take that date and

you would just increment that last number by one, that makes it a greater number than the one before it.

So when secondary servers come to it, basically they compare serial numbers and say, secondary will say, "Here's my serial number. Is it less than the primary serial number?" If the primary serial number is higher, then it'll say, "Yes, you're out of date. You need to come and get a new update." So at that point, it'll do the query to get a synchronization between the different servers.

We have different values: refresh, retry, expire and minimum. These mean different things to different parts of it, and in that, I think it's the – I want to say refresh, but that looks pretty low for the minimum cache. I have a couple DNS people here. Ed, I'll call you out. Where is the default cache, the TTL on that? Just yell it out, that's fine. If you have a default TTL, it's set up here, right? This is Ed Lewis, everyone.

EDWARD LEWIS: Hi. The full TTL for your data would be set up when you write out the zone file.

STEVE CONTE: Right. Would it be in the SOA?

EDWARD LEWIS: No. That's actually a piece of confusion over time. The last number there is the default for negative answers. If I say no, it's no for five minutes in that example. But there isn't a default for a positive answer.

STEVE CONTE: Okay. Thank you. Not what I was looking for. Not what I was expecting, I'll say. Thank you. There is only one SOA record per zone. It's always at the beginning of that zone file, and we've just talked about those values.

So now if we look at the CNAME type, we talked about the A record and the AAAA record. The CNAME type is a canonical name of the target. This is the assumption that we already created an A record for mail.example.com, but it's also not only our mail exchange server, it's also doing some other function within our network too. So we want to give it a different domain name.

We can't give it another A record because there's already a single A record for it, but we want to associate it with a different domain name. We use the C name, the canonical name. The way we do that is we put the A record name first, the mail.example.com, and we CNAME it to the host. I'm sorry, it's

the other way around. The [sum] host is actually the target. Which slide is the canon – yes, so mail name was the one that we put in the A record and the target is the new name on that one.

Creates an alias and it's don't do aliases on the right side of the record, so don't put an alias after an alias after an alias. There's always one A record associated. Always do aliases directly to that A record. Even if it's multiple aliases, have multiple lines always pointing to the primary, to the A record name first, and then go into the alias.

So when we send an e-mail – we're talking about e-mail now – e-mail servers need to figure out how to get to the mail server. Not just the domain, but the mail server that's going to serve that domain. And we do that through – in the old days, we used to just do address lookups because there was very little number of hosts on the machine so it would just go there and do it. But there was no flexibility on that. So if u wanted to change your mail server of if you wanted to have more than one mail server, it became very difficult – if not impossible – to do.

So DNS offered more flexibility by adding the MX record type. What that does, you can specify a mail server within that domain and you can specify a preference on that too. So in this slide, we see there are two mail servers associated with example.com. We have an MX record with a value of 10 going to mail.example.com

and we have an MX record with a value of 20 going to mailbackup.example.com.

That allows you to have multiple mail servers. The lower the preference number, the more preference it has. So in this case, the value of 10 is your primary mail server. This is where you really want your mail to come to. But for some reason if that's unavailable, either through route flapping or the server might be down for maintenance or something like that, you still want to get your mail. So we've set up a preference of 20 saying if you can't get to the other one, go to the second one next and put the mail there. And then the two mail servers will talk and synchronize mails from each other.

We all know about e-mail. Everything to the left of the @ sign is the user, everything to the right is the domain. So reverse mapping as I mentioned isn't used as much for an end user. You won't see it as much, but network administrators use it and some applications and different services use it.

What you would be doing, there's a zone file out there called in-adder.arpa. What that does is you would set that up as a zone administrator and you would put in stuff like this. So if we have this entry here, 7.2.0.192.in-adder.arpa as a PTR to example.com, what that is actually doing, it's referring directly back to the IP address but in reverse order.

So here we have 192.0.2.7 which is your actual IP address associated with example.com. Now if we take that backward, you'll see that that's how it's added into the PTR record. So any time you're looking for something, and then just like any domain, in-adder.arpa has a parent and a delegator.

In-adder.arpa is managed by IANA and they co-manage out piece of that to the RIRs so they delegate certain zones out to the RIRs. The RIRs then delegate chunks of the in-adder.arpa space out to the customers and then their customers manage that piece of it. So it's just like any other zone management thing, it's just for reverse lookups. We're not going to g heavy into that at all, but just the fact that it exists. There is one for v6 reverse lookups too, that's called ip6.arpa and it's managed the same way.

Alright, so DNS security. There is a more in-depth session on DNSSEC at ICANN meetings. If you have interest, I suggest going to that. This is mostly just a quick synopsis, a summary of what DNSSEC is. In some ways, it's kind of on this misnomer in the fact that traditionally when you think of security, you're thinking of encryption and things like that. DNSSEC doesn't encrypt the data, so in some ways DNS Auth might be more relevant to call this.

It's more of – it authenticates the data, the source and the distention of the data. It doesn't do any type of encryption that's not meant for that. It's meant to make sure that when you do a query, that the answer you're getting is from the target that you thought you were asking the question of, and it does that through [pairers] and through different levels of keychains and stuff like that.

Within DNSSEC, there are more record types now that we have. We have a DNS key which is the public key for a zone. With key management, typically you have a public key and you have a private key. You as the administrator of a domain will hold the private key, and then you publish the public key. And through various algorithms and methods, when you're doing resolution, you're comparing the public key and the private key together to make sure that what is happening is where you're meant to go.

Like I said, there's more of an in-depth DNSSEC session this week. I don't have it on top of my head. Probably Wednesday, maybe Tuesday. They'll go into much more depth on how the actual algorithm works and how the key comparison works.

But back to record types, we have RRSIG which are the digital signature data for the resource records for the DNSSEC.

We also have NSEC and NSEC3 pointers. These provide authenticated denial of existence. One of the things that you can

look up to see is if no domain exists. If you're looking for icann58.example.com, you want to get an authoritative answer if it doesn't exist as well as if it does exist. It prevents spoofing or malicious behavior. If you don't get an answer and it just doesn't give an answer, there's no certainty that the domain did not exist.

NSEC provides an authoritative answer that says, "This doesn't exist," and your server will say, "Okay, cool. Don't need to look at that anymore."

The DS record is a delegation signer. This resides in the parent zone. This is part of the chain of trust that happens. You have a DS record in the zone above you, so if I was signing example.com I would have to answer a DS record in .com in order to start building that chain of trust going downward. So when you go to the algorithms to determine the authentication model, that comes into play as well.

More resource record types we look at, here's TXT, there's URI, there's TLSA, there are other ones. I'm not going to go into them. These are special case ones, they're hardly ever used. And if they're used, they're used for specific purposes. There are a couple of examples, like TLSA is used by DANE, which takes DNSSEC authentication of DANE entities, associates them with certificates.

This is one really easy to see file form the back, I'm sure. Not much here, except this is an example of what a text file zone file looks like. This is a super basic one, this is for example.com. At the very top, we see the SOA, the start of authority record. And as we go down, we see our NS records. We have to have NS records, and they cannot be IP addresses, so we have NS records relating to fully qualified domain names.

Following that, we have an A record with an IP address afterward. We also have a AAAA record. We have a couple MX records. We have an alias, the CNAME, canonical name and then we have our glue which is showing the NS record name with an anchor, an A record to an IP address.

This is very typical of what a registrant's, an end user's domain would look like. There's really not a lot of rocket science to it. You're really just trying to associate a couple hostnames to the domain name. So you can have www, you can have FTP. Whatever hostnames you want, you put it in there. The higher, the deeper into the DNS tree, the longer those host files become, and more specific in the data type too.

So now we're going to get into the resolution process. Before we do that, I'll just pause for a second, open up for any questions about the boring bits that we talked about. Or do we want to get to resolution? Any hands? No? I put everyone to sleep? I've got

one hand here. Hold on, I have remote users so I need to have you hold the mic.

UNIDENTIFIED MALE: Thank you. This is likely going to be covered, but I just was wondering, the zone file that you showed, and you showed multiple zones. So there is such a file at every zone?

STEVE CONTE: In the zone file – can you go back a slide, Cathy? I can't reach it from over here.

UNIDENTIFIED MALE: So is there such a file at the root zone, and then at lower, like .com zone?

STEVE CONTE: Absolutely, yes. So this is the zone file specifically for example.com. There would also be a zone file for .com and there's certainly a zone file for dot, for the root zone too. And if there was a subdomain within example.com, let's say gov.example.com or whatever, and you wanted to have that delegation and managed by another entity, you can delegate that through at this level and then point NS records to that

subdomain and keep going down that chain. Cathy, are there any questions online at this point?

CATHY: We had one question from Jared. He wanted to know if we were going to be looking at intellectual property issues.

STEVE CONTE: We will not. This is purely technical, to discuss the DNS resolution process.

Okay, so we're going to talk about resolution now. If we looked at that model before, we had stub resolvers, we had recursive resolvers and we had authoritative name servers. So if we look at that before we get into resolution, there are two types of queries. Stub resolvers will send recursive queries where they say, "I need the complete answer or I need an error."

Recursive nameservers send non-recursive or iterative queries. Basically, they're saying, "I can do some of the lookup work for you, but I'll offer you a referral." And we'll talk about referral because that's a key piece of resolution, is basically saying, "Go look somewhere else." We'll talk about that.

High level algorithm, blah, blah, blah. Let's see, an exact match from local data if possible. If no exact answer, walk up that namespace and look for a referral or look for local data.

And if it's a recursive query, send the query to the nameserver for the enclosing zone – remember, we have the dots around the different zones – and keep following the referrals down that tree. And we'll see again what the referral process is.

Before we start, how do you start the resolution if there's no local data? Just turn on the server for the first time or you just built it and you have no cache at this point. So how does it know where to go? Well, there are hints file on the servers, and that hints file is very basic, it leads pointers to the root servers. So it has entries only for the root servers in that hints file. So a brand new nameserver or a cache-free nameserver still knows how to get at least to the root server.

It's through basically glue. It'll show the root server name, followed by the IP address. So it'll have a.rootservers.net with an A record and then the IP address for a.rootservers.net. It would have that for all 13 server instances, including v6 space too.

There's a sample of hints file you can get from InterNIC.net but it comes with every build if DNS software that I know of. At least it comes with a version of it. And the root server IP addresses don't change very often.

So we have the root zone administration, root zone manager. You might hear the term RZM throughout the week. That means the root zone manager or maintainer rather, sorry. It's a split function between – now it's PTI who runs the IANA functions operator and then VeriSign who's a root zone maintainer. Those two entities together create and maintain the root zone. And then we have 12 organizations who operate the root zone files.

We will have the root zone operators, RSSAC here this evening at 5:00, in c.1.2 I where we're having that session. They'll talk more about what their role as a root server operator is and how they interact with the root zone maintainers.

So even though there are 13 root server instances, there are 12 root server operators. That's because VeriSign runs a-root and j-root. There's no difference between the two. There's no difference between any of them, I should say. One of the myths was that a-root was more authoritative than any of the other root servers, and that's not really the case. They're all sort of the exact same data.

In fact, none of these are the primary server. These are all considered secondary servers. They use a methodology called the hidden master, which means that there's a server out there that will only talk to the IP addresses or to the instances of these root servers. They all act as a secondary. They all go query the

data to get the latest serial number and get the latest sync on this. The hidden master will only talk to the root servers, and it's meant that way so there is a data breach at all on any of the servers, the zone file that's authoritative isn't sitting in the public Internet. It's sitting behind protection.

So that way, if there's ever a compromise on any of these machines, they're all acting as secondary and only that machine will be compromised, not the entire domain name system. So if someone – I'm going to pick on l-root because it's ICANN – if someone compromised l-root and decided to add a TLD called Conte – I don't know why they'd do that but who knows – and started publishing data for .conte, well, only the l-root server would be publishing that data because it's not authoritative. It's not the primary zone data. It's only going to publish that until it goes out and queries and gets a new serial number, or until someone catches me and says, "There shouldn't be a .conte here." But when it goes and queries for the next serial, it'll make the realization that it's got an old serial number and it'll pull from that hidden master to get the new server data out of that.

Root servers have their own website, root-servers.org. It's a pretty interesting website if you're interested in DNS. It shows you – I keep calling them root server instances, and RSSAC will go into detail on this. There are actually more than just 13 root servers out there anymore. That used to be the case, but there's

a technology called Anycast – which we won't touch in this session – that allows you to basically mirror the root server instance using the exact same IP address and route announcements and things like that.

So what we used to have, 13 root servers, actual machines on the Internet running the root zone, we now have hundreds of root server instances serving that data around the world. And this is fantastic because it serves a lot of purposes. It balances the load, so instead of having 13 servers accepting the query load for the entire Internet at the root level, now you have hundreds of them around the world doing that. It balances the load because those root server instances are globally diverse. They're in exchange points throughout the Internet, through various parts of the world. They serve a lot of purposes and it's a really robust thing. It also protects against Denial of Service attacks for the root servers because of the way the technology of Anycast works. It becomes much more resilient, much more elastic in how it can handle specific types of DDoS attacks and stuff.

So if you're interested in seeing how many, go to root-servers.org. You can drill down on the map, you see yellow and green dots. Those are indicators of how many instances are in that location. We look at the root zone change process. In this case, the TLD manager – it could be anyone, it could be .com, it

could be .uk. A top level domain manager needs to make a change to their NS data, their nameserver data at the root level, they have to go through a process.

They submit the request to change, they submit it to the IANA functions operator, to IANA. IANA goes through process to figure out, to ensure the request is being made by an authoritative source of that zone. So I can't go in and make a change to .dk because I'm not – they go through steps to authenticate that I actually have the authority to make the request for change, and they go through that through various out of band methods as well as in band methods to ensure that.

Once all that process has happened, then they request the implementation and that goes to the root zone maintainer, to VeriSign. VeriSign puts – root zone maintainer puts that information into a database. That database then generates a root zone file. That root zone file then gets put onto the hidden master. They might be calling it a different word these days, so it might be called the root distribution server or something like that.

From that master, it goes out, and as I said the secondaries query the primary for updates. So I think it's every 24 hours. It could be I think maybe actually 12 hours. The root servers will go and query the primary server to see if there's a new update, and

it does that by comparing serial numbers. So when the root zone database generates the root zone file, it increments that serial number to make sure that it's a higher number than what's currently being served on the public Internet.

Alright, so let's go through resolution process. In this scenario, the phone is configured to send queries to a recursive name server with address 4.2.2.2. We'll say this phone is using Verizon services and it wants to go and make a query. Verizon services would be running the recursive nameserver within their network. Verizon would be acting as the ISP in this case on the phone.

In this case, the phone is the stub resolver. It's the one asking the question, it's making the query, and it's looking to go to www.example.com. But it's never been there before, has no chance and it doesn't know how to get there, so it needs to ask the question.

It has a hints file. No, it doesn't. I'm sorry. In its IP configuration, it has pointers to its DNS servers.

If you've ever looked at your laptop and opened up your network configuration, you'll see your IP address, you'll see your subnet mask and then you'll see one or more nameserver IP addresses. That's how it knows how to get to the recursive nameserver to ask the question.

So it goes to the recursive nameserver and it says, “Hey, I don’t know what the address of www.example.com is. Do you know that?” In this example, we’re going to assume that it’s a brand new nameserver that’s never gone anywhere on the internet before either, so it doesn’t have any cache, it doesn’t have any data other than what is in its hints file. So in this case, it says, “No, I don’t know how to get there, but I know how to get to the root server, so let’s ask the root server how to get to www.example.com. But I’ll do this on your behalf because I’m a recursive name server. I’m going to act in proxy for you.”

So it goes to the root server, in this case l-root, and says, “Hey, what’s the IP address for www.example.com?” And the root server says, “I don’t know, but I know how to get to .com servers. I have the NS records for .com servers, so why don’t you go ask them?” So it passes the IP address for the .com servers back to the recursive nameserver who’s acting on your behalf.

Recursive nameserver says, “Hey, cool. I’ll go ask the .com servers.” So it goes and says, “Hey, .com servers, what’s the IP address for www.example.com?” And the .com, the TLD server says, “I don’t know, but I know how to get to example.com nameservers, so why don’t you go ask them? And here’s the IP address for the example.com nameservers.” Your recursive nameserver says, “Oh, cool, thanks. I’ll take that referral and I’ll go to the example.com nameservers.” And now it says, “Hey,

example.com nameservers, I'm looking for the address of www.example.com."

And example.com's nameserver says, "I know that. I'm the authoritative source of that information. I'm going to give you the IP address for that. Here's the IP address." Going back too far. "Here's the IP address or all of the IP addresses associated with www.example.com." Saying all the IP addresses because it might have some AAAA addresses too so it might give you that data.

All that goes back to the recursive server. The recursive nameserver says, "Cool, thanks. I'm going to pass that back to my end user. Here's the IP address for www.example.com." And now finally, it goes into the stub resolver, into the application – in this case Safari – and then it will talk directly – using that IP address – to that server.

That's a pretty long journey for a question and answer [inaudible]. It had to go through at least five different servers to get the answer. And the beauty of the Internet is that that might have taken maybe 200 milliseconds, so it doesn't take a long time to get this answer, to get this process going. But it kind of all adds up, so one of the things that happen is that you have caching. Caching remembers the core answers that it has been asked before.

Now instead of going to example.com, we might be wanting to look at footwear and want to go to nike.com. Well, there's no reason to go and ask the root servers for the address for .com anymore because it already has that cache, so it knows that answer. So caching will take steps out of the process. Or conversely, in this past example we were a cell phone user and we said it was on Verizon cell phone service. There might be another Verizon user out there who's also trying to get to www.example.com. The recursive nameserver holds that cache, that information for a set period of time. So if someone else is trying to get to www.example.com within that TTL, that time to live, it'll give that answer right away. It'll do no queries beyond that because it says, "This data was authoritative and I'm holding onto it until that TTL expires, at which point then I'll go ask the questions again."

So if we look at caching process, so we've already been to www.example.com. Now we're going to ftp.example.com, so we type that into our browser. The browser goes through a stub resolver and it sends the data up to the recursive nameserver and says, "Hey, I'm trying to get to ftp.example.com." Well, as we've already said, the recursive nameserver has already been to the root, it's already been to .com, and it's already been to example.com, so it has all those answers cached already.

So all it really needs to get to is example.com, the nameservers, and say, “Hey, remember me? I asked for www, now I’m looking for ftp.example.com.” And ns.example.com will return that address and send it back to the recursive nameserver, who sends it back to the stub resolver, who sends it back to the application, to Safari, who then goes and uses that information and talks directly to the server.

So we missed all those other spots now, those other stops. We missed talking to a root server, we missed talking to a TLD server. That saves time for the user and it saves bandwidth for the TLD operator or the root operator. It doesn’t sound like a lot for one query where we’re talking bytes, but when we’re looking at an Internet where the queries just aren’t for humans anymore, they’re for machines, we’re talking about millions and millions of queries per second at the root level.

We’re probably talking about hundreds of millions, if not billions of queries at the TLD level per second. So the less that we can go out to the root server, to the TLD to query, the better it is both for the end user but also for the manager of that zone as well. Any questions about resolution before we go on? Yes.

UNIDENTIFIED MALE: Hi. If I buy a new domain, myname.com, are they going to go to the gTLD and add that for that domain, www.myname.com, it

should go to a nameserver for the company I bought from and then they know where they have hosted my site?

STEVE CONTE: That's an interesting question. If I heard this right, if you get a new domain, how do you put in the – what does the registry do? Okay, so this is a good example of where we are too then.

So you as the purchaser of a new domain, you're the registrant. The registrar is the company that you bought the domain through. It could be GoDaddy, it could be whoever. Exactly. The registry is the top level domain in that space, so if you did myname.com, the registry would be .com, which would be VeriSign in this case.

Now, in order for the Internet to be able to see your zone data, first of all you need to have set up or have managed a domain name server, an authoritative nameserver out there to host myname.com.

You need to create pointers within .com to point to myname.com. You would do that through typically your registrar. Once you get your domain name, there's a place where you can manage your namespace on that. You would point those nameservers that you created to an IP address in the registrar, and it takes that data and it creates NS records for you.

Now, where that nameserver resides is really up to you. If you're a business or you're technically adept, you could create and house your own authoritative nameservers and manage those yourselves. If you just want to use someone's services, there are a ton of people out there, a ton of businesses out there who will not only host your website but they also run DNS services for you.

You could mix and match and run nameservers from one reseller and website off another one and your e-mail off another one. As long as there's a nameserver out there that's pointing to myname.com, everything else can be managed within that zone file. Does that make sense? Okay. [Go ahead.]

UNIDENTIFIED MALE: Thank you. Out of the 12 organizations that manage different root servers, which is the one that actually gets the most consultations from and which is the one that has the most servers distributed?

STEVE CONTE: Out of the 12 instances, which gets the most traffic, and what was the other part of the question? And more instances, okay. The one that gets the most traffic is actually a great question. I don't know. The more significant question might be which one

gets the most traffic in a specific region, because some of het instances are set up to be strategically more responsive in certain regions that were typically underserved in historical periods.

Back years ago, Africa, Latin America, parts of Asia would have to use satellite link to get to some of the larger networks out there, tier 1, tier 3 networks to get to the root server. That path would be expensive because they're using satellite traffic, and long because they're going topographically and geographically to different location. One of the reasons to use instances in Anycast was to bring those root servers closer to the end user so they didn't have to do those large hops to get to the answer.

So strategically, they put them in Internet exchange points in countries. Those exchange points might be only serving that specific region, so they might be getting a bunch of queries but only from that region. So I think getting a number of – the question of who gets the most queries is not a fair question to ask because it's meant to be distributed. And it's meant to be dynamic, so if their servers are down or if routing is better one day to go to one place, your number of queries are going to vary depending on that.

Who has the most number of instances, I'm going to say ask RSSAC this afternoon, but I think l-root right now has the most

instances. But don't hold me to that at all. I know there's a number of root servers that work really hard to create – it's not a race to get the most instances, it doesn't make one root server more robust or better than another root server, it just means that there are either resources or a plan available to distribute that instance out to more locations. To the end user, it makes no difference. All the data is the same that's being served. Rather, it's l-root or a-root or k-root or whatever. Does that answer?

Any other questions on resolution or anything else that we've talked about so far? Cathy, anything online? Okay. So as I just mentioned, we have different levels of human interaction and agencies within the domain name system. The registrant is typically the end user. It's my mom registering a domain name, or anybody else. The registrant will register that domain name either through a reseller or through the registrar who the reseller also has an agreement with.

Once the domain name is registered through the registrar, then the dialog to get the nameserver information into the zone, into the TLD, happens between the registrar and the registry. Registry is the top level domain on that.

So if we look at registry, they serve a couple of different functions. First and foremost and most important is that they handle the authoritative data for that top level domain, for that

domain space. They have all the NS records to all the domains, all their child domains below that.

So for myname.com, for conte.net, for example.dk, those would all be handled by a registry. But along with that, along with running authoritative nameservers for their space, they're also providing a couple of other services too. They have WHOIS information which may be at the registry. It might be at the registrar. It depends on the model of WHOIS that was agreed upon when they took over their registry or when they ran that registry.

So there's an Internet user interface to that. They might be running an RDAP service, which is another methodology of looking up information, either WHOIS type information or other types of information on that. And that would be all public facing stuff, along with the dialog between a recursive nameserver. Inside, they're going to have a database of some kind, and that data base is going to be probably building a zone file for their TLD. It's probably going to be touching customer data at some point. Might not be the same database, it could be a collection of databases.

It's going to have some kind of API or hook between the registrars they have agreements with and the registry. Typically, they use an extensible provisioning protocol, EPP, to have a

method of transportation of the data between the registrar and the registry.

And then they have a one-off relationship the registrant. So if my conte.net ever goes down, I'm not going to contact .net necessarily, if it's not my fault, which typically when a domain has an issue, a lot of times it is at the registrant level with some kind of configuration issue.

But if it's not, then I would go to my registrar, and as a customer, they're my point of entry into that space and I would discuss with the registrar my woes with my domain. Very seldom – I'm trying to think if there's any situation, and I can't think of any one situation specifically – would I as the registrant contact the registry directly. Most of my interaction – there is a case? I want to hear this.

UNIDENTIFIED MALE: Sorry for my voice, I have a cold. I'm from .dk host master, and the .dk domain, it works as we have a sole registry, not a shared registry. So if the registrant have a problem, they usually contact us directly and not the registrar.

STEVE CONTE: Do you act as the registrar as well for your services, or do you have registrars that sell .dk space?

UNIDENTIFIED MALE: We have registrars who sell the domain names. You cannot register a domain directly through us and we do not offer DNS and name service, but we do support if you want to change your DNS provider and so on. So we have the contact with the registrant himself.

STEVE CONTE: Well there you go. There's a case. I will now revise my slides. Thank you. That's interesting. Do you get a lot of – I just took the microphone away from you, so yes or nos are fine. Do you get a lot of registrants who come to you with issues?

UNIDENTIFIED MALE: [inaudible]

STEVE CONTE: About 400 calls, about 400 e-mails. And how big is your registry overall? 1.3 million, so that's a pretty small number of registrant interaction compared to the number of total registered domains. But that's still very interesting. Thank you for contributing to that. Any other questions while we're...

How are we doing on time? 12:30 we're done? Oh, look, we are done. Alright, so any further questions? We are at time, but I'll

hang out for a couple minutes here if anyone has any comments or questions.

Going over there, our next session will not be in this room. It'll be in C 1 2 I think. At 1:45, we're going to have a session on Internet networking.

UNIDENTIFIED MALE: Hi. I looked at the map at root-servers.org and I come from Afghanistan, and there was no root server in the country, but we had root server in the neighboring country. So my question is, when do we make root servers? How do we make them? What qualifies to have a root server in a country? I could see some countries had multiple. So, how does that work?

STEVE CONTE: One thing to keep in mind is that network topology doesn't always match real world geography. Just because there might not be a root server in your country doesn't mean that there might not be a root server who's topographically close to you. It's kind of a fickle distinction but it's a true distinction. Network doesn't care about geographical boundaries.

However, to answer your question directly, if you feel that your country can be served well by having a root server in country or if you have an IXP, Internet Exchange Point in your country, you

can contact the root server operators. Terry Manderson from l-root is here. I know there are other root server operators coming to the RSSAC session at 5:00 tonight. You'll see a lot of the root server operators there tonight. Contact them directly and express why you feel there should be a need for a root server in your country, or let's take away the word country and say within your topology or within your IXP, within your region, and why it's an important thing. And then they can either discuss different models on how to get a root server there and the viability of whether or not it would really make sense to have one.

Because if there is one topographically right next door, again country non-specific, then they might say, "Well, we're serving that region with these eight instances and your response rate is still really low." If you can make a case against that to a root server operator, then they might provide you a couple different models on how to get a root server there. Alright, any other questions?

Alright, well thank you all for attending today. Like I said, this slide deck will be posted momentarily. As soon as we're done here, I'll post it up onto the schedule. Our next session will be Internet Networking with Alain Durand from ICANN and Jeff Houston from APNIC. We will be changing rooms. We will be in I think C 1 2, maybe C 1 3. C 1 2, thank you. And that'll be at 1:45.

And then after that – that sounds late. Is there one before that? No, that's right.

UNIDENTIFIED FEMALE: [inaudible]

STEVE CONTE: Okay, so at 1:45 we have Internet Networking. At 3:15, we have Understanding DNS Abuse with John Crain who is our Chief Security Officer, Chief CSSRO. Chief Stability, Security and Resiliency Officer at ICANN. And then at 5:00, we have the RSSAC talking about the root servers. All the rest of the day will be at C 1 2.

Alright, well thank you. Enjoy your lunch. Take care.

[END OF TRANSCRIPTION]