MONTREAL – How it Works: Understanding DNS
Sunday, November 3, 2019 – 08:45 to 10:15 EDT
ICANN66 | Montréal, Canada

STEVE CONTE: Good morning. I was hoping for a larger room since we gave you an extra hour to sleep but I applaud you guys for showing up. Welcome, this is the first session of the How It Works Series that we'll be doing all day today. We've got four sessions today; we'll be starting with DNS.

We have Introduction to RDAP later on, RDAP is next in this room at 10:30 to 12, and then at 3:15 in the main room we'll be doing a Root Server Operators Presentation with RSSAC, and then at 5 o'clock today in room 512G, which is also on the schedule, we have Aaron presenting Understanding the RIR, the Regional Internet Registry. I know that's kind of a late night or a late session on a Sunday but we got moved around and please consider attending because it's brand new to us, the RIR's have never done a How It Works Session before and we'd like to give them the support they deserve on this.

With that, I'm going to present to you Matt Larson, who's our VP or research at ICANN for Office of the CTO and we're going to do the DNS Fundamentals. Matt, my understanding is that we're going to take questions as they come up. If you have a questions while Matt's speaking, go ahead and raise your hand, I will run up there with the microphone and make sure that you have a mic to ask the question because we are recording this, we'd like to capture the questions and make sure that we have the full dialog. If you have a question, please

raise your hand, either I will see you or Matt will poke me, one of the two and we'll get a microphone out to you. With that, Matt Larson, please.

MATT LARSON: Good morning, everyone. Thanks for coming out early on a Sunday morning. As Steve said, just raise your hand if you have questions. If it's a quick one, I can just repeat it for the record as well. We've got plenty of time for the material that we have., so let's get started.

I want to give you a little bit of history and background to explain where DNS came from and why we have it. The really most important part is, IP addresses are easy for machines but hard for people. If you can remember an IPv6 address you're better than I am, your memory is better than I am. I used to have a bunch of IPv4 addresses memorized for various networks that I worked on, but that's just engineering geek thing to do.

People can't use numbers; people need names and so that's why we have DNS and that's why we had its predecessor that I'm going to talk about. Because things on the network, if you want to reach things on the internet, you need a name. In the early days of the internet, names were simple, they were what we would now call a single label name. They had a maximum length of 24 characters and these were called host names. There were no domain names, there were no dots in these names because there was no DNS yet, DNS hadn't been invented.

This process of mapping names to IP addresses, so that a human can use a name and then an application can turn that name into an IP address and then the application uses the IP address, that process is called Name Resolution. In the early days of the internet, Name Resolution was very simple, it used what we call host file, and for historical purposes, it's called HOST.TXT.

If you're familiar with a modern Esty host format on Unix/Linux, this is the same ides, it was a different format but the concept is the same, it was a plain text file and it had a list of host names and a list of IP addresses. And this was centrally maintained by an organization called the Network Information Center and again, for historical purposes, for what it's worth, that was at the Stanford Research Institute, they had a government contract to run that.

The update mechanism was extremely low tech. Network administrator, whenever they made to their network, it would need to be reflected in host file. This would be for example when they added a machine to their network, when they took a machine off the network, when they changed a machines name or they changed a machines IP address, all of those actions would have to be reflected in the master host file and so they send an email to the NIC and says, "Hey, I'm making this change."

On the NIC's end, it was also extremely manual, they would literally bring up the file in a text editor and make the change, there was no database behind the scenes that they updated that generated the text file, no it was completely a low-tech solution. Because remember,

we're talking about the days when the internet was very, very small, we're talking thousands and then tens of thousands of hosts, which of course, compared to today is much, much smaller.

The way this worked is ideally, everybody had the latest version of the file but of course, they never did, you were always a little bit out of date because the NIC released a new file once per week and this was downloadable. The idea was you would use the FTP protocol to get a new version of HOST.TXT, maybe you set it up to happen automatically, maybe you did it whenever your thought of it but that's how it worked and all the lookups were made locally on your machine in this file. I just want to stress this concept because it's so different than what we have today, literally the name and IP address of every machine on the entire internet was in one place.

There were obvious problems with this, you can just imagine what they were. One of these was what we call naming contention because as I said, the updates were made manually, there was no good method to prevent duplicates and remember, we're talking about a maximum length of 24 characters, so that sounds like a lot but as the internet grew in size, you had to choose more and more obscure names to not conflict with anybody else. Only one person can have Frodo, only one person can have Gandalf, all the good names get taken and then you have to have more and more cryptic names.

Synchronization was also a problem, nobody ever had the same version of the file, it was just impossible. You were always a little bit out of date. The traffic and load just to move this file around, as the

internet got larger and the file got larger, it took more and more time to transfer it and it took more and more of the actual bandwidth on the internet, just moving around the HOST.TXT file.

I'm told, this is before my time, not much before but before, I'm told that at the end of the HOST.TXT life, the time it took to download the file was longer than the update period that the NIC released the file. It was impossible to get a current version of the file your file was not done transferring by the time the NIC had released a new version.

Clearly, a centrally maintained host file did not scale and something had to be done to replace it. The discussion started in the early 80's on a replacement, just to set the stage for the timeframe. The goals were to address the scaling issues I've just described and also to simplify email routing. I'll talk more about how email connects to DNS later on in the presentation but that was another goal behind replacing HOST.TXT. The result, as well know, is the domain name system.

If you had one slide to summarize DNS, this is the one slide that I have anyway, my DNS in a nutshell slide. Fundamentally, DNS is a distributed database. This database is massive and it's massively distributed, it's distributed over the entire world. The way this works is the data in the database is maintained locally by people who own their partition of the data, your portion of the data, the names corresponding to your network, you maintain local to you and everybody else does that and it's available globally. People can look up data in your portion of the network and you can look up data in theirs.

DNS follows a client server model. The client is what we call resolvers and the main thing to remember about resolvers is resolvers send queries. Resolvers send requests for name resolution. They send the request to have a name mapped to IP address or something else. Name servers receive those queries and name servers answer queries, they're the server side. That's kind of an important concept, that resolvers send queries and name servers answer queries.

There are some cool optimizations to DNS. Caching improves performance. Because we're talking about a database that distributed over the entire world, if you want to look something up in this database, it can take awhile because the speed light is only so fast and what if you have to contact multiple places to find the ultimate answer to your query. It could be hundreds of milliseconds away. It can take some time to answer a DNS query and so by caching the answer, you can remember the answer that you got and you can remember also the intermediate steps so that the next time you're going to do a query for that same piece of information or something similar, you have cached information stored locally to speed up that process.

The other thing DNS uses is replication to provide redundancy and load distribution and what I mean by that, is I said remember, you've got your local copy of all your DNS data but by replication I mean you don't have one copy, you have multiple copies and you want that for redundancy because if you had only one copy of your information available and somebody trips over the power cord and powers off that machine, then nobody can look up DNS data for you but if you have multiple copies that's not a problem. Load distribution means, that if

you're getting a lot of DNS queries, you can have the load spread out among these multiple sources of the data that are answering queries.

I think at this point a picture would hopefully help a lot. This is an overview of the major DNS component. I should point out here that really, this whole presentation, I'm talking about DNS from the resolution side, I'm specifically not talking about the registrar, registrant, registry ecosystem, that's not a focus of this presentation. Basically, this presentation picks up after a registrant has registered a domain name and set it up and has the domain names information in DNS, that's sort of where this presentation picks up.

Let's look at this picture starting at the lower left. We have there a phone, we have a device, it could be phone, could be a computer, we use to joke ha ha refrigerators when your refrigerator is on the internet but now you can't really joke about that because there are refrigerators on the internet. The point is, that's a device, it's some device that needs to turn names into IP addresses or names into other information because as we'll see, you can store other things in DNS besides IP addresses.

You map names to other pieces of information but the main purpose behind DNS, the main reason for its existence is mapping names to IP addresses. Here we have some device, in this case a phone, we have some app on this device, that's the icon for the safari web browser and that web browser wants to map a name, let's say a name of a website to the IP address of a website. That application calls a piece of code

called a stub resolver and a stub resolver is a DNS client, remember, resolvers are clients in DNS, that's what we call them.

The stub resolver is very, very simple, its job is to accept that request from the application and then get an answer for the application. But as we're going to see, it doesn't do most of the work itself. You'll find stub resolvers everywhere you find devices that have applications that need to use DNS and use the internet, so that means stub resolvers are everywhere. They're usually a part of the operating system, they're basically a service that the operating system has to provide programs on the device. You could combine the stub resolver in an actual program itself, but why do that, you don't need to rewrite the code, it's already been written for you, it's already part of the operating system.

You'll notice I have different colored arrows here; the red arrow is representing that that stub resolver is code and that the application calls that code as part of running. The stub resolver's job then is to turn that applications request for DNS data into to an actually DNS query. The stub resolver, its configuration is very simple, it has one or more IP addresses configured of what we call recursive resolvers and that's what we have in the middle of the screen.

If you've ever gone into the DNS properties page on Windows or if you've ever changed your DNS settings, that is the configuration for the stub resolver on your device, what you're putting in there are IP addresses of recursive resolvers. The stub resolver it assembles a DNS query and it sends it to its configured recursive resolver. Recursive resolvers are considerably more complicated. They do the actual work

or looking up the answer, they know how to navigate the distributed database that is DNS, they know how to search for the answer and they search for the answer in what we call authoritative name servers.

Remember, name servers are the server side of DNS, they hold all the information and that recursive resolver might have to contact multiple authoritative servers to answer this query. It might contact one and it might say, "Oh, well I have partial information but I'm going to refer you to somebody else." And they it would contact another authoritative server and that might say, "Well, I have still more information but I'm going to refer you somewhere else." And all this takes time and work and complexity on the recursive resolver's part. Eventually the recursive resolver will get an answer to the query and maybe the answer is that the information doesn't exist or it has the information requested.

You can see at the lower part of that box in the center of the screen notating that there's cache, this is what I spoke about just a moment ago. The idea that you can remember information that you've looked up to use it in the future. All of the responses that are coming back from the authoritative servers, the recursive resolver is caching those, is remember them for future use and then it ultimately sends the answer back to the stub resolver, on the lower left and the stub resolver than passes that information back the application, in this case the web browser and away it goes because it's not turned the name of a website into an IP address and it can contract the webserver and do its thing.

At a very high level, those are all the components to DNS and we're going to expand on those as we go on but I think it helps at this point, hopefully, to have that high-level picture to see where we're headed and what gaps we're going to fill in.

Let me talk a little now about the structure of DNS. I said that it's a distributed database but what do we mean by that? The structure of this database, I'm going to have to geek out a little bit on you, do a little bit of computer science stuff here. The structure of this database is what we call an inverted tree and that inverted tree is the name space. The name space is the DNS distributed database, that's the name we give to it and its structure is an inverted tree.

Now, you might be familiar with other databases, like say a relational database. A relational database has multiple tables and each table has rows of data and they're different columns for different kinds of data. That's the way a relational database is structured. The DNS database, the name space, is structured as this inverted tree and what I mean by that is that a computer scientist tree has the root at the top and the branches grow downward rather than up, that's the inverted part.

The root, at the top, I should say the root is at the top and then we have our called nodes in this tree. Each box that I've represented there is called a node in the tree. Each node has a label, it has a name. We often name, we refer to these nodes in the name space relative to the root. We talk about nodes that are immediately below the root in the top level and one level down, we talk about those being second level

and third level and so on. We're talking when we say level, we're talking about their position in relation to the root.

The root is special, it has not label, it has the null label. Sometimes you see it written as a dot like and I've put the dot in quotation marks to show that the roots label is not really a dot but we have to show something. The legal characters for the label names that you put in here are what we sometimes refer in shorthand to as LDH, Letters, Digits and the Hyphen, those are the only legal characters. The maximum length of one of these of these labels is 63 characters and you cannot have to worry about case, upper case and lower case are compared equivalently, in other words, not case sensitive.

The point of these label names is that together all of the label's names make up a domain name. We all know I think what a domain name is but I'm going to give you a more rigorous definition of it right here. Every node in the name space has a domain name that's like the full name of that node. Its label name is its short name but the domain name is the full name for it. The way we get a nodes domain name is you start at the node itself and you write down its label and then you put a dot and then you look at its parent, you go up in the inverted tree and you put that label down and you put a dot and you keep going up the tree, writing label names and that get's you the domain name.

As you can see on this slide, we've got the domain name www.example.com and then after the top level label, after com in this case, you would put a dot and then you would go up and you would see the root, the root has no label, it's the null label, so what you

basically have is com and dot and then the null label. Well, you can't write the null label so you end up with a dot at the end of the name and that's sometimes why you see the root node written as a dot because a domain name, strictly speaking ends in a dot.

By doing this, you can identify the position of any node in the name space, it has a domain name. I don't have slides for it but if this analogy helps, a file system, a modern file system whether its Window or Unix, that is a data structure that is also an inverted tree. The root directory is at the top, in Unix it's slash, in that case the root has a name and the name is slash and then you have files and directories. When you would get to a particular file or directory it's absolute path name, it's full path name starting with root is like a domain name. If that comparison helps, you may be familiar with an inverted tree data structure and didn't realize it.

What we call fully qualified domain name is a domain name that goes all the way to the root, that's the one I just described. You could have a domain name that's only partial, if you knew that you were talking about a domain name in com, you could talk about www.example but most of the time when you see domains, they're fully qualified domain names. They're the entire name of the node, all the way out to the root and including a dot at the end.

A domain then, is a related term and domain is a node in the name space and everything below of what we would call its descendants. Also, when you're talking about inverted trees, sometimes you use family terminology, parent and child. In this example here, if we look

at literally at example there on the right, its parent node is come and it has two children www and mail. In this case, I've highlighted the dot come domain.

Last I looked there are 140 million plus names in dot com, I'm showing three. Obviously, I can only show a portion here to make my point on this slide. This slide is just a small portion of the name space to illustrate what's going one. The com domain is com and everything below it. We could talk about the example.com domain, that would be that example node and everything below it, which is just two nodes in the name space. That's what we mean, strictly speaking, when we talk about a domain.

Now we get to an important concept, zones. Remember I said that DNS is a distributed database and the reason it's divided up is to allow distributed administration, in other words, everybody gets to administer their own data because as I describe to you with HOST.TXT, that was a scaling disaster. When there was only one organization managing everyone's data centrally, that didn't work, so distributed administration is key to DNS. It was one of the important design goals and it's what has allowed it to survive and grow so large. Everybody maintains their own data.

When I say DNS is distributed and broken up, it's broken up into what we call zones and zones are administrative divisions. You would manage one or more zones if you have one or more domain names. There is this process called delegation that creates zones and I need to just give an example to explain this. Here is the name space and it is

divided up into these administrative regions called zones and you can't look at the name space and know where the zone boundaries are. Zone boundaries are administrative, humans have decided where to put those boundaries.

I'm going to just fill in what I know the boundaries are in the name space as it's used every day. At the top we have the root zone and below the root zone every top level, what we now know is a top level domain, every top level domain has its own zone. Every third level domain, well at least here on the slide, every third level domain has its own zone. What I mean by delegation that I talked about a couple of slides ago, delegation is the process that creates zones.

One zone at level in the name space can delegate and create a zone below it and then that zone can create zones below it and so on and so forth. For example, the root zone delegates to dot.com and the dot.com delegates to a 140 plus million subzones below it. A subzone of com, like example.com, it could keep delegating if it wanted. The parent zone retains information pointing to the child zone. In the root zone is information about all the top-level zones. For every top-level domain, which corresponds to a zone in this case, there is information about those.

The root zone, at last count there are like 1200 and some TLD's, the root zone is not very big in the scheme of things, it only has to have information about the 1200 and some zones that it delegates to. I might have that number wrong and now I've said it for recorded for ever. It's a number between 1200 and 1600, now 1500 and change is

what's coming to mind. Sorry for everyone hearing the recording and hopefully I don't look to dumb. Anyway, not very many in the scheme of things, the root zone is relatively small.

The com zone, with its 140 million plus delegations is huge, it's just vast, it's got a lot of information. The com zone is the largest zone on the internet by far, by several factors. But there are other top-level domain zones that have a lot of delegations as well, like well into the tens of millions and they're relatively large as well. The point here is that zones can be completely varying sizes, they can be very tiny. The example.com zone, if that's literally all that's in the example.com zone, it only has information about three domain names in it, it's very, very small.

Where zones get important is that—I've talked about name servers and how name severs answer queries, well name servers know about zones. You have DNS information for a domain in a zone and then a name server loads that zone, reads the zone, what we'll call a zone file, we'll talk about that in a minute and then it knows about everything in the zone and it can answer queries about information in the zone.

This is what we call an authoritative name server, a name server that is authoritative for a zone, has complete knowledge about everything in that zone. You don't ever load part of a zone or only know part of, you know about the entire zone. Authoritative means in contrast to a recursive resolver, could have information cached from an authoritative server, it could look something up in an authoritative sever and cache it and remember it but its data is not definitive, it has

a copy of the data that's good for a little while. The definitive authoritative source of that data is the authoritative name server. The recursive resolver is only caching it for a little while.

Zones really should have multiple authoritative servers. This is the part that I said earlier when I said that DNS uses replication to achieve redundancy and load distribution. The redundancy is for your zone, you have multiple authoritative servers. Those are multiple places that somebody can send a query, a resolver can send a query to ask questions about information in your zone. Obviously, if you have more than one, you can tolerate the failure of one or more, you can spread the queries out among all of those authoritative servers.

There's a lot of words on this slide, I'll just explain it. If for a given zone, you're going to have multiple authoritative servers, how do you synchronize them? How do you keep the information about a zone in sync across all the authoritative servers? The nice thing about the DNS protocol is that it has a way to do that for you. It has a way to synchronize authoritative servers. You don't have to copy the information yourself, you just have the definitive information on one server, that's called the primary and then the other authoritative servers are called secondaries or sometimes they're called slaves and they contact the primary and they download the zone and then they have an equivalent copy.

The primary and the secondaries, they all have the same information so they're all authoritative, they're not degree of authoritative, you can't be more authoritative or less authoritative. The only difference

between a primary and a secondary is where it gets the data. The primary name server has the zone information in a file on its disc, whereas the secondary has to contact the primary to get the information for the zone but the information itself is the same.

Let's go into still a little more detail here. Let's crack open a zone, let's look inside a zone. What is inside a zone? So far, I've been waving my hands and saying its DNS data but let's talk about the actual data itself that makes up a zone. Remember, every node in this name space has a domain name and a domain name, a node or a domain name, depending on how you want to look at it, it can have different kinds of data associated with it.

That data in DNS we call resource records. A resource record is just a piece of data in DNS and sometimes you see that abbreviated as RR. There are different records types for different kinds of data because DNS as I said, it can store all different kinds of things. Everybody thinks of mapping names to IP addresses but there are other things that you can store. A zone is nothing more than multiple resource records. The zone is simply the sum of all the resource records and all of those records go in a file that we call a zone file and every zone has at least one zone file. You never mix records from multiple zones in multiple files, there is a file per zone.

I have to talk in just a little more detail to the upcoming slides make sense. Resource records have five fields, they have five components to them. The first is simply, what is the domain name, the owner name? That's the domain name that this resource records belongs to? Then

there's a time to live value that controls how long a recursive server can cache that data, I'll talk more about that later.

There is a field called class that now a days just basically isn't used and can safely be ignored. Then there's the type of data, is an address, is it an IPv4 address? Is it an IPv6 address? Is it the name of a mail server? Is it any one of the dozens of other things? Then what we call the R Data, that's the data itself. This is again, a lot of words on this slide, potentially confusing. The important point here is that a resource record, fundamentally it has a domain name, it has a type of data and it has whatever the data is.

I'm going to show you resource records in the standard text base format. The DNS standard defines a way to write down resource records so that you can put them in a zone file and every name server can understand this standard format. Let me give you some examples coming up of what resource records actually look like.

This is the most common list of resource records and we're going to talk about most of these here. By far the most common are the A and what we call the Quad A, AAAA, which hold IPv4 and IPv6 addresses respectively. There are a few others here that are very common. Then we sort of fall off a cliff in terms of popularity, there are many more but they're not that used. The last time I actually looked and wrote this number down on a slide, there were 84 types and this number changes very slowly.

It takes an action in the IETF, the Internet Engineering Task Force, it takes and IETF document to actually assign a new DNS resource record

and that requires time and effort to write that document and convince people and get it through the process. I haven't looked within super recently but there are on the order of 84 types.

Then if you know what the IANA does, the Internet Assign Numbers Authority, it has information protocol parameters, you may have heard about the IANA protocol parameters registries, that there are thousands of these things and one of them is the DNS resource record type registry and there's the URL for it. You can click on that link and look at all of the assigned types. The last time I clicked on it and took a screenshot, there's what it looks like. You could scroll down and see all the types that are valid in DNS.

Let's go through a few of these types and describe them. As I've said multiple times now, the most common type is to map domain names to IP addresses and there are two resource records that do that. The A record stores an IPv4 address and then the Quad A record stores and IPv6. This is literally what those records would like in a zone file, on a disc and that file would be loaded by a name server and then resolvers could ask DNS queries and get this information back.

We have the domain name, in this case that first record, what that says is that the domain name example.com has an IPv4 address of 192.0.2.7 and the second record says the example.com domain name also has and IPv6 address of 2001 etc. When it comes down to it, that's it, this is what makes the internet work right it. Lots of these records in DNS, mapping names to IP addresses.

Now, when it comes to types, I have an analogy here that will hopefully make this clearer. Some of the types on that page that I showed you, some of the types of data in DNS are only there to make DNS itself work, nothing outside DNS cares about those types. The whole reason we have DNS in the first place, A and Quad A and others.

I like to use this analogy of warehouse. If you rent a warehouse and you've got stuff you want to put in the warehouse, you don't just back your truck up and just throw your stuff in there, you first have to set up shelves, you have to get the warehouse ready. Then, once you have the shelves in place, then you can put your goods onto the shelves. Similarly, in my warehouse analogy, what we're going to see, we're going to see record types called NS and SOA, those are the shelves.

Those are types of data in DNS that only DNS cares about, they have to be there for it to work. Everything else, like A and Quad A, that's the stuff we care about, those are the goods on the shelves, that's the reason that you have the shelves in the first place. We just want to keep that in mind, that DNS sort of mixes types of data together, it mixes the stuff to make DNS work with the stuff that we want out of DNS.

The very next record I want to talk about, is called a name server record and this one of those pieces of shelving and this is how within DNS, you describe and document what are the authoritative servers, the authoritative name servers for a zone. In this case, these NS records are telling us that the zone example.com has two authoritative name servers and you can see the names of the authoritative servers.

What's kind of interesting here, is that note they're not IP addresses, we're not saying example.com has authoritative servers with this IP address, it's giving us the name of the name server, that means were also going to now have to look up the name NS1 example.com and NS2 example.com to get the corresponding IP 4 and IPV 5 addresses so that we can actually contact them. That's something to note here.

Then NS records right away get complicated because they appear in two places and let me explain what that means. I had arrows on that previous name space slide and I talked about delegation and I said that the parent zone delegates to a child zone and that there's this delegation information in the parent that describes the child zone, well that delegation information is these NS records.

Let's zero in on dot com, just to use an example. Those are the actual dot com NS records, 13 of them and if we sort of say those in plain English, what that text is saying on the upper right is that the dot com zone has 13 authoritative servers and they are named A.GTLD-servers.net through M.GTLD-servers.net. Those NS records, those 13 pieces of data are actually in two places. They are in the root zone where they actually delegate the com zone.

The presence of these records in the root zone, is what tells the world, "Hey, there's a dot com zone and if you want to talk to it, these are the 13 name servers, authoritative for dot com that you could send a query to." Then those same NS records also appear in the com zone itself because they're definitively owned by dot com, they're part of the dot com zone's authoritative data.

If we had this to do all over again, if we could send a time machine back to the DNS designers, I think one thing that we would say is, don't put the NS records in two places, make a different type. We could imagine that we'd have one type in the parent to do the delegation and then another type in the child to actually have the list of what the authoritative servers are. But we don't have a time machine and it is what it is. Just bare in mind that NS records appear in two places.

When a new TLD is created, where the rubber meets the road, where the TLD actually springs into existence, where the rest of the world is actually going to know about it, is when the root zone is updated with the set of NS records for that then TLD. As soon as those go in the root, then it becomes real, then somebody can look in the root zone, as we're going to describe how you look up data, they can look in the root zone and they can say, "Oh, there's a TLD and I know who the name servers are and I know who to go contact to send queries to."

In the case of a TLD for example, if it's not delegated in the root, it doesn't exist. If you take a second level name, a name in dot org, PIR.org, that won't exist unless n.org, they're NS records for PIR.org and that tells the world, "Okay, if you're looking for PIR.org, here are the authoritative servers to go ask for information."

This is a little more detail, more than we need to get into. This just shows that delegation information doesn't include only the NS record. It also has to include some times the corresponding IP addresses, the A or Quad A records for those name servers. I don't want to belabor that point but one thing I do want to show here, that I think illustrates an

important concept is that, you can think of these resource records that are associated with a domain. You can think of them as sort of hanging off the domain name.

Here I'm showing -- look at the example.com node, the node that represents example.com in the middle of the screen there with the red arrow pointing to it. That domain name, example.com, that node in the name space has some pieces of data associated with it. Here we see, in the square box, that those pieces of data are name server records or NS records. So conceptionally, you can think of every node in the name space is kind of like a coat rack with hooks on the wall and you can hang different pieces of data on that coatrack.

In this case, for example.com, you have NS records hanging off it, associated with example.com. If we look down of the lower right, you see NS1.example.com and that node in the name space has an address associated with it and you can see that address record at the bottom of the rectangular box. We have those address records each hanging off of NS1.example.com and at NS2.example.com. If you think visually and that helps, the idea that everyone of these nodes in the name space, you hang pieces of data off it and there can be nodes that don't have any data associated with them at all or you can have nodes that have a bunch of data.

Yet another piece of DNS shelving, is the SOA records. This is another record that only DNS itself cares about. As a general rule, there's no things outside of DNS that care about this. Every zone has one and only one SOA record and it's at the top of the zone. If I go back a slide, I

don't show it here but example.com, that node in addition to those NS records that we show, because it's the top of a zone, if you see where the zone boundary goes, example.com was delegated from dot com and it's the top of a zone, it's the name of a zone and therefore an addition to those NS records it's going to have an SOA record.

The one and only SOA record has information that applies to the entire zone. Most of it has to do with zone transfers and that's that replication protocol that I mentioned a few slides ago, but let's quickly step through them because you might see an SOA record someday.

The first field there, is the name of the primary name server, it's just documentation, it doesn't have to be but it's the name of the primary name server. The second field is actually a domain name but you need to read it as an email address, you need to turn the first dot into an at sign. That's the contact, the administrative contact for the zone. DNS has a place for you to put who the owner of a zone is, who you should contact if there's a problem with the zone.

Nowadays, if you want to know exactly whom to spam for the zone, you can look in the SOA record. That was a joke. The best jokes are the ones you explain. In this case, we would read that as, hostmaster@example.com and that is the contact email for if you wanted to reach the owner of example.com.

The next field is what we call we a serial number and you can maybe think of this, it's easier to understand as a like a version number. Every time this zone changes, that number has to go up and that's how the secondary authoritative servers know that the zone is changed on the

primary and that they need to get a new copy of it with the zone transfer. When a change made to example.com, that serial number has to increment and then the secondaries will know, the zone has changed, I should get a copy.

The other fields, I don't want to go into, I don't want to go into great detail but those are all time values in seconds and they relate to how zone transfers work and some other things that we don't need to spend a lot of time on. That's the SOA record, one of those per zone.

Now, I said early on that one of the goals for DNS was to fix HOST.TXT, to fix all the scaling problems. The other goal was to simplify email routing. Let's talk now about the tight connection between email and DNS. The issue that this is trying to solve is, if you have an email address like let's say user@example.com, where does the mail go?

In the early days of the internet, before there were domain names, before DNS, in the HOST.TXT days, email addresses were still user@hostname ad the host name was a name of a machine and what this meant was, your email address corresponded to the physical machine where your email was sent to. If computers fill rooms and you have one computer in your department at the university and everybody's email should go to that one place, this isn't a problem but what if you have more than one mail server because you need to have more space or what if you change departments or change jobs completely?

We have a situation here where before DNS, in the HOST.TXT days, email addresses were strictly bound to a physical machine, the name

of the machine was that right side of the email address and that's a problem. DNS provides a way to separate that, to decouple the right side of an email address with actually where the mail should go. We do that with what's called an MX or a mail exchange record. It decouples the physical mail server portion of the email, decouples the physical mail server name from the email address. In today's world, user@example.com, example.com is not necessarily and is most certainly not the name of an actually physical machine, this is where MX records come in.

MS records basically say, for this name if your sending email to this name, then here's the name of the mail server or severs you send that mail to. MX records, they have two fields on the right side, they have the name of the mail server and then they have what's called a preference field and counterintuitively with preference lower is more preferable.

What these MX records say is, if you've got a piece of mail for any user at example.com, now you're a mail server, a mail sever has this piece of mail to deliver, in order to know where to send it, it looks up the MS records for example.com and it gets this list back. What this list says is, okay, the most preferable place, because it's the lowest number, 10 is lower than 20, is mail.example.com so I should try to send this piece of mail to user@example.com, I should connect to mail.example.com, I need to look up the name mail.example.com and find the A or Quad A records to know the IPv4 or 6 address for this mail sever, I should connect to the mail server using the SMTP mail transfer protocol and then I should send that piece of mail.

But if for whatever reason mail.example.com is not working, it doesn't answer me, it times out, whatever, then I fall back and I try the next most preferable one, which in this case is mail-backup.example.com. You can see here, now that we have MX records, we can send mail for example.com anywhere, like to a third-party mail provider for example. Or you want to switch providers, you want to switch your mail from going to one third party provider to another third-party provider, you simply change the MX records and that redirects where the mail goes.

Let's talk about reverse mapping. It's too early in the morning for reverse mapping but we have the slides so we have to talk about it. Who made these slides anyway? Actually, I made the slides. What you're probably most familiar with, you may not even realize that reverse mapping is a thing. Forward mapping is what we've been talking about for the last hour, that's take a domain name and getting some piece of data, very often an IP address.

But sometimes you want to do the reverse of that. Sometimes you have an IP address and you want to know what's the name of this thing? Maybe it's a router and you want to know the name of the router because maybe you're familiar with the trace route command for network troubleshooting that shows you which IP address your packet is going through. Well, unless you're a real nerd, you don't know the IP address is, you don't know what they correspond to, especially outside of you network, so wouldn't it be nice if you could see the names of the routers that your packet is traversing. That's one example of where reversing mapping could come in handy.

Another is, imagine that you have a server of some kind, let's say its webserver and web browsers are connecting to your webserver and you're logging the IP addresses of all of the web browsers that are connecting. You have all the IP addresses of all of the different browsers that have connected to you. Wouldn't it be nice instead of the IP address to also know the name of the machine that was running that web browser, even if it's just some name that only has the name of the ISP in it, which is often the case.

But the point is, that's another example for reverse mapping and arguably those are the only two examples for reverse mapping. I'm showing my bias here, I don't think reverse mapping is nearly as important as forward mapping. If you can't map names to IP addresses, you've come to a screeching halt, nothing works on the internet. Mapping IP to name is considerably less important, things will still break. There are certain things that require it but it tends to be more of a diagnostic advantage or an optimization then it is something that's actually mission critical. So there, I've said it.

Reverse mapping, how does it work? Imagine back to the HOST.TXT days, you have this file and it has names and IP addresses, if you want to do forward mapping, if you want to know, I have a name, I want to know its IP address, well you just do a search through that file until you find the name and there's the IP address. Now, what if you have an IP address and you want to find the name? Well, you just do a search through the file until you find the IP address and then there's the name. Easy. The same mechanism works for both forward and reverse mapping.

Now we are in the brave new world of DNS. There's the name space. I haven't, I'm going to talk about it shortly, how you look stuff up in DNS but pretty clearly I'm not spoiling the story to say, if you start at the root, you can work your way down and find any domain name, you just follow the delegation pointers. There, now you know how DNS resolution works. You start at the root and work your way down.

Here you can see pretty clearly, let's say I want to look up www.example.com and if I start at the root, I just work way my down and then I get to www.example.com and whatever resource records are there that I care about. But what if, take the IP address in the lower right, what if I want to know the name of the corresponds to the IP address 192.0.2.1? Where do I look that up? The name space as I'm showing you, is optimized, it's structured not just optimized, it's structured for mapping domain names to other things and 192.0.2.1 is an IP address, how do I look that up in this?

Well the answer is, you have to turn the IP address into a domain name and then you look up that domain name and they'll be information there that corresponds to that IP address. Basically, it's this gigantic, extra part of the name space that supports this. This record type is called the PTR record and all of the IPv4 address space can be mapped under a domain name called in-addr.arpa. INADD is short for Internet Address, and for complicated historical reasons it was put under .ARPA so now we're stuck with .ARPA for forever. At the time, people didn't think it should be made a TLD, there shouldn't be an INNADDER TLD, so it went under .ARPA.

Every possibly IPv4 address is underneath in-addr.arpa, in other words, if you tell me an IP address, I can tell you a domain name under in-addr.arpa that corresponds to that IP address. Here's an example, look at the IP address on the lower right, 192.0.2.7, the in-addr.arpa domain name that corresponds to that is, you flip the IP address around, 7.2.0.192, that domain name corresponds to that IP address and then the PTR records is what tells the name that corresponds to that IP address.

Now, this doesn't work by magic, none of this is automatic, it has to be set up. The RIR's administer the in-addr.arpa name space and there's a corresponding domain name for IPv6 addresses that it's even more terrible looking than the IPv4 one. They administer those domains and there's delegation underneath those, just like there is in the forward part of the tree. If you have IP address space from an RIR or an ISP, there is a DNS delegation path that delegates the corresponding in-addr.arpa or IPv6 ARPA zone to you and it's up to you to put in the PTR records for how you use that address space. In other words, only if the owner of the address space goes to trouble to make sure that what we call the reverse mapping zone is set up correctly, only then will this reverse mapping work.

Here's showing this in the name space. I'm showing you the whole in-addr.arpa reverse map for IPv4 and I can't show IPv6 on one slide because it's too big because there's one level for each hexadecimal digit in IPv4 address, so that means it goes 32 levels deep. Underneath in-addr.arpa there is a possible domain name -- there is a domain name for every possible IP address.

There's NS records and delegation happening in this portion of the name space, just like the rest of it and there are PTR records down at the bottom. This is a thing, so I need to tell you about it but I think it's important that people be aware of it but it's not something I think you have to be prepared to take a test on or anything like that.

Now, there are many other record types, remember I said 84 as of a couple years ago anyway. I just looked through they type list and just pulled, not a random, but I pulled some of the more interesting ones out to just highlight them. Now, it's really important to note that a lot of these are -- some records types now are obsolete, we've actually decided, you know what, that was a bad idea, that didn't make any sense. Or it made sense 30 years ago and now we don't need anymore, so let's just deprecate it.

Then there are other records that people say, "Oh, this is a brilliant idea, the world must have my brilliant idea, life will not be complete until there is an LSC record in DNS, to put the geographic location of a domain name." They're not really used but they're, there, they're on the books. The point is, DNS is the biggest database in the world, it's arguably the biggest distributed database success story there is. It's completely reasonable to look at that and go, "I have some idea for something I could put in there." And there are some ideas that are really interesting.

One of those I think is the TLSA record there on the left and there is a protocol called DANE, which to summarize it super quickly, DANE let's you put a digital certificate, associate it with a domain name and

because domain names can be authenticated, we're not going to talk about that at all today, it can be cryptographically authenticated with a protocol called DNSSEC, you can imagine that you could trust a certificate coming out of DNS because it was authenticate with DNS rather than being authenticated by a certificate authority.

Maybe you do things like you say, this certificate should only be good if it's authenticated by a particular certificate authority. I'm getting way too far into the weeds but the point is, that's an interesting application, I think there's a lot of potential for, to basically put certificates or certificate authorities into DNS.

If we put all these different record types together, this is what a zone file might look like. We could imagine a zone called example.com and literally this would be the text file on the disc of the primary name server, this would be all the DNS information in that zone file for example.com. Every zone is always going to have an SOA record. Every zone is going to have some number of NS records and the NS records say who the authoritative servers for the zone are.

In this case, just by looking at the authoritative servers that the names of the NS records on the right side, we can infer something about example.com, you can see that two of them are named example.com, we could guess that those are name servers that are maybe run by example.com itself, we could imagine example.com own IT department maybe runs those.

Then, the next four, you can see the name Dyn in there, those are the names of name servers, authoritative servers then run by Dyn as part

of their manage DNS server. They would happily run authoritative servers for your domain. We can assume that example.com must also be a customer of Dyn, so they run some of the name servers themselves, they have Dyn running some and then you can see Verisign, that name buried in that, in those last three NS records, we can assume they must also be a customer of Verisign's managed DNS servers.

This is pretty common to have multiple -- first off, it's common to not run your own authoritative servers. It's common to completely outsource it to somebody else. In the early days of DNS, that wasn't a thing at all because there was nobody to do that. If you got a domain name, you ran your own authoritative servers. Maybe you ran one at your college and you talked to your buddy at the college across the country and you said, "Hey, can you be a secondary for my zone and I'll be a secondary for your zone?"

That was pretty common. Nowadays, hardly anybody runs their own name servers anymore, I mean I do but look. Most people outsource it to somebody else and you probably want to outsource it to two different parties. If anybody remembers the very, very, bad day that Dyn had in September or October of 2016, I had already moved on but they came under a denial service attack and many of their systems went down and anybody who used only Dyn as their authoritative DNS provider had a very bad day as well. But people who used two providers, were for the most part okay because maybe half of their name servers weren't responding but the other half were.

If we keep going down out list, we can see that there's an address record and Quad A record, V4 and V6 address associated with example.com, what is that the IP address of, the 192.0.2, if there's an IP address associated with a name of the domain, what is it almost certainly the IP address for? Anyone? The webserver because we've all be trained to type in the domain name and expect to get the website. 192.0.2.7, we can reasonably guess as well as 2001 etc. those are the V4 and V6 addresses of the webserver for example.com we don't know example.com has a website but almost everything has website, so that's a reasonable assumption.

Then we see the MX records, that tells us for any email addressed to anybody at example.com, it tells us where that email should go, the names of the mail servers. It looks like example.com runs its own mail servers, which is a thing but often you have third party mail providers as well, which is much, much more common.

Then we have we have www.example.com that's a record type that I didn't describe called C name but that's basically an alias and what that says is, if you look up www.example.com it's really example.com and you start your lookup all over again and you should look up example.com. If you were looking for IP addresses at www.example.com you should really look for those IP addresses at example.com. In other words, this is our proof that the example.com address is really the webserver because we can see if we look up www.example.com where by convention that's where the webserver is by longstanding convention, we can see it's really just used the address at example.com.

Then we have some IP addresses for the name servers that example.com runs, its own authoritative servers. This is very common zone file, in that if you think of -- if you have a domain name on the internet, what are you probably going to use it for? Web, email and this is a small zone file that has enough information to support, it tells you were the website is and it tells you how to deal with email addressed to user at this domain. The vast majority of zones on the internet are about this big, with this kind of stuff in it, this is very representative.

The big finish is putting all this together and describing how the resolution process works. The resolution process is simply, how do you look stuff up? We've talked about all the information that's in DNS but we actually want to use it and to use it we've got to look it up and how do we do that?

You remember my picture slide early on, in the lower left I had a stub resolver, part of a device, an end user device, in the middle I had a recursive resolver that does all the hard work and, on the right, I had authoritative servers. All those components together, they cooperate to look up data in the name space. Let me give some examples of how this works. I'm going to skip over some of the words here because it's easier to just give examples in pictures, which is what's coming up.

I already let the cat out of the bag. I said that resolution starts at the root zone. That if you don't have any cached information, which we'll see helps, you have to start at the root and you work your way down. There are a set of severs that are authoritative for the root zone called

the root name servers or the root severs. If you're going to start at the root with resolution, you have to know how to contact one of these root name servers, you have to know what their names and IP addresses are. How do you find them?

Well, the answer is, you have to be told. A recursive resolver has to have the names and IP addresses of the root name servers configured. You can't come up and discover them. When you open your laptop on this network, your laptop basically says, "Help, I want to join the network."

The network has a DHTP server and that protocol says, "It's all okay. Here's your IP address, here is the IP address of the name server you should use, here's all kinds of information that you need to be on this network." You can bring your laptop in from anywhere, start it up and it will happily join the network and get it everything it needs, the network will tell it. DNS recursive resolution does not work that way. Recursive resolver cannot start up and say, "Someone tell me the root name servers."

They have to be configured. This means that the names and IP addresses of the root name servers are configured in millions of places because there are millions of recursive resolvers on the internet. That information in what we call the root hints file and for very complicated historical reasons, it has that odd-looking domain name, it's the only thing left but that is the definitive location if you want the hints file, which looks like this. This is an up to date hints file. It has the names and IP addresses of root name servers.

Now, at the top are 13 NS records and the dot on the left is how we refer to the root node, remember. What this says is, the root zone has 13 authoritative servers, named arootservers.net through mrootservers.net, now these are just names of name servers, the reality is considerably more complicated and I'll get to that in a minute. Each of these names, you can then see if you look down, you can see that each name has both an IPv4 and IPv6 address. There are 13 root name server names, 13 identities and each identity have an IPv4 and IPv6 address. This is enough if you are a recursive resolver to get going.

Let me take a pause from resolution and let me talk about the root zone itself, which is very complicated for historical reasons. Two organizations corporate to administer the contents of the root zone. One is ICANN, which has a role and strictly speaking, its ICANN's subsidiary PTI is the IANA functions operator, that's the role that is -- I'll talk more about what the IANA functions do.

Then we have Verisign, they have a role called the Root Zone Maintainer. Then 12 organizations operate the 13 authoritative root server identities. These are the root server identities and the operators, there is A through M. There are 13 identifies but 12 operators because Verisign runs two, they run A and J. And that is for, everyone say it with me, complicated historical reasons.

These are the 12 organizations, 13 identities. This list has been relatively consistent, I don't believe it has changed since 1997 with exception of C, was run by PSI Net that went out of business and its

assets were bought by another ISP called Cogent but other than that, this list has been stable since 1997. The organizations running root servers have been doing it for a long time and they date back therefore to the early days of DNS because many of them go back even much earlier than that.

In the early days of DNS when it was just getting going, a guy named Jon Postel, one of the early internet pioneers who sadly died too early in 1998, when DNS was just getting going, for complicated historical reasons, he was the one who went around and said, "We have to -- these root servers, they're going to be really important to DNS, so I've got to pick people whom I know, who I know have an appropriate network and the appropriate level of clue to do it."

He was responsible for choosing the early root server operators and then it gets complicated. Fast forwarded to today, we have this list. It's an interesting list to say the least of organizations in that they have nothing in common with each other accept that they run a root name server. We have commercial organizations, nonprofit, educational, US government, little of everything.

If you want to know more, there is a How it Works Session just about that and that is later today at 3:15 in the main room, 517D. If this intrigues and you say, "I want to know more about those complicated, historical reasons." Then you could come to that tutorial and know more. That's all I have time to say about this now.

Now, in reality, there are not 13 physical root servers corresponding to the 13 identities. Way, way back when, that was sort of the case but

using a technic called IP Anycast, there are not 13 locations where the root servers, there are nearly 1000 maybe over 1000 at this point. IP Anycast allows you to have a root server or any server for that matter with a given IP address in multiple locations on the internet.

The same IP address can exist in multiple geographic locations and when you have a query you want to send to that IP address, the underlying internet routing infrastructure will decide which is the best one based on various complicated definitions at best, that your query should go to. That's all I have time for to say about IP Anycast. But there are more authoritative servers for the root zone then there are for any other zone on the internet, the coverage is amazing. If you zoomed in on this map, you would see where all the instances are, we call them instances, each site where there is a root server is called an instance and there are nearly a thousand instances and each has one of those 13 identities.

When did anycast start? The early 2000's. M Root was the first one to do it. That's probably something they talk about later on. Anycast has been use for coming up on 20 years. The question for the record, when did IP Anycast start?

I have one more slide before I can talk about resolution. This is really an injustice almost to have this slide in because it is at such a high level and there is so much of the process that is not covered, you really should come at 3:15 to hear this in more detail.

At a super high level, here is how the root zone itself gets changed. Because the root zone has information about top level domains, it's

administrators of top-level domains who care about what's in the root zone and they're the ones who have changes. When they want to make a change to the root zoon, they submit a change to the IANA functions operator, which is ICANN subsidiary PTI, which has a database of root zone stuff and IANA does various checks and then they communicate that change to the root zone maintainer, that's Verisign.

Verisign has its own corresponding database of root zone stuff, it generates a root zone file, I don't show it on here, it's cryptographically signed with DNSSEC and then it's made available to all the root server identities and then the root server identities A through M, they pull down that root zone. Twice a day is the standard cadence for a new root zone. In some cases, there aren't any changes, the only thing that happens is the serial number incremented but twice a day, pretty reliably Verisign releases a new root zone. Hopefully this just wets your appetite for 3:15 today in the main room.

Here is the resolution process. Here's how this works. We have the stub resolver, somebody's phone with a web browser running and in their phone they type www.example.com That web browser program calls the stub resolver which is a code, it's a function, probably in some library on the operating system and it says, "I want you to look up the IPv4 address for www.example.com" The stub resolver is configured to know what a recursive resolver IP is, it almost certainly got that when it came up on whatever network it's on.

The network told it, “Hey, I have a recursive resolver, here’s its IP address” If you’re going to run a network, you have to provide a recursive server or point somebody to somebody else recursive server because devices on your network need that. In this case, the stub resolver is configured to use a recursive resolver with the IP address 4.2.2.2 and so the recursive resolver send out this query and it says, now I have it in English here but what it’s really asking for is, “Can you give me the address records for www.example.com?” The recursive resolver has to look that up.

To make this example more interesting we’re going to assume that the recursive resolver was just turned on, so it has nothing in its cache, it’s starting from scratch. Remember, the one thing it has to have, is the names and IP addresses of the root servers. In the absence of any other information, the recursive resolver says, “Alright, I have no choice but I have to ask a root server about his query.” It picks one of the root servers, in this case it picks L, happens to be the one run by ICANN.

It’s up to the recursive resolver to decide when it has multiple authoritative servers for a zone to chose from, it’s up to it to decide which one, there are different ways to do that beyond what we have time to talk about. In this case let’s just say it picked L. It sends it, the same query, notice that it says, “What is the IP address for www.example.com” And in this case, the root server, well it can’t answer that, it doesn’t know that IP address, it doesn’t know anything about example.com but it does know about com because remember,

the root zone delegates to dot com, so it has a list of NS records for dot com.

It says here are the name server, here are the NS records for dot com and this is what's called a referral. An authoritative server is referring a recursive resolver to another authoritative resolver, in this case a set of them. The recursive resolver than picks one of those dot com servers, let's say it picks C.gtld.servers.net, this is potentially confusing, Verisign has named the servers for dot com similarly to how the root servers are named but the root servers are root-servers-net, the com servers are gtld-server.net.

In this case, recursive resolver sends that query to a dot com server, says I'm looking for the address record for www.example.com so the dot com server does not that but it does know about example.com because dot com delegates example.com to a set of servers, so it says here are the NS records for example.com

Finally, the recursive resolver can ask one of the authoritative servers for example.com, the same query for a third time and this time the authoritative server can say, I actually know that, I am authoritative for example.com zone, I can definitively tell you the IP address for the A record for www.example.com and the recursive resolver than passes that back to the stub resolver and then the stub resolver passes that back to the application and the application now knows the IP address of the webserver for www.example.com and it can connect and away we go.

Now, I've talked multiple times about caching. After that previous query that recursive resolver knows more stuff, it started out only knowing the root name servers but not it knows the names and IP addresses of the dot com servers, the names and IP addresses of the example.com servers and it knows the IP addresses for www.example.com.

Now let's say that we have another query, let's say that somebody is going back to the 1980's and they're going to FTP.example.com for whatever reason and they want to look that up. They pass that query to the stub resolver, the stud resolver ask the recursive resolver and the recursive resolver than does not have to go to root, it does not have to go to dot com it can go straight to an example.com name server because it had that list of example.com name servers cached and it can say what's the IP address, it can get it back and it can hand it off to the stub resolver which can hand it back to the application.

One thing I haven't pointed out here but should be obvious is that you can see the stub resolver really needs the recursive resolver. The stub resolver is by design very, very simple, all it knows how to do is take that applications request for DNS information, turn it into a DNS query, send it to a recursive resolver and wait. The recursive resolver has to either give it the answer that it asked for or it has to give it an error saying I can't do it or the information doesn't exist, something like that.

A recursive resolver cannot give back a referral because stub resolvers are too dumb to follow referral, they're just not designed that way. All

the smarts are in the recursive resolver. In fact, there are two different kinds of queries, the stub resolver sends a query with a flag called recursion desire, that basically says, "Help, I'm a dumb stub resolver, I need you to do all the work for me." And the recursive resolvers send queries without that bit set and that means to an authoritative server, I know what I'm doing, you can send me back a referral and I'll know how to process it.

With that, I have run almost up to our time. I know it's a lot of information. I know that I talk fast and I know it's early on a Sunday morning, but hopefully this was useful and I would be happy now in the remaining time to take any questions.

STEVE CONTE: I've got about five minutes for questions if anyone has anything. While we're waiting, Matt, on page 37, as you were looking at the zone file, I have to applaud you, I think you're the very first person I've ever seen who has embedded his resume into a zone file.

MATT LARSON: Oh, okay.

STEVE CONTE: Matt used to be with Verisign and then he moved on to Dyn, and now he's at ICANN.

MATT LARSON: You've given me an idea. I'm switching this to ICANN.Org.

STEVE CONTE: We've got a question in the back here.

UNKNOWN SPEAKER: What happens if the priorities are the same on the MX records?

MATT LARSON: That's a good question. In that case the mail server just choses randomly.

UNKNOWN SPEAKER: Meaning if somebody sends an email, it will later be delivered to any of the mail servers?

MATT LARSON: Correct and so it would be up to the person configuring those mail servers, to make sure that the mail servers know what to do with the incoming email. If you're going to say, "I've got multiple mail servers." You've got make sure they all have the same configuration and they all know what to do with an email that comes in.

UNKNOWN SPEAKER: Okay, maybe my question is out of the context but what if I want to have like a high availability name server? So if maybe this maybe MX, a second MX, it's not reachable or a primary MX is not reachable, I want

the same email to be delivered. So, a couple of the emails should be delivered to any of the mail server; what can be done on the DNS side to ensure that? I don't know if you get me.

MATT LARSON: So basically, you're saying you want to have let's say two mail servers and if one's down you want the other one to be able to accept the mail? You could have them at the same preference and then make sure that they have identical configuration and both can send the mail where it needs to go, in which case you can have either one go down and things will work.

Or you can configure one to be a backup, so that only when the first one is down, mail will que up on the back up and it will stay there until the main one comes back up and then the mail server will send it to the main one. But again, we're off into the area of mail server configuration and it would depend on the mail server to configure it, how to do that.

STEVE CONTE: Any final questions or thoughts?

MATT LARSON: One way on the other side.

STEVE CONTE: I'll come do you, don't worry, relax.

UNKNOWN SPEAKER: You talked about the Dyn attack that happened in 2016 and I was a slight victim of that, when the internet on the entire East Coast seaboard in the US, most of it went out just because they were relying on Dyn servers, DNS systems for that. I was wondering if you could briefly describe how they came under attack? What was the Achilles heel, why that happened in the first place and how do we prevent that from happening again?

MATT LARSON: I should say I was no long CTO of Dyn at that point, I had moved on. I only know what's in the public and I guess I would encourage you to read back. Or actually, I'm going to pitch my own podcast, this is so shameless and self-serving but I'm going to do it anyway. So, my good friend and former colleague, Cricket Liu, and I have a podcast called The-Ask Mr. DNS Podcast, so ask-mrdns.com.

And we did an episode in late 2016 where we had Joe Ably and Kyle York from Dyn come on and talk about it, and they were able to give us the public -- what was public anyway about the attack. Basically, it was a typical denial of service attack and that it was a botnet, it was multiple machines flooding them with traffic, that's the really short answer. That's something in the internet should be scared of because it can happen to anybody. And on that affirming note…

STEVE CONTE: Yes, I want to thank Matt Larson, our VP of Research for Office of CTO for his time to give us a tutorial on DNS 101. Thank you, Matt. I invite you all to hang out. In about 15 minutes we'll have Francisco Arias from ICANN in our Global Domains Division talk and give an intro about RDAP, which is directory services like WHOIS and other types of information for that. Please hang out for 15 minutes and join us for that.

After that, as I mentioned, at 3:15 in the main room we have Root Server Operators giving a Root Server Tutorial, and then at 5pm, and I don't have the room in front of me, we have Aaron talking about Regional Internet Registries. I invite you stay and join us all day today, we've got really good sessions. Thank you.

[END OF TRANSCRIPTION]