BARCELONA – How It Works: DNS Fundamentals
Saturday, October 20, 2018 – 08:45 to 10:15 CEST
ICANN63 | Barcelona, Spain

UNIDENTIFIED FEMALE: Good morning, everyone. We'll get started in a few. We'll give it a few more minutes. I think it's a bit early for some people. Thanks.

Good morning, again, everyone. Welcome to the first day of our How It Works tutorials. We have two days of How It Works, four sessions each day. We are starting today with DNS Fundamentals and our presenter is Matt Larson, Vice President of Research from the Office of the CTO at ICANN. Welcome, again.

MATT LARSON: Good morning, everyone. Welcome. So, considering the number of us that we have in here, we have plenty of time for the material, so please feel free to wave your hand and ask a question. We have a roaming mic and there's a mic there, so we're good. Let me know if you have a question. You can actually probably just shout it out and I'll repeat it for the recording. Well, let's begin.

I'm going to assume everybody knows what an IP address is. These are easy for machines, but hard for people. There's examples. I'm sure everyone has seen an IPv4 address. You may not have seen an IPv6 address. They're not quite as common, but they're even harder to remember than IPv4 addresses. So, if you're going to try to access a

resource on the Internet, it simply doesn't work to try to remember the IP addresses. People deal in names.

So, this is an issue called name resolution that has to be taken care of for people to be able to use names on the Internet and machines to be able to turn those names into addresses, so that they can get the packets to the right destination.

So, in the early days of the Internet, this name resolution process was simpler and names themselves were simpler than what we have now. They were what we would now call a single label name. That is, there were no dots. There were no domain names yet and the reason is that domain names hadn't been invented in the early days of the Internet. These single label names could be up to 24 characters, and they were called host names.

So, as I said, the process of mapping names to IP addresses is called name resolution. This is something that any application on the Internet is going to have to do because users are going to type in or otherwise communicate in names, and the application needs IP address to send its packets to the right place.

So, this process of name resolution has changed over the years. In the early days of the Internet, it was done very simply using a process and a file called a host file. For historical reasons, I have it up here it that it was named HOSTS.TXT. This is the same function of the modern /etc/hosts file on Linux or Unix, if you're familiar with that. But, it was a slightly different format, but the idea was the same.

You had a file that had the name and IP address of literally every machine on the Internet. The Internet, of course, was much, much smaller. We're talking about the late '70s/early '80s timeframe here, or early '80s timeframe. So, it was possible to have every machine on the Internet's name and address located in one file. It was a big file, but it was possible.

Name resolution itself was really easy. When you wanted to look up a name, you just searched through the file until you found the name. Then, you looked over in the next column, and there was the IP address. So, it was very easy for applications to do.

Now, this file had to be centrally maintained, and there was an organization that had a U.S.-government contract to do some network maintenance/network operations tasks in the early Internet. One of the things they did was maintain this file.

Network administrators all over the Internet, when they had changes, when they would add a machine, or delete a machine's name or its IP address, would send an e-mail to the NIC (Network Information Center) and say, "Oh, hey, can you update the master host table with this change I'm giving you?"

This process worked reasonably well for a while. Ideally, everyone had the latest version of this HOSTS.TXT file. It was released once per week, and it was downloadable via the FTP protocol. So, whenever you thought your file might be out of date, you would FTP a new one. Some people just had it automatically schedule to download a new file every

week. You'd maybe download the file once for your local area and then distribute it among your local machines.

Did I do that? Sorry.

Okay. I will try not to touch my laptop. [I've never] used the clicker to advance.

All right. So, as you might guess, there were some obvious problems that happened with HOSTS.TXT grew, and, therefore, the file grew. One was naming contention. Remember, this was before domain names. So, all of the names in this file were up to 24 characters long. Everybody in the entire Internet had to share that name space, if you will, of 24 characters. That meant that you had to use more and more obscure, complicated names to be unique from all the other names.

On top of that, the file maintenance process was extremely low-tech. Nowadays, what we do is we have a database and you extract a text file from the database periodically. But, back then, there was no database behind this. It was literally an ASCII text file that someone at the NIC pulled up in a text editor and made changes to.

So, that meant that there was no good method to prevent duplicates, and sometimes duplicates did creep into the file because it was just a human editing a file.

Of course, synchronization was a problem. Nobody ever had the same version of the file, ever. There were always different versions floating around because it was impossible to stay up to date.

Finally, the traffic and load involved on the early Internet just to move this file around began to significant. It began to be a significant portion of the overall traffic on the Internet, just moving HOSTS.TXT. I am told – this is a little bit before my time – that, in the last days of HOSTS.TXT, the file was so big and the download was so slow that it took over a week to download it. So, by the time you actually finished the download, there was already a new one ready. So, it was physically impossible to stay up to date.

So, clearly, the idea of a centrally maintained file with every name and IP address of every machine on the Internet just did not scale.

So, in the early '80s, discussion started on a replacement. There were a couple of goals. One was addressing the scaling issues that I've just talked about. Another one that's important that gets overshadowed by the first goal is to simplify e-mail routing. Back in the days we're talking about, your e-mail address was still username@, but rather than domain name, it was usually named @hostname. So, your e-mail address was tied not to a domain name – because domain names don't exist yet – but tied to a physical machine.

This was an issue, because what if that machine got overloaded and the administrators needed to sort of rebalance and puts accounts on other machines or otherwise maybe retire that machine. They no longer wanted it to be the mail server. They wanted another machine to be the mail server. But, everyone's e-mail address was tied up with the physical name of the machine. So, a goal was to simplify e-mail routing

to make it possible to separate your e-mail address from the physical machine.

So, as we all know, the results of this discussion on a replacement was the domain name system. So, this is my one slide summary of DNS; DNS in a nutshell. Fundamentally, DNS is a distributed database. It's distributed all over the world, but everyone owns their own portion of the data. So, that's what I mean by: data is maintained locally. The people who own the data get to maintain it. But, this is available globally. Anybody all over the world, all over the Internet, can look up that data.

DNS follows the client-server model. Clients are called resolvers. The main thing to remember about resolvers is that resolvers send DNS queries. The server side is called name servers. The main thing to remember about those is that name servers answer queries.

Now, there's some important optimizations. DNS uses caching all over the place to improve performance. If we're talking about a distributed database that's over the entire world, the speed of light is only so fast. So, you can imagine doing a lookup, if you had to do multiple lookups all over the world – send queries, get responses – we're talking hundreds of milliseconds. Those eventually add up to seconds. They add up to a lot of time. Often, there's a real user waiting for a lookup to complete. So, DNS uses caching to improve performance. DNS clients can remember information they've looked up previously to speed up further lookups.

DNS also uses replication to provide redundancy and load distribution. What I mean by that is this data that a given organization will maintain locally – they don't have just one copy of the data. They replicate it and have multiple copies of the data. That provides redundancy in case one copy is unavailable. Also, if your organization is getting a lot of DNS queries, you have multiple places where that data is available to answer those queries.

So, what I think is helpful at this point is to give a high-level overview of all the components of the DNS ecosystem because we're going to be talking about these as we go on. I think it helps to get this 10, 000-ft. view early on to sort of set it in your mind.

So, let's start at the lower left. We have a device there that needs to use the Internet. Of course, it used to be that such devices filled rooms. Then they sat on desktops. Now we literally carry them around in our pockets or wear on our wrists.

So, any device that's going to use the Internet almost certainly is going to use domain names. Therefore, it's going to need a DNS client to turn those domain names into IP addresses. This client is what we call a stub resolver. Remember, resolvers are DNS clients. There are actually two kinds of resolvers. The first is this very, very simple one called a stub resolver.

A stub resolver is often, almost always, provided by the operating system. It's a system the operating system has to provide for applications. So, for an application there, for example – we have the Safari web browser icon. So, we can imagine a user typing a name into

Safari. So, Safari, via an API call, via a function call in the Safari program, calls the stub resolver and basically says, "Here's a name. I need the IP address."

So, the stub resolver's job is very simple. It's to accept that request from an application and then generate a DNS query and send that query to something called a recursive resolver, which is what we have in the center there. It's appropriate that that's in the top-center of this diagram because recursive resolvers are the work horse of DNS.

So, the stub resolver is the simple DNS client. The recursive resolver is the complicated DNS client.

So, it actually consists of two parts there, you can see. It has a name server component in that it's answering queries itself from the stub resolver. But, it also has a resolver component – a complicated resolver component – that knows how to navigate the entire DNS and track down the information that the stub resolver is looking for.

It does this by sending queries to what we call authoritative name servers. It might send a query to an authoritative server that might say, "Well, I can't answer your question, but I can get you closer to the authoritative server that can. Let me refer you to another authoritative server." The recursive resolver is smart enough to follow that referral, contact another authoritative server. Eventually, it contacts the right authoritative to answer the stub resolver's questions exactly. It gets that response, and then it relays it to the stub resolver.

So, as you can see, the stub resolver's job is pretty easy. It just throws a query over the wall to the recursive resolver, and it waits. The recursive resolver either has to provide the answer to the query or an error of same kind, saying, "The name doesn't exit. I couldn't do your query," or something along those lines.

Another important point about the recursive resolver – I know I've shown there that there's a cache, that it remembers all of the lookups that it has done so that it can speed up future lookups.

Caching actually happens at multiple places. I don't show it, but now modern stub resolvers also typically have a cache. And even applications have a cache. They remember what names they've looked up with the stub resolver.

So, you have DNS information getting caches multiple places, which makes changing information sometimes take a while because you have to make sure that it's going to time out of all those caches – the old information – before the new information is available.

But, that's a high-level view of DNS that hopefully will help as we go on here.

So, I've said that DNS is a distributed database. Well, what's the structure of that distributed database? Well, the structure is what we call an inverted tree. This is a computer science thing. I have an example of an inverted tree there. It's inverted because, in a normal tree, of course, you have the root at the bottom and the branches grow

upwards. But, a computer scientist tree is upside-down. A computer scientist has the root at the top, and then the branches go downward.

So, this inverted tree structure is something we call the name space. This is the structure of the DNS database. It's a massive, massive inverted tree.

So, if you compare this, you may be familiar with other kinds of database. Think of a relational database. The structure of that database is you have tables, and each table has rows, and, in a given row, there are multiple columns. So, that's one way to do it. That's one way to store information in a database.

But, the important point here – the reason I'm making this distinction – is that the DNS name space is not like a relational database. Fundamentally, there's this inverted tree down underneath everything, which has important implications for writing software and for understanding how DNS works.

So, in this inverted tree, we have nodes. Each one of those boxes is what we call a node in the tree. Every node has a label and has a name, except the root node. The root node has a null label. That is, its label is that it doesn't have a label. That's sort of mind-bending for this early in the morning on a Saturday – the idea that the label is no label. I'll let you think about that for a moment. That was a kind of a joke, but it's also a little early on a Saturday morning for a joke, and it wasn't a very good joke.

So, the root node has a null label. Sometimes you see it written as here with a dot, just to indicate that there's something. We're trying to show: how do you show that nothing is there?

So, if you look on the right of this slide, you'll see that we often talk about the relationship of these nodes to the root, where they are located physically in this tree. The root, of course, is at the top. Immediately below the root, those nodes are what we call top-level nodes. Below that is second-level, and so on down.

Sometimes you use parent-child terminology to describe nodes. For example, well, the node named Example is a child of the node named Com. Com is the parent of the node named Example. So, you use parent-child terminology to refer to nodes that are above and below one another in the name space tree.

So, legal characters for these labels are what we abbreviate as LDH: Letters, Digits, and the Hyphen. That's all U.S. ASCII. That's all you can use for label names. They have a maximum length of 63 characters. Case does not matter. Uppercase and lowercase are compared similarly.

Now, I know everyone is maybe familiar with the concept of internationalized domain names, where there are many different characters and domain names other than U.S. ASCII.

So, you might be asking, "Well, how is it that that's possible, yet you're telling me that the legal characters for a label are letters, digits, and a hyphen?" That's something called internationalized domain names. We

don't have time to go into it in any depth, but the short answer there is that internationalized characters are encoded into ASCII characters. From DNS's perspective, it all just looks like letters, digits, and hyphens. It's an application, like a web browser, that sees the encoded name and turns that into the appropriate internationalized characters.

An example of one of those names is in the upper-left. All internationalized domain names, all labels that are internationalized, start with xn--. That's completely random. There's no semantic significance to xn. It was just chosen at random. Then, the characters following are the internationalized names.

Actually, I confess that, even though I made this slide, I don't remember which internationalized name that is, which is one of the downsides of IDNs from an administrative like this, that it's just a string of characters without an application to turn that string back into internationalized characters. You don't even know what they are.

All right. So, every one of these nodes in the name space has a domain name. That's how we name the node and how we describe it and describe where it is. So, you see there's a node highlighted there at the bottom in blue, that www. Well, that node's domain name, to generate that – we start with that node and we write down its label. Then we write a dot, and then we write its parent's label, and we write a dot, and then its parent label, and so on, until we get to the very top of the name space, until we get to the root.

So, you can see that, if we do that, working our way up the domain name that's highlighted there, it's www.example.com. A fully qualified

domain name that is not relative to any other domain name. It's absolute. It can refer to one and only one name in the name space. A fully qualified domain name ends in a dot. That dot at the end is the separator between the top-level node – in this case, com – and the root's null label.

So, you imagine that, if you're working your way up, you write www.example.com-dot. And then the root is null label, but you can't write the null label. So, you end up with a name with a dot.

That dot at the end indicates that we are talking about one and only one domain name here. Domain names can be relative. You, for example, could talk about www.example relative to the com name. So, www.example relative to com. Www.example is not a fully qualified domain name.

If it helps – I don't have a slide for this – an analogy would be a file system, whether it be the Linux or Unix file system or the Windows file system. Underneath the file system, that data structure is also an inverted tree. The root directory – slash of backslash, depending on Linux or Windows respectively – that is the root directory. So, imagine a path name that starts with a slash, and then you have all the various names, directories, down to the file name. That's like a fully qualified domain name. An absolute path name starting with a slash is like a fully qualified domain name.

But, everybody is familiar with relative path names, like you were talking about only the name of a file. That file could be in any directory. That's a relative file name, as opposed to the absolute path name.

Just a couple more terms here, and then we'll move on to some other material. So, a domain is a term we need to understand. That is a node in this name space and everything below it, what we would call its descendants – its children and its children's children, and so on.

So, let's look at the node dot-com here and let's highlight it. Dot-com there is what we call the apex, or the top, of the dot-com domain. The dot-com domain includes dot-com and everything below it. How many dot-com names are there? At this point, 120+ million. So, I'm showing three out of 120 million. But, the dot-com domain is huge. It's dot-com and everything below it in this name space.

Now, we contrast that to this term, "zone." This is a pretty important term here, this concept as well. So, we talked about, just a moment ago, about the host file didn't scale. The idea that you had everything maintained centrally by one organization just did not scale, did not work.

So, a key design goal of DNS was to allow distributed administration, to allow everybody to maintain their own data directly themselves. So, this entire name space is divided up to allow distributed administration. So, different organizations have different parts of the name space, and they control them directly.

So, how do we divide up the name space? Well, those administrative divisions are called zones. Zones are created by a process called delegation. So, a zone higher up in the name space delegates to a child zone, and that child zone can delegate to another child zone. So, just as in real life, where you can't stop your children from having children

themselves, when you delegate to a child zone, that child zone in DNS can then also delegate further down the name space and delegate child zones.

So, here's an example. Here we have that same portion of the name space that I'm using for these examples, the same one I've been using. This is just the name space, just the inverted tree.

Let's draw some zone boundaries. Now, the thing about zone boundaries is that they reflect administration. They reflect how we've divided up the name space to administer it. We could divide it up differently depending on what the requirements were.

So, you can't go from this to this without having the additional information of knowing there those zone boundaries are. I happen to know, because I made this example, that these are where I'm placing the zone boundaries. And they do reflect how things are in real life. In real life, there is a root zone at the top of the name space that delegates – got some arrows here that represent delegation. The root zone delegates to various top-level zones. The top-level zones then keep delegating on downward, as I've shown here.

So, what's the relationship of name servers that answer DNS queries to zones? Well, this is where that term "authoritative" that I used on that overview slide comes in. We say a name server is authoritative for a zone if it has complete knowledge of the zone. So, you take zones and you put them on name servers, and then name servers know all about those zones and can answer queries about data in the zone.

So, the idea is that an authoritative name server can give you a definitive answer about information in the zone. It can say, "Here is the answer. I can tell you definitely because I'm authoritative for it," or it can say, "What you were looking for doesn't exist, and I could definitely tell you it doesn't exist because I know everything there is to know in the zone as the authoritative name server for that zone."

Now, for a given zone, you don't want just one authoritative server. Imagine what happens if somebody trips over the power cord and the server power goes off. Then your zone is off the air, and nobody can ask you queries. So, you want multiple authoritative servers for any given zone. This goes back to what I said a few slides ago, that DNS uses replication to achieve redundancy and spread the load.

So, by having multiple redundant authoritative servers for your zone, you have redundancy, and then, if you're getting a lot of queries, you have multiple authoritative servers to answer those queries, to spread the load out.

Now, if you're going to have multiple authoritative servers, you have to have a way to keep them all synchronized. You need the same information for your zone on all servers. But, fortunately, DNS makes this easy. There is a built-in zone replication protocol called the zone transfer. I won't read you all the words on this slide, but it describes how the concept works.

You have a given authoritative server that you designate the primary. That's where you make all the changes. Your other authoritative servers are configured to load from the primary. So, you make a change in the

primary, the primary tells what we call the secondaries that the zone has changed. The secondaries come in and they say, “Please give me a new copy of the zone,” via this process called the zone transfer, and then the secondaries get a copy of the zone. So, that keeps everyone in sync.

Now, it’s important to remember here that the information on all these authoritative servers is the same. Once the replication happens, it’s the same information. So, the primary isn’t any better than the secondaries. The only difference is that it’s where you make the changes. It’s where the changes start propagating. But, the information on all the servers is the same. So, there’s no degrees of authority. You can’t be more or less authoritative. You’re either authoritative or you’re not. And primary and secondary are equally authoritative.

So, we’ve talked about zones at a high level, but let’s look into a zone and talk about the kind of data that we can put into DNS and that a zone would hold. So, remember, every one of these nodes in our name space has a domain name. The idea here is that a domain name can have different kinds of data associated with it. Those data are what we call resource records. We sometimes abbreviate that as RR. There are different types of records for different kinds of data that we want to store in DNS.

I don’t know why we got some things blocking here.

So, a zone consists of multiple resource records. What’s blocked there is it says that all the resources for a zone are stored in what’s called a

zone file. Every zone has its own file. You never mix resource records from multiple zones in one file.

So, the point is, a zone is the sum of all its resource records, and they go in a file called a zone file. Then, that zone file goes to an authoritative server, which reads the zone file, and then it knows everything in the zone, so it can answer queries about that zone.

So, I want to go into just a little bit of detail of what makes up an individual resource record. So, we're talking now about a single piece of DNS data because, sometimes, you'll actually see these written down, and it helps to understand the syntax, at least, at a high level.

So, a resource level has five fields. It has the domain name that the resource record is associated with. Every resource record has what's called a TTL, or a Time To Live. That's how long that record can live in a cache. It doesn't apply to the authoritative server. The authoritative server knows about that information forever. But, when it gives it out to someone and a resolver and the resolver puts it in its cache, it has to obey this time-to-live value.

So, I want to skip the rest of that and show some examples that make it a little more straightforward. Here are some common resource records types. I think I have – yeah. These are, by far, the most common, this list of seven. There's a handful of resource records that make up the vast majority of all the information in DNS at this point. This is that short list. We're going to talk about these, each, in some level of detail, sometimes king of briefly.

As we'll see, the most common type of resource record are the address records to store v4 and v6 addresses, because that's the main purpose, at this point, of DNS: to turn domain names into IP addresses. So, it would make sense that most of what's in DNS would be records that store addresses of domain names. But, there's other information in there as well.

So, in fact, as of about a year ago – the last time I looked – there were 84 different types allocated. There is an IANA registry for these, if you're familiar with what IANA is: the protocol parameter registries. Here's the actual URL and what the webpage looks like. So, there are 84 different official types of data that you can put in DNS.

It's relatively easy to get a new type. As you can see from some of the numbers there, it's a 16-bit value. We can 65,000 different types, ultimately. So, we have a lot of room for expansion. There's a lot of additional types. As people think up new and crazy things that they want to put in DNS, there's plenty of type space available to create those records and do that.

But, let's talk about some of these types of resource records. So, as I said a moment ago, the most common use of DNS is mapping domain names to IP addresses. That's arguably the main reason this whole thing exists. So, the way we do that is with two different types of records: The A record, or the address record. That maps a domain name to an IPv4 address. And then there's what we call a Quad A record because there are four A's. That maps a domain name to an IPv6 address.

Here you have an example of what these records actually look like. If we were to look in a zone file, the plain text representation of those records would look exactly like I have on the slide here. So, that first record simply says that example.com has the IP address 192.02.7, and then the second record there says that example.com also has the IPv6 address, starting with 2001 there. So, that's actual DNS data shown on a slide.

Now, DNS is sort of both clever and confusing at the same time in that these types, this list of 84 types – there are types that are used by people outside DNS, like A and Quad A; so, consumers of DNS, if you will. But, DNS itself, to make DNS work, also uses certain types. They're for internal use only. They're when you open up the computer or whatever and it says, "For Internal Use Only." That's those types. But, these types are sort of mixed together.

So, I think it's important to remember that some types are used by people outside DNS, and some types are used by DNS itself to make DNS work. The analogy I use – think of a warehouse. If you've got stuff you care about and you want to put it in a warehouse, you don't just rent the warehouse, back your truck up, and just throw the stuff in. That doesn't work. You need shelves and scaffolding. You need the infrastructure in the warehouse first, and then you take the goods out of the truck and you put them on the shelves.

So, some of the types for DNS – examples would be NS and SOA, which we're going to talk about – are like the shelves. They're used only by DNS itself. Nothing outside DNS cares about those types. But, the stuff that things outside DNS care about would be examples like A and Quad

A, where we're actually saying, "This name has this IP address." That's something that people care about and that they look up in DNS.

So, let's look at one of these shelving types, if you will, these internal types. There's this NS record, the Name Server record. This is how you say that the authoritative name servers for a zone are. So, as an example here, these two records show that example.com, the zone, has two authoritative name servers named, ns1.example.com and ns2.example.com. So, note that the right-hand side is the name of the authoritative name server and not the IP address. It's always the name of the authoritative name server.

Now, right away, NS records get a little complicated because they actually appear in two places, as I'll show. NS records appear both in the zone itself and in that zone's parent.

So, here, for example, are the actual NS records for dot-com. So, dot-com has 13 authoritative server names. They're named, as you can see, A (through M).gtld-servers.net. So, those NS records appear in the dot.com zone, but they also appear in the root zone. The root zone is the parent of dot-com, and the root zone delegates to dot-com to make dot-com exist. The actual delegation in the root zone, the thing in the root zone, the information that makes dot-com exist, is this set of NS records.

Now, I think if we had this to do all over again, we would have used two different types. We would have used one type in the zone, and there would have been another special type for delegating. That's not how it was designed. It was designed to have the same type used in both

places. So, you have the NS records for dot-com in dot-com, and you have the NS records for dot-com in the root. It's when they appear in the root that [it] actually delegates dot-com into existence. So, there's just explaining that on the bottom-half of the slide. You can think of these NS records as sort of hanging off the dot-com node in the name space. They're associated with dot-com, but then again, they also appear in the parent zone.

Here's an example moving down one level in the name space tree. Let's look at example.com. Here I have some example NS records for example.com. Again, I want to point out that they appear both in the example.com zone, and they appear in the dot-com zone. It's the presence of these NS records in dot-com that actually delegate example.com, that bring it into existence.

So, dot-com, then, actually is 120 million sets of NS records like this, because that's what causes a domain name to existence: the delegation NS records in the zone's parent.

Now, you can see that I've also got a couple of address records shown there. So, sometimes you need to have the address records for name servers as part of the delegation. We call that glue. I'm not going to go into the details here, but delegation actually consists of the NS records and sometimes some address records for the names in those NS records.

Another one of these internal-use record types is what we call the Start of Authority. There's one and only one SOA record per zone. It goes at

the top of the zone. It has some values in it that mostly control zone transfers, that replication process, from the primary to the secondaries.

So, this is an example of an SOA record would look like. I'm not going to go through all the values in detail. An important thing to note, though, is that every zone has a serial number. That's simply used as part of that zone transfer synchronization. Every time a zone changes, its serial number has to go up. That's so the primaries can tell, "Do I have the current version of the zone? My serial number is the same as the primary's?" Or, if the primary's serial number is greater, then the secondary knows, "Oh, I'm out of date. I need to do a zone transfer and get the current version of the zone."

So, remember, I said on one of the first slides in the class that one of the other design goals of DNS, in addition to name resolution, to mapping names to IP addresses, was to simplify mail routing. This is the issue. I'll recap it here. In a modern e-mail address, like the one there – user@example.com – how do we know where the mail goes?

Well, in the old days, as I said, you would have looked up the address for the right side of the e-mail address. It wouldn't have been a domain name. Domain names didn't exist. It would have been a host name. You would have looked up that host name. There would have been an IP address. Then, you would have looked up the IP address for that host name. You would have connected to it via the SMTP mail protocol, and you would have delivered your message.

DNS lets us decouple the e-mail address from the physical machine name. We do that with this MX record, or this mail exchange record. It

looks like this. It lets you say, for a given domain name – say, example.com – where do I want the mail to go? What machines are the mail server for this domain?

There's also that other field, the number you see there – the 10 and the 20. You can have multiple MX records to specify multiple equivalent mail servers or backup mail servers. So, counterintuitively, lower is more preferable.

So, in this set of MX records, what it says is that, if you have a piece of mail for example.com, you should send it to mail.example.com first. But, if you can't, for whatever reason – let's say mail.example.com is not available; you try to reach it and it doesn't respond – well, then you send it to mail-backup.example.com, because that has a higher preference – well, a higher preference number, but it's actually less preferable.

If you had multiple mail servers that were all equivalent to say, "Spread the load of incoming mail," you could have multiple MX records, all with the same preference.

So, what happens is that, when mail server has a piece of mail that it needs to deliver, it looks up MX records because it doesn't know how to deliver the mail. If a mail server has a piece of mail addresses to matt@example.com, the first thing that it does is it looks up MX records for example.com to know: where should it send the mail? Then, when it sees this set of MX records, it knows, "Oh, okay. I should open a connection to mail.example.com and deliver this piece of mail addressed to example.com to that mail server."

Now I want to talk about something called reverse mapping. So, we usually think of, in a DNS context – I've said this multiple times – domain name to IP address. That's what most people care about. I have a domain name of a website, of a mail server, of a whatever. I have the domain name. I want to go there. So, I need to turn that into an IP address. That's DNS.

Sometimes you want to do the reverse. You have an IP address and you want to know what's the domain name associated with it. So, that's what we call reverse mapping. Name to IP is forward mapping. IP to name is reverse mapping.

Now, in the good old days with HOSTS.TXT, this was easy because you had the file with names and IP addresses in it. So, you want to look up a name? You just go through the file until you find the name. There's the IP address.

Let's say you have an IP address. You want to find the name. Well, you look through the file at the IP addresses until you find the IP address. And there's the name. Simple. It's the same process for forward mapping and reverse mapping with the host file.

DNS doesn't work that way. The DNS name space that we have – let me just back up a few slides to it; here we go – is optimized for searching on name. If I tell you to look up www.example.com, you can pretty clearly see that, if you start at the root, you can work your way down and find it. Indeed, that's exactly how name resolution works. I just haven't gotten there yet.

So, the DNS name space optimized for looking up names.

Now, if I say, "I have the IP address, 182.0.2.1. What's its domain name?" well, if you look at this, where do you start? You can't look up IP addresses in this data structure. It's not optimized for doing that.

So, what people said, though, is, when DNS was being designed, "We still want reverse mapping. It's useful for a few things." It tends to be useful really only in a network administration context. I think nowadays the canonical example is a utility called Traceroute that lets you show the hops that your packet is taking as it travels across the network.

Well, Traceroute will show you the IP addresses, but using reverse mapping, it will turn those IP addresses into the names, and it can show you the name of the routers that it's traversing. So, that's an example where reverse mapping helps you understand the network better.

Reverse mapping is also sometimes used if you're looking at logs on a server – let's say the logs on your web server – and you see connections from different places. Rather than the IP addresses, it's nice to see the names of the machines that connected to your web server.

So, reverse mapping is important. It's not nearly as important, in my opinion, as forward mapping. If forward mapping stopped, the Internet would screech to a halt. Society would collapse. Dogs and cats would live together. If reverse mapping stopped, it would just make a lot of network administrators upset, but the world would go on and keep spinning.

All right. So, how does reverse mapping happen? Well, the answer – remember, I said the name space is optimized for searching on names. So, if we're going to have a way that we can look up an IP address, we need to take the IP address and turn it into a domain name. So, if we turn the IP address into a domain name, now we have a domain name that we can look up, and we solve our problem.

So, in fact, that's exactly what happens. So, the IPv4 addresses are all mapped onto a domain name called in-addr.arpa. In-addr stands for Internet address, and arpa is a long story. But, in-addr.arpa is where the IPv4 address space goes. So, every single possible IPv4 address has a corresponding domain name in in-addr.arpa. At that in-addr.arpa domain name for that IP address, you put a PTR record – for pointer – and that points back, if you will, to the domain name.

So, here's an example in in-addr.arpa, 7.2.0.182.in-addr.arpa. That actually corresponds to the IP address, 192.0.2.7. You have to flip the octets of the IP address around. What that PTR records says is that the IP address, 192.0.2.7, has the name example.com. As you can see, it corresponds to that A record. You need both. If you want reverse mapping to work, you have to have PTR records.

I have an example of this. So, this is more of the name space. This might be buying a house. You have a door and you don't know what's behind that door. You finally open the door and there's something horrible back there. That's sort of like reverse mapping. We have this name space, and, "Oh, look. Over here in the corner is in-addr.arpa with all this stuff in it to make reverse mapping work."

So, with reverse mapping, the only reasons it works is that it's a convention that everyone follows. If you have IP address space assigned to you and you want those IP addresses to reverse-map, from the regional Internet registries you can get a delegation to the corresponding in-addr.arpa zone.

So, you would manage, for example, the zone 2.0.182.in-addr.arpa, and you would populate with the PTR records that corresponded to the names of the devices on your network. Then, any consumers of DNS, anyone who wants to do reverse mapping, if they have an IP address and they want to know, "Well, I wonder what the name is," they have to take the numbers in the IP address. They have to flip them around. They append in-addr.arpa. They look up PTR records like this, and then they get the name. So, everybody understands the convention, and it works.

This is for IPv4. IPv6 is even uglier. Those go under, as I had on the previous slide here, ip6.arpa. There is one level in ip6.arpa for each hexadecimal digit in an IPv6 address. There are 32 hexadecimal digits in an IPv6 address. So, there are therefore 32 levels underneath ip6.arpa instead of just four, as there are for IPv4.

So, it used to be that humans could maintain this. It was awkward and a pain in the neck, but you could do it. Nowadays, you have software called IP address management software that does all this for you. That's the way most people do this. They have a database of all the machines on their network. Then, out of that database, that generates zone files. It generates forward zone files. It would generate

example.com. It would also generate the corresponding reverse map files, like 2.0.192.in-addr.arpa.

So, that's probably more than you ever needed to know about reserve mapping. But, it is a thing, and you do hear about PRT records, so I did want to mention it.

Now, there are many more types of records, as I said. This is just a sampling of them, not to talk about them, but to just you the idea that there's all kinds of crazy stuff that people can put into DNS.

Some of these I have literally never seen, looking in the upper right. I have never seen an ISDN record, ever. I know that it exists. I don't know that anyone is ever using them. So, some of these get defined. They get maybe used a little, and then they get effectively abandoned. Some of them are up and coming. But, there you have it.

The point here that I want to make with this slide is that DNS is extensible. We think of DNS, as I know I said multiple times here, as mapping name to IP addresses. Of course, that's the main thing it's used for. It's critically important. But, it's extensible. Anything you can think up, any kind of piece of data that you want to associate with a domain name, you can generate a resource record type for it. You can get one, move it through the standards process and get it created, and then you can put your crazy piece of data into DNS associated with domain names.

An example of that, not that mine is the crazy part – a recent example of that – is the TLSA record. That lets you associate a digital certificate,

an X.509 certificate, with a domain name. So, that's another way to say, "This website has this digital certificate."

Now, you have to trust what's in DNS, and that's a separate issue called the Domain Name System Security Extensions, or DNSSEC. But, the point is, if you can trust what comes out of DNS, then it's pretty reasonable to say, "Well, I have a domain name, and here's the associated digital certificate with it." Now you don't need a certificate authority to tell you to trust that certificate. You can the trust that you get from DNSSEC to trust when you find a piece of information in DNS that says, "This name has this digital certificate." So, that's an example of a relatively new thing, a relatively new use of DNS, that's enabled by DNSSEC.

I don't have any time to talk about DNSSEC today, but the idea is that, once we layer that on top of DNS, you can trust the information that comes out of DNS. So, if you start to think, "Well, now, if I really have this trusted store, where I know that what I put in there is going to come out the same on the either end and a bad guy can't get in the middle and change it, if I really trust what's in DNS, what kind of things can I do with it?" We're just starting to see people think of that.

So, if you take the various records that I've shown you and you put them all together, you would have something like this. This is a sample zone file. This is literally the text file that would on-disk somewhere that would represent the example.com file, the example.com zone. So, this zone file. Every zone file is going to have an SOA record and some NS

records because that's just required. Every zone has an SOA record and NS records, and then everything else in the zone.

Most zones are this small because, if you think about it, for most domain names – not all – what do you need? Well, you probably have a website, and you probably want e-mail to work.

So, this is an example zone that has enough information to show what the IP addresses are for the example.com website. You can see the A and the Quad A record. And it has MX records to say, "Where should the mail go for this zone?" So, this is very typical, a zone this small.

All right. So, the last thing I want to talk to you about today sort of ties everything together, and that is to talk about the resolution process, which is, knowing everything that we know, when a stub resolver sends that DNS query to a recursive resolver and says, "Here's a domain name. I want the IP address," what happens? How does the recursive resolver find the information?

Well, a DNS query always comprises three paraments: the domain name, the class, and the type. I didn't talk about class because class is one of those ideas that we had early on in DNS for a way to extend it that never got used. So, as a result, the class is always what we call the Internet class.

So, the thing to care about here is the domain name and the type. So, for a DNS query, you could say, "I have the domain name www.example.com. I want the address record." So, that's a DNS query. You always have to specify the name and a type for a DNS query.

There are two main kinds of queries. Stub resolvers send what are called recursive queries. This is how a stub resolver says, “Help. I’m dumb. I don’t know how to do anything complicated, so I need you to tell me the exact answer to my query. I need you to do all the work and tell me the answer.”

Now, recursive resolvers are considerably smarter and more complicated, and they send what are called non-recursive or iterative queries. What this query says is, “I am a smart recursive resolver. You can either send me the answer, if you have it, or you can send me a partial answer and I will figure it out from there.”

So, let’s see how this works. So, resolution starts at the root zone. What that means is that it starts at the top of the name space and it works its way down. I think, having started at the name space on a few slides now for the past hour, you can see how that works. If you start at the root, you just work your way down and you can find anything in the name space. That’s how DNS name resolution works.

But, what does it mean to say, “Start at the root”? Well, there is a root zone, and it starts by sending a query to the authoritative servers for the root zone. Those are very special servers, and they’re called root name servers. So, root name servers is a short way of saying the name servers that are authoritative for the root zone.

Now, if we’re going to start at the root, we need the names and IP addresses of the root name servers then, just to start off. We have to know how to reach them. How do we do that?

Well, the answer is that you have to configure them. Every recursive resolver has to have them configured. There's no way to sort of come up and discover them. When your laptop came up on the ICANN Wi-Fi network here, it said to the network, "Help. I don't have an IP address. I don't know anything," and the network said, "That's okay. Here you go. Here's your IP address. Here's some other configuration information. Enjoy Barcelona." That's the DHCP, the Dynamic Host Configuration Protocol. So, your laptop could come up with no information, and the network says, "Don't worry. Here you go."

A recursive resolver cannot do the same thing. It can't come up on the network and say, "Help. How do I find the root name servers?" It has to be configured with the list of root name servers.

Nowadays, of course, where everything is packaged up for you by you operating system distributor or your software developer, the root name servers come configured in recursive resolvers already. But, the point is, somebody had to put them there in the first place. The recursive resolver can't magically discover them.

So, this list of root server names and IP addresses is what we call the root hints file. This is simply how you tell a recursive resolver, "Here you go. Here's how to reach the root name servers." That URL that you can't quite see is the domain name for the canonical location of the root hints file. It is at the internic.net domain name, which is sort of a blast from the past for those of those who have been doing this for 30 years.

But, here is the list of root name servers. This is what the root hints file looks like. So, it starts with 13 NS records. You see that there's a dot on

the left. A free-standing dot is how we represent that top node, the root node, in the name space.

So, what this says is that the root itself has these 13 NS records. You can see that they're named A.root-servers.net through M.root-servers.net. So, those are the names of the 13 root name servers.

Each of those root servers has an IPv4 address. You can see that that's the next list. Then, it has an IPv6 address. So, if you give this file to a recursive name server, it knows how to start and get going and contact a root server when it needs to.

So, let's take a slight detour. It's a detour, but it's going in the right direction here. Let's talk about the root zones since it's so important, since it's where name resolution starts. How does the root zone work?

Well, we don't even begin to have enough time to talk about all the complexities of administering the root zone. But, it's somewhat complicated. Two organizations cooperate to administer the contents of the root zone. This is a relationship that goes back, ultimately predating the existence of ICANN. So, it's 20 years plus. The predecessor to ICANN had this role, and the predecessor to Verisign had this role as well.

So, ICANN, via something called the IANA functions, does part of the process. Verisign, via a role called the root zone maintainer, does the other part of the process. I'm talking about the contents of the root zone. I'm talking about the root zone file itself. That is administered by ICANN and Verisign in those two roles.

Then, we have the root servers themselves, the actual authoritative name servers on the Internet that load the root zone file and answer your queries. So, the root name servers are operated by 12 different organizations. So, remember, the root servers are named A through M.root-servers.net. So, these are those 13 letters. Then, these are the 12 organizations that operate them. There are 12 instead of 13 because Verisign is the one operator that operates two. They operate A and J. The reason for that is complicated and happened a long time ago. But, Verisign operates two.

So, if you look at this list, what's interesting is that we have commercial organizations, educational institutions, organizations inside the U.S., organizations outside the U.S. We have some U.S. government there. We have ISPs. We have a little bit of everything.

So, if I gave you a sheet of paper and I said, "I want you to write down 12 different organizations that have nothing in common," you would be hard-pressed to get a better list than this. The only thing that these organizations have in common is that they run a root name server. This is kind of unusual because most zones have one organization running the authoritative servers.

Let's take ICANN.org. So, ICANN.org is run by ICANN. ICANN runs all of the authoritative servers for ICANN.org. That's how it works. Sometimes you outsource it to a DNS provider. But, still, one organization is ultimately responsible for all the authoritative servers for a zone. The root zone is different and special in that way in that we have these

twelve different organizations running the authoritative servers for the zone. The reason for that is complicated as well.

Very briefly, the history is that, when DNS was first getting going, somebody had to run these things, had to run the root servers and other top-level domain servers. So, a guy named Jon Postel, who was an early Internet pioneer and ran the IANA functions long before ICANN even existed to run them, realized that we were going to need root server operators, people to run root servers – these critically important servers – for this new thing called DNS.

So, he went around to people he trusted who knew what they were doing, who had the resources and good networks, and he said, "Hey, would you run a root server?" They said yes. This is a slight oversimplification of history, but that basically led to the list that you have here, at least up through I – no. Actually, the whole thing. So, basically, it was informal agreements, being asked by Jon Postel, that led to this list of organizations being root server operators.

Then, unfortunately, Jon Postel died tragically too early, almost exactly 20 years ago. So, because of the complexities here – this is just when the Internet was becoming more commercial and things were getting more complicated – it has been very hard to change this list of operators. Nobody has really known how to do it. Just now, it's RSSAC, the Root System Advisory Committee, has spent a lot of work, a lot of time, and a lot of effort coming up with the processes to finally talk about, "Well, how would we change this list? How would we add an operator? How would we remove an operator?"

So, this is an example of one of those things that has been this way on the Internet for a long time. It just is this way because it is, because it has always been that way. That doesn't mean it can't change. It just means that that is the way it is today.

So, the root server operators collaborate to run the root-servers.org website, where you can find out more about the root servers as a whole, including this map showing where they all are.

Now, there are more than 13 dots on that map. The reason is that there are more than 13 physical locations where there's a root server. Root servers make use of a technique called IP Anycast that let's you take a server and, rather than have a name server with an IP address appear in one place, it can appear in multiple places on the network.

Using that technique there are, today, around 1,000 what we call root server instances. So, there are 1,000 root server instances. There 1,000 physical places where you can contact a root server and it can give you a response.

So, the root zone is by far the most well-provisioned zone on the entire Internet. There are more root servers than there are any other authoritative servers for a given zone.

One last slide on this. At a very high level, and as I want to have the disclaimer in the upper right that this is a gross oversimplification of the actual process, here is how the process for changing the root zone goes. So, the information in the root zone is all about TLDs because the root zone delegates to top-level domains. So, it makes sense then that the

organizations that would want to make changes to the root zone are the TLD managers.

So, when a TLD manager wants to make a change to the root zone – and this is a pretty small list of changes. It would be adding a name server for their TLD, removing a name server for their TLD, changing a name server's name, changing a name server's IP address.

So, that doesn't happen very often, those things, but when it does happen and a TLD manager needs the root zone to change, it submits a change to the IANA functions operator. The IANA functions operator has a root zone database with information, including metadata about who the operator is, who the contacts are – things like that. So, the IANA updates its root zone database, and it does a bunch of checks to make sure that the request is legitimate.

Then, it sends that request to the root zone maintainer, which is Verisign. Verisign has its own root zone database. It changes the database accordingly. It creates a root zone file.

What I don't show here is that there's a process to cryptographically sign the information in the root zone file with DNSSEC. That happens as well. Then, Verisign makes that root zone available via its root zone distribution network. That's a set of servers all over the world that one can get the root zone from.

But, not just anybody can get the root zone from that. It's designed for only the root servers themselves, A through M. I've just represented

them like this. But, of course, remember, there are 1,000 boxes in that lower level, not just 13.

So, this is how you can see the two roles Verisign has in this process. They are the root zone maintainer in the big box in the middle. They maintain the contents of the root zone. They are also the operators of two out of the 13 authoritative root servers.

All right. So, that's the root zone. I could talk much more about it. I will restrain myself. But, that's pretty important to the resolution process.

So, speaking of the resolution process, let's go through it. Let's show how it works. So, let's run an example. So, here we have – could you…

So, here's an example. We have our iPhone here. Somebody has brought the web browser, the Safari icon down there in the lower left, and they say, "I want to go to www.example.com," so they type that into their phone.

So, the Safari web browser says, "All right. I need the IP address for that name if I'm going to contact that web server." So, it calls the stub resolver, which is provided by iOS, the iPhone operating system, and it says, "All right. Here's a domain name. Get me the IP address."

So, the stub resolver, when that iPhone came on the network, when it got its IP address, got some other configuration information, including the IP addresses of recursive name servers. So, one of the things you get when a machine comes up on the network as part of your configuration is the configuration for the stub resolver. You get IP addresses that tell the stub resolver where to go.

So, in this case, the stub resolver is configured to go to the recursive resolver at 4.2.2.2. Anyway, I hope I got all the two's in there. So, the stub resolver then now sends a DNS query to the recursive resolver, and it says, "I'm looking for the IP address of www.example.com." What it's really saying is, "I need the A records for www.example.com."

So, to make this example more interesting, because of caching, we're going to assume that there is no caching involved. We're going to assume that this recursive resolver has just started up. So, the only thing it knows – it read the root hints file back here. So, this is all it knows. All it knows is the name and IP addresses of the root servers.

So, this recursive resolver, when it gets this query, has to start at the root. So, it sends a query to one of the root servers. There are 13. They're equivalent, so it picks one at random. Different recursive resolvers, different kinds of software, follow different algorithms to choose authoritative servers. That's beyond the scope of this class as well.

But, let's just say that this recursive resolver chooses l.root-servers.net and it sends it a DNS query. You can see that the DNS query it sends it is exactly the same one that the stub resolver sent it. It says, "Do you have the A record for www.example.com?"

Now, the root server here does not have the A record for www.example.com. It doesn't even know about example.com. It does know about com, however, because the root zone delegates to the dot-com zone.

So, the root server here will send back the best information that it has to the recursive resolver. It's going to send back what's called a referral. It's going to say, "Well, here are the name servers for dot-com. That gets you in the right direction. Go ask one of them."

So, the recursive resolver caches that information, and then it sends a query to one of the dot-com servers. I remember that, early in the class, I showed you the list of what the dot-com servers are. There are actually 13 of those as well, and they are named similarly to the root servers, which is potentially confusing. They are named gtld-servers.net, A through M.

So, the recursive resolver picks at random c.gtld-servers.net, a dot-com server. It asks it the same question again. It says, "I'm looking for the A record for www.example.com."

Now, the dot-com server does not know the address for www.example.com, so it can't answer the final answer. It does, however, know the name servers for example.com because, remember, the dot-com zone is full of delegations to second-level domains under dot-com. So, the dot-com server definitely knows the name servers for example.com. So, it returns in a referral listing the name servers for example.com.

So, now the recursive resolver caches that, and it contacts on of those name servers. Let's say it picks ns1.example.com, and it asks for the third time for the IP address for www.example.com.

Now, in this case, example.com is authoritative for the zone that contains that record, so it can say, "Yes. I can answer that. Here is the IP address for www.example.com." The recursive resolver caches that, and then it returns it to the stub resolver, which returns it to the application that has been waiting.

That is the resolution process. That is the most simple way that it goes. It can be considerably more complicated.

Let me run another example. Caching really speeds this up because, as I noted, after that previous query, the recursive resolver, 4.2.2.2, has been caching things, and it now knows the names and IP addresses of all the dot-com servers, the names and IP addresses of all the example.com servers, and specifically the IP addresses for www.example.com.

So, now, let's say there's another immediately following the first, and this time it's to go to ftp.example.com. So, the stub resolver follows the same process. It constructs a DNS query and sends it to 4.2.2.2, the recursive resolver.

But, this time, the recursive resolver can say, "Well, I know how to contact the example.com servers, so I can just contact an example.com server directly. I don't have to do all that other stuff." Then, it can get the answer and return it to the client much faster.

So, you can see that a given recursive resolver is not going to have to contact the root servers very much because, at this point, they're like, I want to say, 1,600 TLDs. About that. So, the recursive resolver is only

going to contact a root server when it encounters a query for a TLD that it hasn't heard of before and that the time-to-live values on the records in the root zone are 48 hours – again, for historical reasons. So, a given recursive resolver isn't going to have to contact the root servers that often.

However, there are a lot of recursive resolvers, so the root servers are very busy. They just are busy from queries from all over the place.

The way that this process can complicated is that these referrals – let me go back a little bit to give a specific example. Let's take a look at this referral right here. So, the dot-com server is giving back the name servers for example.com. If the name servers for example.com are all in example.com – let's say they're ns1.example.com and ns2.example.com – then the dot-com also knows those name server IP addresses. That's called glue. That's that thing that I skipped over pretty quickly.

But, in that case, the referral is completely self-contained. It contains the name of the name servers and all the IP addresses for those name servers. So, the recursive resolver can go immediately to the next step, as I showed in the example.

But, it doesn't have to be that way. You can name your name servers anything. So, what if the name servers for example.com were in isp.net? Let's say that example.com outsources its name server to a third party. This is very, very common.

In this case, the referral from dot-com is going to say, “Here are the name servers for example.com. They’re named in isp.net.” The dot-com server does not know the IP addresses for isp.net.

So, now the recursive resolver says, “Well, I know the names of the name servers for where I need to go, but I don’t know their IP addresses.” So, now the recursive resolver has to put this query on hold, and it has to resolve the names of those name servers in isp.net.

Well, what if the isp.net zone’s name servers are named in yet some other zone? Which is entirely possible. So, you can see how this gets more complicated. The recursive resolver has to have multiple queries going at once on the stack to satisfy dependencies of other queries. So, showed you the most simple example.

Recursive resolvers are complicated pieces of software. They’re considerably more complicated than authoritative servers. You can write a fully functioning authoritative server in under 2,000 lines of C++, but you can’t do that for a recursive resolver. It’s considerably more involved.

All right. And that is the end of my slides. We have about ten minutes. We go until 10:15. So, if anybody has any questions, I’d be happy to answer them. Or, you could just sit and be kind of overwhelmed. It’s up to you.

Okay. If you could please come to the mic.

UNIDENTIFIED FEMALE: A reminder, please. Please state your name and your affiliation. Thank you.

UNIDENTIFIED MALE: There we go.

COLE QUINN: Right there. Okay. I'm Cole Quinn with Microsoft. A quick question about reverse mapping. If you do have an IPAM platform that's managing all your reverse zones –

MATT LARSON: Yeah. IPAM is IP Address management. It's a class of software.

COLE QUINN: Yeah. It is an all or nothing thing? Like, if you decided to go down the PRT route and made sure that there's a PTR record for each record, is it all or nothing? Or, is there some benefit from having PTR records for some of your A records in your forward zones but it doesn't necessarily need to be 100% mapped back and forth?

MATT LARSON: That's a good question. The answer is, I think it's the rare organization that has reverse mapping for every single one of its IP addresses.

Well, for one thing, most IP address space is not statically allocated. Look, there's huge net blocks that ICANN owns to service the Wi-Fi

networks at an ICANN meeting. But, there aren't the same machine that those IP addresses – it's all of our laptops, and they're moving around the entire time.

So, for this big chunk of address space for ICANN, the possible machines at those IP addresses are going to change all the time. So, there's really no point in trying to have a name associated with those IP addresses in those dynamic ranges.

On the other hand, for things like mail servers and web servers that stay where they are, that don't have variable addresses – pieces of network gear, like routers that, again, stay where they are – those are the kinds of infrastructure devices, infrastructure IPs, that you typically do have PTR records for that do reverse map to names.

Was there another question?

FIRDAUSI FIRDAUS: Thank you. My name is Firdausi. I'm from Universität de Barcelona. I'm a bit a new at this, but I'm kind of curious. If, for example, because I saw that, from the list that you made about the root servers and operators for addresses or something like that – I was wondering when, for example, for law enforcement and something like that, when it's concentrated in one country, an area or something, how easily, if we need something – I don't know the technical stuff – but is it important as [inaudible] and so on – how do we get this?

Is it like we need to maybe contact the government or contact ICANN directly? Or, how does it work?

I'm also kind of curious about some of the important data. Do you have a kind of data retention policy? You mentioned that there is a server that runs the databases for the IP addresses and so on. So, could you tell me about this? Because I'm not really from a technical background. I'm more from a legal background. So, maybe briefly. Thank you.

MATT LARSON: Sure. My colleague, John Crain, is the better person to talk about the law enforcement angle to all this and how law enforcement interfaces with all of the domain name provider ecosystem to get information that they need, whether it's from registrars, registries, or ICANN.

Typically, the information that ICANN has is relatively level. It's about the top-level domain administrators, and that information is all well-known and public. It tends to be the registries or the registrars that have information about specific domain names that law enforcement might be interested in.

I apologize. I can't speak in any level of detail about the processes that law enforcement follows to contact registries and registrars.

As for data retention, ICANN spent as much as effort as everyone else, if not more, on GDPR compliance to get on the right side of that. So, while I am not a lawyer and I don't even play one at ICANN meetings, I do know that we spent a lot of time and energy to confirm that we're compliant with not only GFPR but with whatever other data retention legal frameworks there are.

So, I just simply don't know enough information to comment specifically, but I'm confident that our legal department has done all the right things there.

Any other questions?

JOAO PEDRO: Hello. My name is Joao Pedro. I'm with the NextGen program. You mentioned, in the parameters, the ones that usually we don't see but can be used for security. So, [it'd be] a protocol in the upper layer.

My question is, using UDP as the transfer protocol, does it guarantee the same security? What kind of things would be involved to guarantee that we couldn't spoof the website name?

MATT LARSON: I will talk in three minutes about DNS security. You mentioned UDP. So, for the most part, all the transport for DNS queries is what we call UDP, which is much simpler than TCP. TCP actually sets up an association, a connection, between two machines, and there's a formal protocol to exchange information that they follow.

UDP is much simpler. It's one machine blasting a packet at another machine. So, most DNS queries: one packet over UDP. Most DNS responses: one packet over UDP.

So, what that means is that it's possible for a bad guy to send a response spoofing the IP address, pretending to be the IP address.

In this example here, there are ways that a bad guy could impersonate ns1.example.com with the wrong answer, a malicious answer, and that the recursive resolver would cache that and return it to the stub resolver.

Now, it's not easy, but it is possible. That's been a fundamental issue with the DNS protocol since the very beginning, which is why, not long after DNS was invented, we started talking about, "All right. How do we layer security on top of this?" That's where the DNS security extensions come from. That adds cryptography to all this. So, all data in DNS gets a digital signature.

This response here, for example, would be not only the IP address for ftp.example.com, but it would be a digital signature over that address. Then, the recursive resolver could choose to validate that digital signature to confirm that the answer is legitimate.

Now, that's the quickest version of DNSSEC that anyone has ever explained because it's a lot more complicated than that. Every zone in the tree has a public-private key pair that it uses to sign its data. Parents sign the keys of their children. You thought recursive resolvers were complicated before DNSSEC? Add cryptographic validation on top of that and they get even more complicated.

But, ultimately, that is the solution to this. When you have a protocol based on UDP, like DNS is, it's very easy to slip in extra packets and try to maliciously confuse the actors. The only way around that, the way that we found with DNS, is DNSSEC, to layer on security using cryptography and to make digital signatures and to verify them. It adds

a considerable amount of complexity, but it is the only way to provide that assurance. Or, at least, it's the way we've chosen to provide that assurance.

Well, I think we're about out of time, so thank you very much. I'll be here for a few more minutes if you have any questions you want to ask me directly. So, thank you.

UNIDENTIFIED FEMALE: Thank you, everyone. A reminder that, in about 15 minutes, our next How It Works session on understanding DNS abuse will be here in this room at 10:30. That will be presented by Bryan Schilling, the Consumer Safeguards Director at ICANN.

So, you have a few minutes to stretch your legs and come back. Hope to see you then. Thank you, again.

[END OF TRANSCRIPTION]