COPENHAGEN – How It Works: DNS Fundamentals
Monday, March 13, 2017 – 17:00 to 18:30 CET
ICANN58 | Copenhagen, Denmark

STEVE CONTE: Okay. I'm not expecting a huge turnout here because we're up against the public forum. So all of you guys, get up, come to the table. Come on. We're going to change this around a little bit because we don't have a whole bunch of people here.

So we're going to tailor this around you guys. I've got 68 slides that I can present and put you all to sleep before dinner time or we can make this fun, as much fun as DNS can be and a little interactive.

My thought is that I'll go through the Resolution Process Map stuff, go over a couple of slides, and then I really want to open it up and find out why you guys are here. What specifically about DNS fundamentals you're interested in? Let's take a deeper dive and see if we can hit the pieces that — and you can walk away with the pieces that have interest and meaning to you.

I've got a whole lot of data dump here. I can talk about resource record types. I can talk about DNSSEC. I can talk about other things but if it's not what you want, then my job isn't complete here. Because we're a small group today, I'd rather do that.

If you guys are all okay with that, then that's how I'm going to go. Okay. I got one thumbs up and I got a whole lot of people looking at laptops so I'm going to take that as an affirmation.

I'm Steve. I'm the Director of Programs for the Office of the CTO. One of the things we do is we run this How It Works tutorial series and the DNS Fundamentals is one of them and we're going to go ahead and go from there.

The DNS, in the simplest terms, is it's a way for — don't sit down over there. I see you coming in. There's chairs. Come to the table. Please join us.

The reason the DNS came about is because remembering numbers is hard and you can remember a few phone numbers and whatnot but you get more than a handful in your head and then you can't remember them anymore.

So the DNS was a way to enable humans to get to IP addresses, to servers, on the Internet without having to remember the IP address. It's fairly easy when there's IPv4 and you only have 3.4 billion addresses to remember. It gets a little bit more difficult when you're jumping to the IPv6 and you have a hundred – duodecillion numbers to remember.

I was told one time that if you imagine the size of IPv4 to be about as big as an iPod, the relative number space for IPv6 is the

planet earth. So that's a lot more numbers to remember. Naming comes in really handy in general on that.

We're going to talk about a couple pieces of the DNS and I'll throw this slide up. We're going to talk about recursive name servers, we're going to talk about authoritative names servers, and we're going to talk about stub resolvers. If you look at it from a perspective here, the stub resolver typically sits on your laptop, your phone, whatever device that you, as the consumer of the data, have. It has a call to whatever application you're using, so in this example here, it's a phone using Safari browser and there's a mechanism within the phone that has a stub resolver on it.

When we talk about recursive name servers, that's typically going to be an ISP. They will have name servers, if you ever go into configuration on your laptop and you see an IP address, you'll see a subnet mask and you'll see a subset of name servers. That's typically being given to you by your ISP or if you're at your office by your office network.

Then we'll be talking about authoritative name servers. These are the root system and the top-level domains which include .com, .net, include country code TLDs, ccTLDs, .dk — we're here in Denmark, or .uk or .za for South Africa.

Those are the components we'll be talking about. I want to do a quick look at the namespace.

The DNS was designed as a hierarchal database. Back in the old days, it was a flat file. It was centrally controlled and it was a single file in a single entry. It was run by SRI and it was easy when there was just a smattering of hosts on the Internet. At that point, in the early days of the Internet, there were just a couple of universities, some U.S. government agencies, and whatnot on it and it was really easy to manage.

But as the Internet became more and more popular, it was clear that they needed a new methodology to do that so the IETF, the Internet Engineering Task Force, developed the Domain Name System, the DNS. They did it as a hierarchal database that's distributed and distributed among multiple entities that can manage different pieces of it.

When you look at www.example.com, we'll go up the chain here. Then we have the dot. The "." is always inferred there and we'll talk about that. But we have .com, which is a registry, a top-level domain. It's a registry and it's managed by a separate entity. It's managed by Verisign for this case. Example is another entity. This is actually the domain name of there. If you have a business, if you have your own domain name, that would be managed by you.

And then either third-level nodes, which could be another domain space or it could be a host. In this case, we're looking at "www" which would be a host. So when we build that, we have hosts, we have domain, we have registry, we have root.

And if we drop it down and we flip it back up, we flip it back up this way, we have www.example.com and that's how your browser knows where to go. All of those are maintained by different entities. It's hierarchical. It's easy to see. It's easy to understand where the management points are but it's distributed too and that allows for a lot more leeway, a lot more scalability between that.

Because now if we look at the domain space and just .com, and we're talking about millions and millions of addresses, so having a single entity for .com and .net and .uk and .coffee and .whatever, we'd be talking about a huge file that would have to be updated all the time. By distributing that it makes it much easier to manage.

Now, we're going to jump all the way to number 40. This is all the data dump. We can go back and talk about this as we have our dialog but we're going to talk about how the name resolving works. Okay. Here we go.

Resolution process. So you're on your stub resolver, which is, again, your laptop, your iPad, your telephone, whatever, and

you're trying to go to an address and you want to go to www.example.com. You type it into your browser and your browser sends that request over to the stub resolver. It's says, "Hey, I want to go to www.example.com." And your stub resolver —and I'm sorry. This is with the assumption that all these are fresh queries. You just turned on your phone. Your ISP has just turned on their name server. There's nothing in the cache. It's never done anything before.

You send the query to the stub resolver. It says, "Hey, I want to go to example.com. How do I get there?" Stub resolver says, "No clue." But when I got on the network, I was given an IP address for a name server, which is, in this case, 4.2.2.2. So I'm going to go ask them.

The stub resolver sends the query over to the cursive name server and says, "Hey, what's the address to www.example.com?" Recursive name server says, "I don't know but I know how to get to the root server so I'm going to go ask the root server." Were any of you in the last session with RSSAC? Some. None. Okay.

It acts now on your behalf. The recursive name server for the entire rest of the resolution process will be your proxy. It will be asking these questions until it gets an answer. So, it first goes to a name server and it says, "Hey, someone in my network wants

to go to www.example.com. I don't know how to get there. How do you get there?" And the root server says, "I don't know, but I know how to get to .com. So I'm going to give you the addresses for the .com name servers."

Here they are. Now your recursive name server goes to the .com name server and says, "Hey, I want to get to www.example.com." The .com name server says, "I don't know but I know how to get to example.com. The entity that's managing example.com." It says, "Go talk to the name servers at example.com."

It says, "Okay." The recursive name server goes down to example.com, to the name servers and says, "Hey, I'm looking to get to www.example.com." Now, the example.com name server says, "I know how to get there because example is a host in my zone file in my configuration. Here's the IP address for www.example.com."

Passes that backup and recursive name server says, "I have the full answer." So now I'm going to pass that full answer back to the stub resolver and that's going to pass the information back to the application. In this case, again, Safari.

At that point, the "www" host is sitting here. At that point, you can have a direct conversation because you know the IP address, the unique identifier to that server. Your machine will

go and talk directly to the web browser — in this case, the web browser. It no longer needs the DNS because it knows the IP address.

That's DNS. Thanks for joining me today and if you guys have any questions then – no, there's a little bit more.

That's cool. That's great. That took an enormous amount of time in computer space. That took probably about, maybe up to 100 milliseconds to get all that information. I don't know if you all remember dial-up when the Internet first started and how much you had to wait and wait and wait and wait. 100 milliseconds is actually really quick in human space and you actually don't notice it.

When you're looking at your laptop and you're going to Nike.com because ICANN meetings are boring and footwear is not and so you want to go look at new sneakers. That resolution will take very quick but it could be quicker.

Now you want to go to — you're still at example.com and you're browsing the web there and they say, "Well, there's a file you can get and it's at FTP.example.com." FTP is a protocol. It means File Transfer Protocol. So you're like, "Oh, I want that file. I need to go there." We go through the process again — go back to that slide.

This time we're going to say — type into your browser. You're like, "I'm going to go to FTP.example.com." You've never been there. You don't have any kind of history here so you send it to the stub resolver. Stub resolver sends it the recursive server and says, "I need to go to FTP.example.com."

Now, if we took it from before, the next step should be — the answer is I don't know and I'm going to go talk to the root. And the root is going to come back and say, "I don't know. Go talk to .com." .com is going to say, "I don't know. Go talk to example.com."

But we just did that, right? We just did that with www.example.com so that information of — the information that the root had gave back and the information that .com servers gave back the first time is now stored at the recursive name server. It's now cached which means that the next step is actually saying, "I already know how to get to the name servers for example.com so I'm going to go directly and ask them."

We've just cut off two-thirds of the query process by doing that. Now, you go to the name server for example.com and say, "Now I'm looking for FTP." It says, "Oh, I know that. Here's the address." Send it back and now you can go and download the file.

That's called caching. That takes place at almost every level, at least in the non-authoritative levels. If we remember stub resolver, recursive name server, authoritative servers. Authoritative servers are everything from a root server to a top-level domain, a registry and to a domain server for a domain name server as well.

That side doesn't cache. There's really no reason for them to be caching. This side caches because — especially at the recursive name level, it's going to be getting a lot of queries. If you're on an ISP, if you're at home, and people are coming through and you want to CNN or whatever news network you go to, there's a good chance that the rest of the community that you're in, the rest of the network, that there's more people asking for those questions as well.

Caching saves a lot of time and space at the recursive level because there's going to be 180,000 other users possibly asking questions that are similar especially at the top-level domain level. No matter what the domain is there's still going to be a .com, a .net, a .dk, and the chances of people asking those same questions are really good, at least at that level, so that means they don't have to touch the root server. They don't have to touch – well, they do have to touch that to get to the next level.

Does that make sense? Anyone have questions or comments about that part? Go ahead, please. Use the mic because we do have people online.

UNIDENTIFIED MALE: So the recursive name server is usually the ISP?

STEVE CONTE: ISP or if you're in a business network, they'll have their own name servers. There are a few that are open on the Internet. Google runs some. 8.8.8.8, is a well-known open server but those are usually an ISP if you're looking at a traditional model.

UNIDENTIFIED MALE: How often do they flush the cache?

STEVE CONTE: That's a great question. We have something in the domain, in the zone file so that every domain is managed by a zone file. That zone file is at every single authoritative server. At the top-level, the zone file just points to the next level down. It has all the configurations for the name servers for the next level, for the top-level.

At the registry level, it's pointing down to all the domains. At the domain level, it has all the information for its hosts. In each of

those, in the zone file, there is something called a TTL, a Time to Live. That TTL is a variable and it really depends on how the administrator for that zone — someone — there's an entity here that's separate that's managing this zone compared to managing this zone or compared to managing Nike.com.

They will set up a default TTL and what the TTL does is when it comes back to the proxy, to the ones that are caching, they'll say, "Here's the answer and this answer is good for X amount of time." Typical configurations if you just open up a name server out of the box — name server software out of the box and you're just setting it up blindly, typical is 86,400 seconds which is 24 hours. One day. You can set changes to that. You can set changes based on the host or you can just have one blanket TTL.

So if we say one day, 86,400, when that answer comes back to the recursive name server, it will cache that and it will start the timer countdown. After 86,400 seconds it'll drop it from the cache. It won't go necessarily re-ask that question. It'll drop it from the cache until that question is asked again.

It might not happen very often at the domain level but almost guaranteed as soon as the TTL expires on the root, they're going to get another query and they have to go ask the root. So that probably gets updated most quickly in the cache.

Makes sense? Any other questions? Kathy?

KATHY: We have a question from a remote participant, Alexandrine Gauvin. What are other examples of recursive name servers other than ISPs?

STEVEN CONTE: Well, we have the stub resolver so the answer is also cached at the main server or at the recursive name server, ISP or in your business. The answer is cached on your computer as well. If you go to www.example.com and then two hours later you go there again it doesn't need to do any questions as long as the TTL is still not expired. It will go directly and talk to that.

Usually any kind of name server that's not authoritative, not a domain name server, not a top-level domain server, or not a root server will have caching.

Anyone else? Yes, please.

UNIDENTIFIED MALE: As I understand it, there is some kind of load distribution for the root servers by using anycast. How does that work in the next level for the sublevel domain servers?

STEVE CONTE: Anycast is a brand new concept, relatively speaking to the Internet.

Let's step back a little bit on that. There are 13 entities of root servers. There's letters A through M. Those entities back before anycast were limited by the packet size of a UDP packet. UDP packet means I'm going to send a query — it's almost a fire and forget type packet. I'm going to send a query and I'm not going to worry about what the answer is because you're either going to get it or you're not.

Unlike TCP, which means I'm going to shake your hand the entire conversation. UDP is a very small, very fast packet. That's why they chose that. The biggest size of a UDP answer was 512 bytes at that point. Within that answer, within that 512 bytes, they had to pack in the amount of root servers and their relative IP addresses to that and still have room to have an answer as well.

In that, they being the DNS gurus of yesteryear, they decided that 13 was the magic number. That's the most amount of DNS servers — root servers we could pack into a single UDP packet and send back at one time without fragmenting that answer. So that's where we got the magic number 13.

That was great but we got more and more traffic on the Internet and 13 servers, although at the time was very redundant, attacks

were relatively small compared to today and it was able to function for quite some time. But it became apparent that the network, the Internet was scaling was getting larger. Attacks were getting larger. The distribution of the root servers was not optimal. There was two root servers, three root servers, four, five maybe more root servers located in the U.S. Others in mostly Europe, parts of Asia, things like that.

If you are out of those regions, your query time to a root server could be quite long. It could be going undersea cables, you could be going over satellite feeds. It could be long as far as latency goes. It could be costly as far as bandwidth goes especially when you're going multiple queries for a root server.

The DNS community started talking about that and they developed a process called anycast. What anycast does is it basically makes — it fools the networks. The Internet is about unique identifiers. You can only have one IP address to one machine. It's on a one-to-one ratio. This machine here, my machine, has a single IP address and none of you in this room have that same IP address. It's almost like a fingerprint.

They had to figure out how to scale the root service and still maintain the concept of having that uniqueness for it. And anycast fools the network using BGP, which is a routing protocol and they determined that if you put a server on one network and

announced its IP address and its route and said here's — I'm trying to think of L-Root at that time. 198.32.64.12 was L-Root at that time. Here's L-Root. It's over there.

Now, if I go to a different network and I announce the same address in the same block, normally if you're in a TCP mode that's going to break things because if I'm standing right in the middle between the two servers that are exactly the same IP address I might start my conversation over here but at the same time I might have a flap and suddenly I'm talking to the machine over here but it has no record that I was talking to it. It has no idea and I was like, "What?"

I have to start my handshake again and that takes time. But with UDP — UDP is a fire and forget technology. You send a packet, you're not expecting an answer. You're hoping for an answer but you're not expecting one. You're not just handshaking.

If I'm in the middle, I could send a UDP query to either one of those because I'm in the middle of the Internet so I'm getting announcements from both sides. From this network here that I'm announcing and I'm getting an announcement from this network. I'm going to choose the best path where I'm sitting.

I could ask that instance and I'll get an answer because it's UDP. Even a second later, if I'm asking that instance and suddenly this

is the best path, I still will get the answer and I'll be like, "Oh, that's cool." I could be doing the same thing from here.

We were able to mimic and mirror the concept of having a unique IP address and multiple servers. So we grew from 13 actual servers to over 600 instances of root identities so we still need to keep the number 13 because we're still restricted by the packet size of 512 bytes of a UDP packet. But because of routing tomfoolery, we are able to pretend that those 13 are actually 600 different announcements on the network.

Did I get your question in that?

UNIDENTIFIED MALE: Yes, but I was actually more interested in the next level.

STEVE CONTE: Okay.

UNIDENTIFIED MALE: Because I would think that a lot more people will ask something .com and then those —yeah. There would be a lot more requests and how do you distribute that load?

STEVE CONTE: Once anycast was established and they decided that it worked quite well, we started working at the root servers and it worked. Some of the other top-level domains also said, “Hey that would equally work for us, too.” Some of the other top-level domains decided that they wanted anycast as well.

When you’re talking about sharing load then it becomes really the decision to load share becomes the decision of the client asking the query, not the server with the response. You’re not going to go to a server and it’s going to do a traditional load balance and say, “Well, instance A is loaded up so I’m going to send you over to instance C.” There’s no control mechanism. I’m going to be in the network somewhere and I’m going say, “Well, my fastest route to that server root TLD, whatever, is this way. I’m going to go talk to that.” Even though there might be three other instances around the globe.

That’s why it becomes the network who manages the load balancing and it’s based on fastest path and all the other intricacies of routing. It works really well with UDP so it’s really good for DNS-type traffic. It works not so well with TCPs so when you’re looking at load balancing in a traditional sense for web-based technologies or other things, then you really need a traditional load balancer there that you’ll talk to and then it makes the decision on where to offset your data.

Yep? Please.

UNIDENTIFIED MALE: I wanted to ask about the issue of TTL. What's the downside, like setting the TTL to zero let's say in case — to run my domains to have those propagation issues when I change name servers?

STEVE CONTE: That's a great question. The question is: why do we have TTLs of variable and not just as a static? Why don't we just not even have a TTL?

One of the answers is that TTL is meant to manage your cache. If you had a TTL of zero, your recursive name server would not be holding any cache data at all. It's going to come back and say, "Oh, my TTL is zero or my TTL is one. Oh, done. Throw it away."

As zero it's going to get the answer and it's going to say, "Oh, there's no TTL. There's no cache." And it's going to throw it away and that means that anybody else who's going to the domain, the fully qualified domain that you're looking for, the recursive name server is going to have to go through the whole process again and rediscover the root and rediscover the TLDs, and then finally get down to the domain.

We're only talking about hundreds of milliseconds on a single end user. That's not that big a deal. When you're at the root level, though — sub-question — what do you think gets more traffic, a TLD or a root server? Anyone? TLDs get a lot more traffic than a root server. When you're asking a question, a single question to a root server — I used to run L-Root back in about 2006 or so and at that time, this is seven years ago, more than that, an average happy day we were getting about 12.5 million queries a second and that was with caching.

Now, image that without caching. The amount of load that would take place. This, as we went through this, remember this only points to the top-level domains so when the recursive name server asks for www.example.com and says, "I don't know but here's .com." And .com then caches, right? You get the caching answer so the next time it goes it's going to ignore the root server. That's why a TLD gets more queries than it does the root server.

So if we take that no caching scenario again — seven, eight years ago we had 12.5 million queries a second. Here we probably have 10, 100 times that and if we have to cache, or if we have zero caching and we're querying that over and over, the load on the network, not just the provider, the data provider that the TLD is sitting on but the load of the Internet itself will become explosive.

We want to have some level of caching. We need to balance now what's a reasonable amount of cache versus what's too fast and too slow and that's really where the administrator comes in. When we look at the data that the root servers have, they're only serving top-level domains. They're only giving the answers to the questions to top-level domains.

That doesn't change very often. Almost, maybe a handful, one handful a month. Then the numbers are very small. It's a pretty safe assumption that if we set this cache to very high that if there's a change, it's okay. We can wait X number of hours. I think the default for a root server is it caches for 12 hours. If there's a change in the data that's pointing to the TLD servers, it's probably not a massive change so 12 hours isn't so bad and they can make changes.

Here at the TLD, there's probably massive amounts of new registries, new registrations for domain names all the time. If you look at .com and you look at the amount of registrations they make, at any given moment, they're probably getting — I'm making stuff up here — probably getting thousands per second. That needs to get into their top-level domain zone file pretty quickly and if I'm a registrant, if I'm registering a domain name I don't want to wait 12 hours or 24 hours before my website is available. I want that be like now. Right?

Their TTL, their Time to Live is probably pretty small because they want to make sure that the cache goes away quickly so they can go get a new answer so they can capture those new registrations.

Now, when we're here, this is businesses. This is your private domain name. This is whatever business you work at. For the most part, some things will change very seldom. Your e-mail server addresses probably don't change very often unless you do a renumber.

Some of your web services probably don't change very much. You want to keep those at a reasonable, 12, 24 hours; 8 hours even is probably fine. That way when they go it they say, "This data is valid and this data is valid for the next 8 hours." Unless you're really doing so network changes in your network, that's going to be fine.

If you are doing network changes and you're moving your services from one web server to another web server, then you, as the administrator of the zone of your domain can go in and say, "I'm going to keep all the other ones cool. I'm 24 hours or 8 hours on my mail servers is fine but I'm changing servers for my web service." I want that to be small because there could be problems so I'm going to set that down to five minutes of cache. TTL of five minutes, whatever that comes out to seconds.

That means that if I make a configuration problem or if there's an issue with the new web server I can back out of that quickly and the most that my customer is going to be impacted is five minutes because they might hit a cache. They might be pointing to a server that doesn't exist anymore, an IP address that doesn't exist.

If I keep it small while I'm doing maintenance things, super handy. I want to make sure that I change that back, once I'm stable, to a reasonable level. You want to balance the freshness of your data to the load of the network and the load of the resolver and stuff like that.

Kathy?

KATHY: Another question from Alexandrine, the remote participant: "What is the average TTL of most TLD servers?"

STEVE CONTE: I don't know if there is an average and I honestly haven't looked lately. I would guess, well — no, that would take too long. I would guess a couple of minutes because a TLD, a top-level domain and I would guess that they'd want to have their caches flush often so new domains that are inserted, or introduced in

that top-level domain, will be propagated on the network quickly.

Seconds, you're saying? 300 seconds is what I'm hearing from over here. That's pretty quick. That's what 60, 120; 3 minutes, 3.5 minutes roughly. Five minutes maybe? Yeah.

We'll go into some more detail but that's really DNS. DNS is about who you know. Trying to get here. "Hey, do you know them?" He says, "No, but I know him." "Hey, I'm looking to get to www.example. Do you know him?" He says, "No, but I know him." Same thing. "No, I don't know that but I know him." And he's finally like, "I know him."

It's like going to a party and you want to meet that person across the room but you can't get to them directly so you need to ask a friend to introduce you to the next friend, to introduce you to the next friend, until you finally get to that person you want to meet. That's what DNS is. It's all about who you know.

What interests you guys? We can go into the nitty gritty, the details of resource record types and mail exchange record types and things like that but we're a small group. I want you to come away with the question that you came here for in the first place.

Anyone? Come on. Please.

UNIDENTIFIED MALE: I'm curious about the — I read an article about introducing the Blockchain concept into DNS.

STEVE CONTE: Blockchain, I don't know a lot about. I will refer you — whenever they say I don't know. That's called a referral. I don't know but I know there's going to be a session on Emerging Identifier Technology taking place this week. I believe tomorrow at 11:00, I want to say. I think that has Namecoin. I think it might have some Blockchain information [Frogan]. I'm going to leave that to those who know what they're talking about in that realm. So I'll give you that referral. Okay?

Please.

UNIDENTIFIED FEMALE: I'm wondering a little bit about the history of the DNS system, in a sense. From my understanding of ICANN as an organization, there were two types of contracts with the United States. One was the White Paper, Green Paper Stream that led to the DNS and one was the IANA Stream where they will lose multiple contracts with their amendments that went to the AoC and then the JPA. I'm wondering how these two are different, the DNS, the IANA or is the IANA basically the DNS system?

STEVE CONTE: That's a great question and I'm going to answer it as much as I can. I'm not a policy person so I'm going to get some of the details of the contracts wrong. I'll preapologize for that.

ICANN was created because before ICANN, the DNS was operated by a gentleman named Jon Postel and he was based out of USC University ISI. Back then, if you wanted to run a top-level domain you would write or call up Jon and say, "Hey, I want to run this top-level domain." And Jon looks through his — they say it's the infamous black book that he would keep. So He would flip through his book and say, "Yeah, that's not being run here. You're the administrator to that."

That worked pretty well in the early days but as the Internet became more and more popular, it became apparent that it was going to break the bounds of University and break the bounds of government, or DARPA stuff. They needed to develop a way that — there was one registry and registrar at that point which just happened — it was Network Solutions. If you wanted a domain you would go there.

We had no business model. We, as the Internet, had no business model and they wanted to change that so that there was more diversity on the network and more opportunities. Many people, there were people from across RIPE NCC and other entities from around the world said, "Let's take that directly out of U.S.

government control and let's create an organization that would handle that."

Then we'll take a sidestep. Jon Postel and his little black book was the IANA at that point. It wasn't just domain space. It was IP addressing. It was also other unique identifiers on the Internet. Jon Postel and Joyce Reynolds and a handful of others were the IANA. They were the ones who were assigning unique identifiers.

That got changed to the concept of the IANA function instead of the IANA person. It traditionally became called the IANA function. Now we want to move this IANA function out of direct government, U.S. government control. How do we do that?

Well, they decided at that time that opening up a not-for-profit organization and letting that organization manage the IANA function was the way to go. Right or wrong, indifferent. That was a lot of years ago.

They came up with the concept of ICANN. Primary goal of ICANN, in the beginning, was to house and run the IANA function and to promote more diversity and name registration because this is when the .com was starting — the bubble was starting to grow and create more diversity on the network.

ICANN was founded, I believe in 1998 and it brought in the IANA function. Now, at the time ICANN was under an MoU,

Memorandum of Understanding with the U.S. government Department of Commerce. At the same time, the IANA function was still under the auspices of the U.S. government but the DOC didn't want to run it directly. Before they were letting USC ISI run it.

There was a separate contract of performance that was taking place and then we had this contract of Memorandum of Understanding and that's why see two as we move forward. We had ICANN's MoU that needed to continue forward to reach whatever milestones that the contract mandated but ICANN was running the services of IANA, the function of IANA and that was given to them under contract through the Department of Commerce.

We had one side that was a Memorandum of Understanding. There was a recognition that ICANN was the entity that was going to manage the domain space. But IANA as a function — it was a contract that needed to be given to somebody by the U.S. Government.

As that matured, through the years, we saw the MoU turned into the JPA or something like that. The IANA contract stayed as IANA contract and then last year or the year before that is when the Department of Commerce said, "We feel ICANN is mature

enough now as an organization," whether you guys agree or don't. I'm not here to argue that semantic.

The U.S. government felt that ICANN was mature enough at that point to be able to handle running the IANA function as an entity that's not mandated by the U.S. government. The U.S. government wanted to step back and they wanted to have an equal partnership role with the rest of the Government Advisory Committee.

That's when we started seeing the work on the IANA transition taking place. That all turned out to be a new organization PTI now runs the IANA function that has agreements with ICANN and I don't know the nitty-gritty details on that.

That's how we moved from U.S. government stewardship completely and then a parallel road down to where we stand today. I hope that answered your question.

UNIDENTIFIED FEMALE: Yeah, but just one more quick follow-up. So the PTI is a separate organization. Is it not within ICANN?

STEVE CONTE: I don't know for sure.

UNIDENTIFIED FEMALE: Okay.

STEVE CONTE: There's a very close partnership to that and I don't want to misrepresent what that partnership is. I'm not trying to evade you, I just don't want to give you a wrong answer.

UNIDENTIFIED FEMALE: No, no. Sure. And one last thing. When we're talking about these two separate types of contracts, is it more or less correct to say that the IANA function is more about IP allocation and the DNS function is more about the naming itself?

STEVE CONTE: The IANA function is more about the unique identifier management and that includes the DNS from a technical perspective from managing that but it also includes IP addressing, autonomous system numbers, port numbers. There is a list of registries of unique identifiers that IANA is either the repository for managers. ICANN is not responsible for the market — I'm going to say this probably wrong, too. I'm a tech. I'm going to policy stuff wrong. I apologize. They're more about the market diversity of the domain name system. If I said this wrong, don't go up to the open mic on Thursday and say, "That Steve guy said." Because I'm already wrong. I know it.

You have a question? Kathy.

KATHY: I have a question from remote participant Alexandrine: "What are the current challenges of the DNS and how will it likely evolve to meet those challenges?"

STEVE CONTE: That's a great question. I'm going to open that up to you to guys. Do you guys have any idea or thoughts of what the challenges of DNS is and evolution? Please?

UNIDENTIFIED MALE: Obviously recently we had the DNS outage.

STEVE CONTE: Yeah.

UNIDENTIFIED MALE: Which took down a lot of big sites so that was obviously a pretty obvious flaw in the DNS architecture, I would say.

STEVE CONTE: I would agree with that. There was an attack on Dyn a while ago. It's interesting and I don't want to misconstrue what Dyn is

because they are a very robust network and they got the smart people there. I don't think it was the fault of Dyn that the ramifications of this attack took place. I think what we're seeing is an explosion of size of attack vectors that we're starting to see more and more of.

The root servers have always been under attack. If we looked at the root server logs right now we would probably see an attack going on. The rule of thumb has always been to over-provision the root servers to be able to handle that.

Dyn did the same thing. They're anycast, they're over-provisioned. I think one of the challenges was that maybe some of the consumers of Dyn put too many eggs in one basket and that means that they put all their services, DNS services, within the DY Dyn N infrastructure.

When Dyn took the brunt of that attack many of the consumer, which is Amazon or whoever was on there. I don't know for sure if it was Amazon. They felt the brunt because they had no secondary servers or not many outside of the DYN infrastructure. That's, I think, certainly is a challenge that we're facing is that botnets are becoming larger. They are becoming smarter.

You mentioned blockchain. I think that's a potential evolutionary path for naming. I don't want to say specifically against the DNS but for naming functionality. DOA, Digital Object

Architecture, could be another evolutionary path for naming on the Internet. There's been attempts at other methods in the past and DNS seemed to be more robust.

As far as evolution goes, we've seen the 2008 or 2009, we saw the root sign with DNSSEC which is not a cryptographic signature as far as it won't encrypt the data that's being asked but it does authenticate that the request you made, the answer that you're getting from the request you made is coming from the entity that you ask that question to.

It's just a chain of trust that you're asking the question. You're getting the answer from the person that you thought you asked the question to. That's getting more and more, we're not 100% penetration on DNSSEC yet. We're still working on that.

The next neat thing is just around the corner and, I think, as we see new ideas evolve, such as blockchain, such as Namecoin or whatever, there's going to be an opportunity to either replace or enhance the Domain Name System and that's yet to be seen, I think.

Anyone else? Please.

UNIDENTIFIED MALE: Hi. It's a technical question. Looking at your example.com illustration up there, the caching resolver, how strict does it

have to be about honoring the TTL because, I don't know if it's true, but I've been told at some large commercialized piece when running caching DNS will not respect the TTL given out by Upstream-Name Service.

STEVE CONTE: I've seen that happen before. That falls into what ITFers and system administrators want it called, BCP, Best Common Practice. TTL works, just like the DNS works just because people believe it works. They say it and this is the TTL or the DNS structure works because people believe it and they'll install servers that follow that belief.

I know you can override a TTL at a recursive nameserver level if you'd like. Some try to do that. Hotels, in the past, have done that. I've seen where they will act for the best interest of their patrons at the hotel and they will keep that TTL for three or four days or more because it costs them and bandwidth to go get new answers.

There is a functionality to do that. I can't speak to any specific ISP that's doing that and I hope that unless there's a real need to override the BCP or the TTL in that, I would hope that people just let the network be organic and run as it was developed to be.

Yeah. Then I'll go to you.

UNIDENTIFIED MALE: Okay. Great. Could I make one other question, if I'm allowed?

STEVE CONTE: Yeah.

UNIDENTIFIED MALE: We see on the WHOIS record and such spaces the terminology primary and secondary name servers.

STEVE CONTE: Yes.

UNIDENTIFIED MALE: Technically, is there any difference?

STEVE CONTE: Yes and no. Okay. See if I can get to the slide here on that.

At any authoritative level, you should have more than one name server and that's going to help against redundancy. It's going to help against topographical and geographical availability. Having 13 identifies of the root server but over 600 instances is a very resilient network.

If you attacked the root servers today, the root servers would be like, “Oh, get away from me.” If you attacked the root servers eight years ago, you’d be hitting them up here but because of that you have a much more diverse network and that’s resilient to attack. It’s diverse and it allows quicker responses from other places and if there is an outage in any part of the network, there’s other places that it can route too.

Having a number of name servers serving a specific zone is beneficial. I don’t know the number of TLD servers especially when we look at .com or .net. I suspect it’s very high. If we look at a smaller ccTLD, it might be smaller. The number of name servers might be three or four of them.

When you look at a company, a domain name, they might only have one or two of them. Hopefully, more than one. One is like, if you can’t reach the name server for your company, then no one can reach you and if you’re a bank, you can’t make money. If you have a domain, you should have at least two, hopefully, more, name servers.

Now, the information they’re giving is identical. So if I have three name servers in example.com, the information that they serve, you could touch any one of those and they should be giving you the exact same information. One of those typically is called a primary server and the other ones are called secondary servers.

The only distinction between that is as an administrator, you want to only put that information, the new information, into one server and not three or not 600. You don't want to go 600 times and type changes in.

You have one server that acts as a primary server and then the other one, the secondary, will go and ask occasionally. That also has almost a TTL. They'll go ask occasional to the primary server and say, "Hey, do you have any changes?" The primary server will say, "Nope." It'll say, "Okay, we'll ask you again." Comes back a little time later and says, "Hey, do you have any changes?" And this time the primary server had a change that was put in by the zone administrator and they say, "Yeah, we do." And it passes that new information to the secondary servers then.

There would be a period of milliseconds which the data might not be in sync but it would be sync very quickly after that.

Now, on top of that, your primary server might not even be publically available. You might hide your primary server behind a firewall to protect that data because that's the data that you, as the zone administrator, are touching. You might put that behind a firewall and only allow access to the secondary servers that are publically available.

The only servers that might be touching the primary would be the secondaries and it might not act as a recursive server or as a public server, I should say.

UNIDENTIFIED MALE: Just take that example then. The hidden primary server. That would not appear in WHOIS record, correct?

STEVE CONTE: I'm sorry. Say that one more time.

UNIDENTIFIED MALE: You've got a hidden primary name server with multiple secondaries so that would not appear in the WHOIS record, would it? Because it's not been publically cleared?

STEVE CONTE: Correct.

UNIDENTIFIED MALE: It's that cross-over between the technical side and requirements for those WHOIS records because obviously registrars normally take WHOIS records and put it into the correct stream data.

STEVE CONTE: Yes. So if you had a hidden primary it would not be in the WHOIS because you'll only want to serve your secondaries.

UNIDENTIFIED MALE: So what was the rationale between the original WHOIS record showing the primary and secondary tertiaries or name servers?

STEVE CONTE: WHOIS is one of the oldest protocols on the Internet so the rational is that they didn't think of it at that point. Even back then, though, there was still no differentiation between primary, secondary tertiary, whatever. They should all be serving the same data.

Ma'am, I'm sorry. Thank you.

UNIDENTIFIED FEMALE: [inaudible] from Sudan, ICANN Fellow.

STEVE CONTE: Welcome.

UNIDENTIFIED FEMALE: It's a technical question related to DDNS, its Dynamic DNS. For example, in case I have a home lap and I need to access it remotely but the public IP I provided by the ISP is changing. I

have told that DDNS, setting up Dynamic DNS it can help. So how it works? Thank you.

STEVE CONTE: I agree with that and it does help especially if you're behind a private network. My understanding of Dynamite DNS is that it really is playing with the TTL of the domain name on that. They're setting the domain name very low so that they can have dynamic changes very quickly.

It's probably much fancier than that at the application level where the server is, but from the DNS perspective it's probably just a very short TTL which means that the domain that you're dynamically assigning to has a very short cache time which then if someone is coming back to you and you just assigned a new address, they will catch it very quickly before and it won't timeout on them.

Sir, let me bring the microphone over to you.

UNIDENTIFIED MALE: Statistically speaking, when you're migrating a set of DNS servers, what is the biggest issue that has been faced by the various registrars? So when you're changing DNS servers and basically you have an outage.

STEVE CONTE: When you, as the domain holder, are changing DNS servers. Okay. What you're hitting with and that's what they call a propagation period where you're changing DNS servers or DNS providers or something like that is you're submitting new DNS information up to the TLD. If I change the NS or the IP addresses from my name servers at the domain level, I've got to push that up to — and I usually do that through a registrar and I've got to push that into the TLD, I am now subject to the TLD's caching – the Time to Live on that.

As you mentioned before, we might be looking at five minutes but that's probably not every single TLD. If you're at a lesser known top-level domain or a country code TLD, they might have a longer period of the — the TTL might be longer which takes longer than for the caches to expire and propagate so then they have to go ask that question again.

If you are migrating your name servers, you might want to find out what the TTL is on your top-level domain to that domain. If you are five minutes, then your propagation is probably going to be very small. If they do have a 6, 8 hour, whatever, then the caching of the last time that they spoke could be up to that amount of time.

It's really a question of the TLD and what their default TTL is on that.

Sir. Just to let you know. Him then you. We are at time. I'm happy to stay if you guys want to ask more questions but I want to let — feel free to step up and go at any time.

UNIDENTIFIED MALE: I just wanted to give a shout out to a couple of sites like "whatsmydns" and "checkmydns" where you can check the global propagation level. When you've made a change or a customer has made a change you can look on this site and see what the global propagation is and then you can send a link to that customer.

STEVE CONTE: Checkmydns and whatsmydns, is that what you said?

UNIDENTIFIED MALE: Yep, yep.

STEVE CONTE: Probably you need to Google those.

UNIDENTIFIED MALE: Google those.

STEVE CONTE: Am I 6:30 instead. Oh, we got 28 minutes. So ask your question. Ask it slowly. We've got plenty of time.

RAZA QURESHI: I'm Raza from MCI. As I [noted], two other DNS type in [telecom] industry, [inaudible] DNS and PS or 3G, 4G DNS. Is there any convergence program to converge these three kind of DNS, especially only one we are following up such project such as IOT (Internet of Things)? Thanks.

STEVE CONTE: That's a good question and I don't think I can answer that one. I don't have the information or the knowledge on those DNS. So I apologize on that.

But I did think of a separate thing for you. Sorry.

So your question was moving your nameserver addresses to different name servers. Something you could do, since you're a good Internet citizen, you've got more than one name server, you can have overlap. Move one name server to your new server and keep your other one running. Let that new name server go through propagation. It's now live and then you can move the other one over as well.

You basically set it up as a third name server. If you have two name servers, just add one.

UNIDENTIFIED MALE: [inaudible]

STEVE CONTE: Yes. So it has the same information but it's using the new IP space that you just moved to. As it comes seen around the Internet, then take on off and add another one and that way you can slowly migrate yourself from one network to another without having any loss of propagation.

Did you have another question?

UNIDENTIFIED MALE: Just on that topic. If you do that, so long as the details are the same, there will be no conflict.

STEVE CONTE: Yes. You want to make sure that the data that the new name servers are getting is exactly the same and being a slave, a secondary to that data. Then once you have them moved over then you can deprecate your old numbers and remove them completely from you NS records and stuff.

Any other questions, comments? Nothing?

All right. We've got 26 more minutes. We can just sit here quietly then. Yes, please, sir.

UNIDENTIFIED MALE: Can I just ask again about caching. NXDOMAIN is no record found. No result.

STEVE CONTE: Yes.

UNIDENTIFIED MALE: And that's never cached. Is that true?

STEVE CONTE: I believe it's true. I'd have to actually look that up. That's a great question. So the NXDOMAIN is a flag in the DNS that gives you a positive/negative. It's kind of weird but it does. This allows the DNS servers, the caching servers and it must have a cache because it's going back to the recursives.

To say if I'm going to blahblahblah.com, and no one registered it, instead of just not getting an answer, which is ambiguous, because it could just be as far as the DNS, the recursive name server is going it could just think that the name server didn't get

the question or it didn't get the answer so it's going to ask that again. By giving a positive/negative from a TLD or from the roots, it's saying, "I heard you and there's nothing I can give you. There's no such domain."

So there must be a TTL on that because if we go through new TLD rounds, new gTLD rounds and we're introducing new top-level domains, next week there could be a .steve and I might be on the luckiest guys in the world but if that doesn't have a cache and it returns an NXDOMAIN because someone searched something.steve, up here a year ago, if it doesn't deprecate, if it doesn't expire as a TTL, then they would never, ever get there. I would assume that there is a TTL on NX record as well.

Nothing. All right. I'm going to let you guys go then. This is your chance. Going, going.

There is a more formal slide deck. It is online. It does have a whole lot of data and boring stuff. We're a nice small crowd. I didn't want to go through that and just have you guys glaze over. I'd rather have the dialog and the conversation today.

I hope it was okay for you. It was way more fun for me. I've done this a lot and I'd rather have a conversation then point and say, "Next slide please."

Thank you guys very much. Feel free to ask questions. I'm here throughout the week. Feel free to download the slide deck. Do whatever or feel free to ignore me in the hallways. That's cool too. I will take no offense to that.

Thank you all.

[END OF TRANSCRIPTION]