About the IIJ Public DNS Service

2.1 Introduction

IIJ released the beta version of its IIJ Public DNS Service in May. It is a caching DNS service that only accepts DNS over TLS (DoT) and DNS over HTTPS (DoH), meaning that it does not support name resolution via the usual UDP/TCP setup, and it is available to anyone, not just IIJ users.

This report explains how DoT/DoH differs from the usual DNS setup and describes key considerations and future challenges for IIJ in providing this service.

2.2 What is DoT/DoH

2.2.1 DNS and Privacy

Information registered to the DNS is assumed to be publicly and widely available. For a long time, therefore, the focus of DNS security has been to ensure that the information is not tampered with (i.e., that its integrity is maintained), whereas ensuring that the information is not intercepted (i.e., that it remains confidential) has not been a priority.

In 2013, however, the Snowden affair revealed the existence of PRISM, an extensive communications monitoring and information collection program carried out by the US National Security Agency (NSA). This prompted the IETF to declare that "Pervasive Monitoring Is an Attack" (RFC 7258) and call for protocols to be designed to withstand pervasive monitoring going forward.

As it became clear that DNS was also being monitoring under PRISM, the IETF began developing mechanisms to ensure DNS privacy, until then a perfunctory affair, through its new DPRIVE (DNS PRIVate Exchange) Working Group. DPRIVE has published various protocol extensions/revisions for DNS, including Qname Minimisation (RFC 7816) and EDNS(0) Padding Option (RFC 7830, RFC 8467), with transport encryption being of relatively high importance among these efforts.

2.2.2 DNS Transport Encryption

Traditional DNS mainly uses UDP for the lower-level protocol (transport), supplementing this with TCP. However, plain UDP/TCP, and DNS over UDP/TCP, do not have a mechanism to provide confidentiality, and communications are easily eavesdropped since they take place in the clear. A decision was therefore made to insert an encryption layer between DNS and the lower layers to provide protection.

Various encryption layers have been proposed, with the following having been standardized so far: DNS over TLS (RFC 7858), DNS over DTLS (RFC 8094), and DNS over HTTPS (RFC 8484). A draft of DNS over QUIC has also been submitted to the IETF and is now under discussion. And if HTTP/3, which is also currently under discussion, is standardized, DoH will automatically support HTTP/3 (Figure 1).

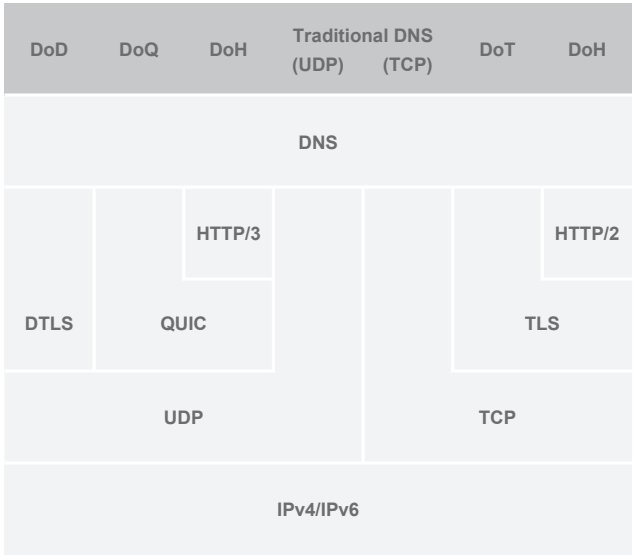

Figure 1: DNS Transport Protocols

These various encryption layers are not integrated into any one protocol; at present, you simply select whichever suits your users' circumstances. But flooding the space with a bunch of different protocols can have its drawbacks, so it is quite conceivable that a subset will be selected, and the rest deprecated, at some point in the future. (At present, DNS over DTLS is a specification only, with no existing implementations, and it seems unlikely that it will become available for use.)

2.2.3 Transport Encryption and DNSSEC

DNS already has a mechanism for verifying DNS information via digital signatures called DNSSEC. So why do we need a new method of transport encryption when we have DNSSEC? And will transport encryption eventually make DNSSEC unnecessary?

Before answering this question, let's look at the scope of transport encryption. With DNS, clients do not directly query the server on which the master DNS information is registered (authoritative server); instead, they query caching servers provided by ISPs and other parties. In general, the caching servers are responsible for querying the authoritative server.

Transport encryption is currently only performed between the user and the caching server. Communications between the caching server and the authoritative server use traditional DNS and are not encrypted.

In general, encryption guarantees both integrity and confidentiality, but when it comes to DNS transport encryption, encryption only happens between the user and the caching server. Since communication between the caching server and the authoritative server use traditional DNS (not encrypted), the integrity of the information obtained by the caching server cannot be guaranteed, and its integrity cannot be guaranteed even if the information is protected by encryption. That is, unlike encryption in general, the DNS transport encryption protocol only guarantees confidentiality between the user and the caching server (Figure 2).

DNSSEC, meanwhile, was introduced to protect against data forging and manipulation. It uses digital signatures to guarantee integrity, but communication itself takes place in the clear and is not confidential.

So both transport encryption and DNSSEC are mechanisms for improving DNS security, with one focused on guaranteeing confidentiality without integrity and the other focused on guaranteeing integrity without confidentiality. In that sense, they are each other's complement, so one cannot replace the other. Each protects something different, which means that transport encryption does not obviate the need for DNSSEC, and vice versa.

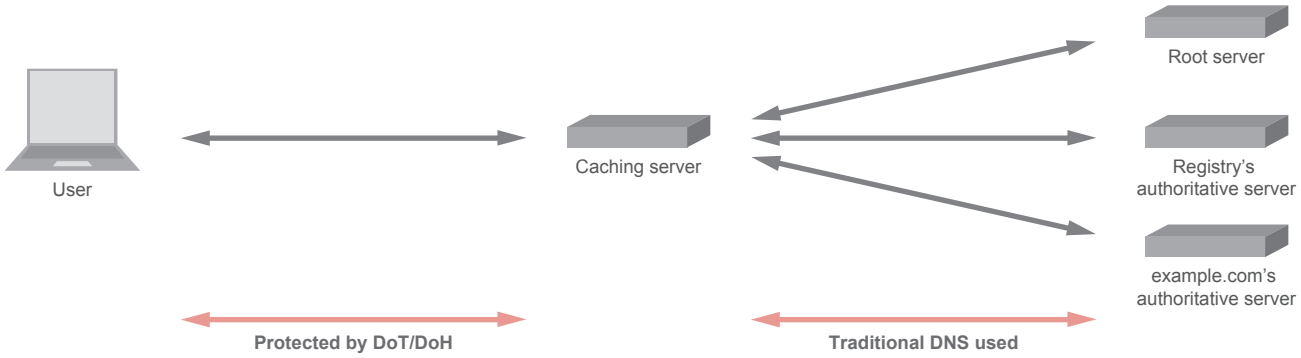

Figure 2: Scope of Transport Encryption

2.3 IIJ Public DNS Service and DoT/DoH

DoT and DoH differ from traditional DNS only in terms of transport-layer protocol. They both use the same DNS protocol as traditional DNS. Yet we still had a number of barriers to overcome to provide them as a service. Let's look at each in turn.

2.3.1 The TCP barrier

The major difference between DoT/DoH and traditional DNS is that all communications take place via TCP before being encrypted using TLS. While traditional DNS can use either TCP or UDP as the transport protocol, in most cases UDP is used, with TCP only being used in limited cases.

With TCP, a session must be established before communications on higher-layer protocols can start. TCP also uses various techniques to ensure the reliability of transmissions, which include checking that sent packets were in fact received and resending them if necessary.

DNS involves very little data exchange. In most cases, both the query and response rarely exceed a few hundred bytes. When TCP is used here, the process of setting up and later terminating the session accounts for vastly more of the communications exchanged than the actual DNS message, resulting in extremely poor efficiency. Possible solutions to load problems include the extravagant approach of simply adding more servers, but nothing can really be done about increased latency stemming from an increase in packet roundtrips.

UDP has no such mechanisms, making it fast and simple, and it is widely used in protocols that involve the exchange of small packets, such as DNS and NTP. Its reliability, on the other hand, is low, and many of the attacks on DNS that have so far been discovered, such as cache poisoning and DNS amp, really stem from the use of UDP in the lower layer rather than from any problems with the DNS protocol itself.

Although it is known that using TCP would preclude or greatly reduce the threat of such attack methods, the massive overhead that would result from the use of TCP with DNS—because DNS, by its nature, involves the exchange of a large number of small packets—has dissuaded the community from shifting to TCP as the main transport protocol. Even when the Kaminsky Attack[*1], which allows vastly more efficient cache poisoning that with previously known methods, was revealed in 2008, we still didn't switch to TCP and opted instead to make do with UDP and treat the symptoms with source port randomization[*2].

So there was considerable aversion to the TCP overhead with DNS, but requiring TLS naturally also means requiring TCP. The DNS over DTLS protocol, which uses UDP, does exist, but it entails a large overhead just like TCP, and moreover, no one can use it because it is only a specification; no implementations exist.

Nonetheless, we must abandon the conventional wisdom if we are to use it as a foundation for providing a secure caching DNS service. A major factor in enabling us to provide the IIJ Public DNS Service without restriction to anyone in the world is that we do not use DNS over the low-reliability UDP protocol, which dispels concern of the service being used as a launchpad for a DNS amp attack or such like. We should focus squarely on the benefits that using TCP actually provides.

2.3.2 The TLS Barriers

DoT and DoH eschew UDP and use TCP, resulting in a large overhead, and they employ a TLS encryption layer above the TCP layer. TLS is widely used in HTTPS and elsewhere, but it is certainly not lightweight, and it causes significant performance degradation with protocols that necessitate the high-speed exchange of small amounts of data, such as DNS.

*1 For example, JPRS Topics & Columns, "Aratanaru DNS cache poisoning no kyoui: Kaminsky Attack no shutsugen" [New DNS cache poisoning threat: Emergence of the Kaminsky Attack] (https://jprs.jp/related-info/guide/009.pdf, in Japanese only)

*2 Randomizing the source used by clients when sending queries increases the number of factors that an attacker needs to guess when seeking to forge packets, which reduces the probability of a successful attack.

Although logic says we will take a significant performance hit compared with traditional UDP-based DNS, we need tools if we are to measure how much performance actually degrades. TCP-based DNS has been in use for some time, albeit only in limited cases, and tools do exist. But DNS over TLS is completely new. And there are no satisfactory tools for measuring performance. We needed to measure how much performance degrades and what the processing load would be so that we could estimate just how much equipment we would need to provide the service, and to do this we had to start by developing performance measurement tools.

Since TLS involves a very high processing load, options are available to reduce the overhead by, for example, reusing information from a previous connection to resume a session (TLS session resumption) and, in TLS 1.3 (the latest version), adding the application data to the initial ClientHello/ServerHello exchange that takes place during the handshake (0-RTT).

But having certain functions available in the TLS protocol is meaningless unless applications actually use them. Running services in a large-scale production environment will be difficult unless you make full use of these options.

The IIJ Public DNS Service uses Unbound, a DNS implementation developed by NLnet Labs in the Netherlands. It is quite old and supported TLS before DoT became an IETF draft. When we investigated Unbound's TLS support, however, we learned that it lacked those mechanisms for reducing the TLS overhead; specifically, it lacked TLS session resumption capabilities. And performance measurements also indicated that performance was inadequate. Further, performance differs greatly depending on what encryption algorithm is used, but this was hardcoded into the source. So IIJ decided to implement the necessary features. The results were passed back to NLnet Labs, and the latest available version now includes our code.

Aside from performance issues, TLS poses one more barrier, namely that communications are encrypted.

To ensure a stable DNS service, you need a mechanism that allows you to collect statistics to make sure that a large number of abnormal queries are not being sent, and that abnormal responses are not increasing despite the queries being normal, so that you can investigate and take action if abnormalities do arise. With traditional DNS, in most cases this sort of statistics gathering and troubleshooting was done not on the DNS server itself but by capturing DNS packets. Since this process can be performed independently of the DNS server, the same method can be used regardless of what DNS server implementation is used.

But with TLS, the captured packets are encrypted. Perfect Forward Secrecy (PFS) is now commonplace, so packets cannot be decrypted even if you have the server's private key. This means that the tools so far used to collect information are no longer viable. The ability to collect statistics is needed for private testing of course, and it is indispensable if you plan to make something widely available as a service, so much so that you may as well shelve the service without it. So we had to rebuild the statistics collection functionality from the ground up to enable us to gather the same sort of information from the DNS server without relying on packet capture, and with this in place we were finally ready to launch the service.

2.3.3 The HTTP barrier

The barrier posed by HTTP was actually not all that high.

With DoH, once TLS has been applied, the DNS message must then be encapsulated in an HTTP message. Getting the DNS server itself to speak HTTP would be quite a

challenge, but a two-stage setup where an ordinary HTTP server receives the queries, converts the message format, and passes them to a DNS server behind it would be pretty much just like that used by any other Web application out there, aside from the fact that the backend is a DNS server.

Clearly because we are using TCP instead of the traditional UDP, as well TLS, the latency and other performance issues are unavoidable. But because we take care of the hard parts in the HTTP layer, which has a proven track record, and not in the DNS layer, which would require us to fumble around in unfamiliar territory, this should not be seen as too much of a setback.

2.3.4 Have we overcome the barriers?

As of this writing, the service has been live for about six months.

We did create a bit of a stir when we published the press release since this is the first DoT/DoH service in Japan, and the Android DoH client Intra[*3] subsequently added an IIJ Public DNS list option for users to select (we did not even have to ask for the option to be added).

I wish we could give a glowing account of how everything went off without a hitch post launch, but reality does disappoint.

As explained, Unbound did not have TLS session resumption, so we implemented it ourselves, but TLS session validity appears to be extremely short on Android 9, and our session resumption implementation has proved ineffective in many cases. When Android smartphone users visit a Web page and then follow a link from that page to another one, often the TLS session established to resolve the previous domain name has already timed out and the whole process has to start again with a handshake. This handshake frequently fails if the network is congested, which means that users are constantly unable to use the Internet because a name cannot be resolved.

Moreover, it looks like performance is even worse on Android 10 (currently under development as of this writing) than it is on Android 9.

Other than this, we have had not major problems. Latency should theoretically be worse than with traditional DNS, but it does not appear to be a problem in practice and we have not received any complaints.

DoT and DoH are still new technologies and the basic specifications have only just been finalized. A lot of the peripheral specifications are yet to be sorted out (for instance, DoT/DoH servers presently can only be configured by hand; network administrators are unable to distribute configurations for automatic deployment).

Going forward, we will continue to investigate issues with the aim of making improvements and implement new specifications that are on the road to becoming standards in the hopes that people will be able to use the latest technologies with peace of mind. And we will continue to give back by communicating the insights we glean from operating our service to the community.

2.4 Public DNS and DoT/DoH

In closing, we look at developments beyond the IIJ Public DNS Service.

Originally, caching DNS servers were only available to users within a particular organization. They were not public in nature. But the realization that making them public would not really be all that harmful eventually led to most of them becoming available without restriction (open resolvers). Later, however, the DNS amp attack was discovered and attackers

*3 Intra (https://play.google.com/store/apps/details?id=app.intra)

began to use open resolvers to stage DDoS attacks. So since around 2010, the scope of access has generally been restricted to the minimum necessary.

Meanwhile, other services aimed at a wide range of users sought to foil attacks by implementing rate limitations and the like, rather than address-based restrictions. The pioneer here is OpenDNS[*4], but this was later followed by the Google Public DNS[*5] service, and ever since that gained traction, these sorts of explicitly open resolvers have generally been called "public DNS" services.

The DoT RFC became a standard in 2016. The public DNS service Quad9[*6], launched in November 2017, and Cloudflare[*7], launched in April 2018, supported DoT from the start, and Google also added support in January 2019.

DoH officially became an RFC in October 2018, but implementations based on the draft came out ahead of that. Cloudflare provided support upon launch in April 2018, Quad9 added support two weeks before RFC 8484 was released, and Google also later added support in June 2019.

On the client side of things, Android has supported DoT at the OS level since August 2018, and the DoH client app Intra was released in October 2018. Among Web browsers, Firefox added DoH support in August 2018. In Chrome's case, only the development version supported it as of this writing, but the stable version may also have it by the time this is published.

Mozilla has stated that it plans to make DoH the default for name resolution in Firefox[*8], meaning that the public DNS services selected by Firefox would automatically be used unless the user configures the browser to do otherwise. But public DNS services, because they are public, cannot resolve namespaces such as those found on intranets. And services like parental controls that use DNS will be ineffective if the browser selects its own DNS server instead of what the OS settings dictate. In light of these points, there is debate about the pros and cons of imposing DoH as default.

Google Public DNS supports DoT/DoH, and use of these protocols ensures confidentiality between the user and Google. Meanwhile, Google also supports EDNS0 Client Subnet (ECS; RFC 7871). With ECS, when a client queries a caching server, the caching server relays information about the network to which the client belongs to the authoritative server. This is intended to be useful in content delivery traffic management. But it is important to note that traditional DNS is always used between caching servers that use ECS and the authoritative server. If the user is using DoT/DoH, communications on the route between the user and Google will not be eavesdropped, but communications on the route between Google and the authoritative server can be eavesdropped because traditional DNS, which does not guarantee confidentiality, is used here, and this could result in a breach of user privacy since the ECS information traverses that route.

There is now a definite move toward encrypting DNS transport, and it is unlikely to be stopped at this point. But as providers of a new public DNS service, we have an important responsibility not to blindly accept every development that unfolds but to evaluate each one by one and determine if it really is the right way to proceed.

Takanori Yamaguchi
Application Service Section, Application Support Department, IIJ. Mr. Yamaguchi works on support for DNS services etc.

*4 OpenDNS (https://www.opendns.com/)
*5 Google Public DNS (https://developers.google.com/speed/public-dns/)
*6 Quad9 (https://www.quad9.net/)
*7 Cloudflare (https://developers.cloudflare.com/1.1.1.1/)
*8 Firefox Nightly News, "What's next in making Encrypted DNS-over-HTTPS the Default" (https://blog.mozilla.org/futurereleases/2019/09/06/whats-next-in-making-dns-over-https-the-default)