ZeroDNS: Towards Better Zero Trust Security using DNS

Levente Csikor, Sriram Ramachandran, Anantharaman Lakshminarayanan
[csikor_levente,ramachandran_sriram,lux]@i2r.a-star.edu.sg
Institute for Infocomm Research (I²R), A*STAR, Singapore

ABSTRACT

Due to the increasing adoption of public cloud services, virtualization, IoT, and emerging 5G technologies, enterprise network services and users, e.g., remote workforce, can be at any physical location. This results in that network perimeter cannot be defined precisely anymore, making adequate access control with traditional perimeter-based network security models (e.g., firewall, DMZ) challenging. The Zero Trust (ZT) network access framework breaks with this traditional approach by removing the implicit trust in the network. ZT demands strong authentication, authorization, and encryption techniques irrespective of the physical location of the devices. While several prominent companies have embraced ZT (e.g., Google, Microsoft, Cloudflare), its adoption has several obstacles.

In this paper, we focus on three problems with practical deployment of ZT. First, the DNS infrastructure, a critical entity in every network, does not adhere to ZT principles, i.e., anyone can access the DNS and resolve a domain name or leverage it with malicious intent. Second, ZT's authorization procedures require new entities in the network to authorize and verify access requests, which can result in changes in preferred network routes (hence requiring additional traffic engineering), as well as introduce potential bottlenecks. Thirdly, ZT adds additional time cost, increasing the time-to-first-byte (TTFB).

We propose ZeroDNS, wherein the control plane of Zero Trust is implemented using the DNS infrastructure, obviating the need for a separate entity to issue authorization tokens. Since the control plane is implemented using DNS, it reduces the number of round-trips authorized clients require before accessing an enterprise resource (e.g., web service). Furthermore, we apply ZT principles to DNS, meaning access to DNS requires authentication, authorization, and encrypted communication. ZeroDNS uses mutual TLS for DNS communication for authentication, and only permitted clients with valid certificates can query domain names. We implement ZeroDNS on top of `NGINX`, a reverse proxy typically used as a load-balancer in enterprise settings. We show that the additional packet processing time in ZeroDNS has a negligible impact on the overall name resolution latency, yet it decreases TTFB.

ACSAC '22, December 5–9, 2022, Austin, TX, USA

ACM ISBN 978-1-4503-9759-9/22/12…$15.00
https://doi.org/10.1145/3564625.3567968

CCS CONCEPTS

• **Security and privacy** → *Security protocols*; *Domain-specific security and privacy architectures*; Web application security.

ACM Reference Format:
Levente Csikor, Sriram Ramachandran, Anantharaman Lakshminarayanan. 2022. ZeroDNS: Towards Better Zero Trust Security using DNS. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3564625.3567968

1 INTRODUCTION

Traditional network security enforces perimeter-based access control, where accessing enterprise resources within the network perimeter is (usually) unconditionally granted but from outside, access is only possible through a heavily protected single point of entry, i.e., a firewall. This principle stems from medieval history, when walls and moats surrounded fortresses, and the only access was through a drawbridge, which was strongly guarded. Therefore, it was assumed that everything physically inside the wall is safe, and everything outside is dangerous. This network security model (just as the medieval fortress) has a severe flaw; once the perimeter is breached, adversaries can freely move laterally, access and leak sensitive data.

This perimeter-based approach has not changed much over the years; its existence has even been enforced by the IPv4 address exhaustion problem [1] and the inherent Network Address Translators (NATs) and private IP addresses that were introduced to overcome it. The concept of DMZ (Demilitarized Zone) further fostered the perimeter-based approach, wherein the publicly available services (e.g., a company's website) are deployed at the perimeter, increasing the number of components of the network that can potentially be compromised.

However, it is becoming increasingly hard to define the physical perimeter. Due to the fast-paced evolution of network speeds and virtualization techniques, companies adopt (public) cloud solutions to benefit from low costs, high availability, and efficient resource provisioning. From a recent study [2], the respondents claimed that on average, *only* around 23% of their network will be kept on-premise, and the rest will be outsourced to the cloud. 5G technologies make it even harder to define a perimeter, because 5G enables massive deployment of IoT devices (e.g., for automation, monitoring) connected to the enterprise network from practically anywhere without having (a company-leased/-built) infrastructure. The recent COVID-19 pandemic has just strengthened our reliance on remote (net)work. Several companies now allow (and sometimes prefer) hybrid settings [2], i.e., employees can work from anywhere. Employees can connect (with their own devices) to the enterprise network from anywhere using VPN (Virtual Private Network) technologies. As a result, enterprises cannot assume that their internal network is safe as there is no such clear identification of internal

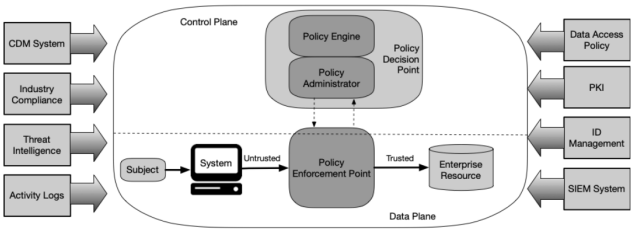


Figure 1: Core Zero Trust Logical Components according to NIST [56].

network anymore. Moreover, thanks to the BYOD (Bring-Your-Own-Device) models, not all connecting devices are closely monitored and managed by the enterprise.

In 2014, Google's BeyondCorp [63] proposed a security model where the implicit trust in the network is removed. This whitepaper popularized the term *Zero Trust* (ZT), wherein accessing any resource requires strong authentication of the device (e.g., via X.509 [61] certificates) and its user (e.g., via credentials), strong authorization (i.e., fine-grained access control), and strong encryption (i.e., TLS), irrespective of where the connecting device is physically located. However, this introduces additional entities to the network. Based on the NIST definition of ZT architecture (cf. Fig. 1), access requests are verified in the control plane by the Policy Engine (PE), and accordingly, the Policy Administrator (PA) generates corresponding authorization tokens. Furthermore, in order to enforce these policies in the data plane (e.g., allow, deny, or shut down connection establishments), Policy Enforcement Points (PEPs) are deployed (see more details in §2.3). While several prominent companies have embraced this new ZT architecture (e.g., Google [63], Microsoft [46], Cloudflare [10]), it has some deployment barriers.

In this paper, we point out and aim to address *three problems* regarding the practical deployment of ZT.

1) ZT's security measures come at a price. The ZT authorization requires *new entities* in the data and the control plane (cf. Fig. 1). As all access requests are authorized and verified by these new entities, *default network routes can change* (hence requiring additional traffic engineering to maintain balanced network utilization), and ZT's verifying entity component can potentially become a bottleneck and/or a victim of a denial-of-service attack.
2) Due to the requirement to upgrade existing devices (e.g., client devices, servers, routers) and extend the architecture with new ones to accommodate ZT, *the overall response time increases*. In particular, the increased number of entities involved in the connection establishment, the increased number of round-trips and potential bottlenecks at the control plane, and computational overhead of additional security measures (e.g., encryption, authentication) can *significantly affect the Time-to-First-Byte* (TTFB[1]). These communication overheads will hinder the adoption of ZT for latency-critical applications.
3) When introducing new security measures, the DNS traffic that precedes every single connection attempt is one of the few traffic classes left intact, i.e., the *DNS (traffic)*, which is unsecured by default, *remains unchanged*. This stems from its critical role; it is required for virtually every single networking entity. Therefore, traditionally, network operators are reluctant to interfere with DNS traffic, and it is allowed to cross the organizational perimeters unencrypted freely, and every component within the network (perimeter) can communicate with the DNS resolver. While several articles (e.g., [16, 29, 41]) raise security concerns about unencrypted DNS and how malicious actors can leverage it (e.g., abusing DNS protocol to exfiltrate data, using it for malware Command & Control communication, DNS hijacking, distributed denial-of-service attacks), solutions only recommend the adoption of encrypted protocols e.g.,DNS-over-TLS (DoT [27]), DNS-over-HTTPS (DoH [26]); not adequately addressing the access control of this critical DNS infrastructure.

Therefore, we propose ZeroDNS, an optimized realization of the ZT architecture. In ZeroDNS, to minimize the impact of *(1)*, the Policy Engine of the control plane components (i.e., PE and PA) is realized in the DNS infrastructure, i.e., when a client wishes to resolve the domain name of a service, the fine-grained authorization is also done by the DNS protocol level. In particular, after authenticating to ZeroDNS (see details below), by piggybacking the DNS responses, the client also receives its specific authorization tokens (e.g., JSON Web Tokens [35]) for the service/enterprise resource[2] whose domain name was queried. Therefore, even with the general inherent overhead of the new authorization components of ZT, ZeroDNS can avoid significant increase in the round-trips, thereby reducing TTFB (and addressing *(2)*).

To address *(3)*, ZeroDNS enforces mutual TLS (mTLS [8, 55]) to protect communication with DNS, ensuring that only authenticated and permitted clients with valid certificates can query domain names. Moreover, ZeroDNS does this without the need for any actual change to the existing DNS infrastructure (see details below).

We implement ZeroDNS on top of `NGINX` [24], a commodity-off-the-shelf (COTS) and publicly available software middle-box solution, originally developed as a HTTP and reverse proxy server. To foster seamless adaptation of the ZT security model (cf. §3.2), both traditional and ZT security components are offloaded to our `NGINX` proxy (cf. §3). In particular, application-layer security mechanisms (i.e., the TLS and mTLS endpoints) for the DNS communication, and the fine-grained access control required by the ZT environment are implemented as `NGINX` plugins using `NGINX JavaScipt` (NJS [50]). Using `NGINX`, ZeroDNS not only optimizes resource utilization and provides load-balancing features but offers a DNS back-end server-agnostic solution. Therefore, ZeroDNS does not require any modification to the existing DNS infrastructure.

We evaluate the performance of ZeroDNS in a simulated environment to measure the latency impact of its inherent computational overhead (cf. §4). In particular, we compare the DNS response time of of a regular client in ZeroDNS to different traditional approaches (i.e., pure unencrypted DNS communications, different

[1]TTFB refers to the time between the client application initiating a connection to a service and when it receives the first byte of relevant information. This metric is usually used to measure the response time of a web server[21].

[2]Throughout the paper, we use the words "service" and "enterprise resource" interchangeably.

encrypted DNS communications through NGINX). We find that on top of NGINX's negligible off-the-shelf load-balancing overhead, invoking NJS scripts during the packet processing (i.e., running ZeroDNS) does not introduce significant additional overhead.

Before concluding our work in §6, we discuss possible limitations of ZeroDNS and potential future works (cf. §5).

2 BACKGROUND AND RELATED WORK

2.1 Traditional Perimeter-based Security Model

The traditional perimeter-based network security approach fragments a network into smaller zones, each protected using firewalls. Some zones might have stricter security controls than others, depending on the resources within that particular zone. For instance, in an enterprise setting, the employees' network traffic might have no practical limitations as they are considered trusted as being behind a firewall and constrained to a physical location (e.g., the company's premises). On the other hand, services deemed riskier, e.g., web and mail servers facing the public internet, are situated in different zones (termed as demilitarized zones or DMZs) with much tighter security controls, i.e., the incoming traffic is tightly monitored, controlled, and potentially filtered. Naturally, once the first-line-of-defense firewalls are breached, or any device inside the trusted zones becomes compromised, lateral movement of the adversaries, accessing and leaking sensitive data due to the lack of adequate security measures and tight access control turn out to be a huge threat.

Despite all these well-known facts, the perimeter-based security approach is still prevalent. This phenomenon can be attributed to several reasons, which, might unintentionally, have even enforced the existence of this broken security model. For instance, as the Internet adoption has grown in late 1990s and the publicly addressable IP addresses appeared to be running out, the IETF defined RFC 1597; three IP network ranges reserved for private networking only, i.e., these IP addresses are not globally routable. Therefore, entities that need Internet connection only for consuming information but not for providing any service, could be placed behind a Network Address Translator (NAT). NATs perform private-to-public IP address translation at network boundaries, i.e., they are used to provide Internet access for the entities behind them but do not let any unsolicited traffic coming in from outside. This approach not only slowed down solving the problem of the limited public IP address space but resulted in more isolated networks. The use (of the term) of private addresses also inherently (but erroneously) meant more secure environments as these networks are incapable of being connected directly to other networks, or to the public Internet.

2.2 Why We Need a Better Model?

Nowadays, the physical location of an enterprise's services cannot be determined accurately, and the definition of the network perimeter has become blurred.

Thanks to increased network speeds, virtualization techniques, software-defined networking (SDN [40]), and network function virtualization (NFV [18]), enterprises have adopted public cloud solutions to benefit from low infrastructure costs, reliable and redundant connections with high availability, flexibility, scalability, and efficient resource provisioning. 5G technologies further fosters this infrastructural expansion. For instance, massive number of (inexpensive) IoT devices can be deployed (e.g., for automation, monitoring, sensing, measuring) and connected to the enterprise network from virtually anywhere without having a dedicated infrastructure maintained by the company. By utilizing these technology advancements, companies can easily create new offices all around the world and federate them to the same enterprise network. The recent COVID-19 pandemic has strengthened the reliance on broadband connectivity and remote working. This has resulted in a paradigm shift in working arrangements; employees can rely on VPN technologies to connect to the enterprise network from anywhere. By further reducing the capital expenditures and allowing BYOD models, employees can even connect to the enterprise network using virtually any kind of device (that are not necessarily monitored and managed closely by the enterprise).

Enterprises can no longer assume that whatever is inside their network is safe because the definition of "inside" has been lost.

2.3 What is Zero Trust?

A network adopting the ZT security model is based on five principles [22]: *(a)* the network is always assumed hostile, *(b)* external, as well as, internal threats perpetually exist, *(c)* the implicit trust in the network imposed by the physical location of a entity is removed, *(d)* every device, and user is authenticated, and authorized, while the corresponding network traffic is encrypted, *(*e) policies governing the operation of the network must be dynamically adjustable, and calculated as many sources of information as possible.

According to NIST's ZT definition [56], ZT architecture is composed of several logical components. As depicted in Fig. 1, the Policy Decision Point (PDP), consisting of the Policy Engine (PE) and Policy Administrator (PA), is situated in the control plane, while the Policy Enforcement Point (PEP) is realized within the data plane. PE is responsible for describing the enterprise policies, while PA implements the policies defined in the PE, e.g., it generates session-specific authorization tokens. Note, even though there is no trust in the physical location anymore, ZT still needs the control plane entities to operate in an environment with strict (physical) access control.

The PEP is responsible for manifesting the policies enforced by the control plane components in the data plane. It enables, monitors, and terminates connections between the clients and the enterprise resources deployed in the trust zone [56]. Note that while the PEP is shown as one logical component in the data plane, it is usually broken into two parts. One at the client's device (e.g., IoT device, employee laptop), termed as PEP-Agent, while the other is at the service (e.g., the web portal, dashboard), termed as PEP-Gateway. These separate entities make ZT possible without modifying the client's and the server application's logic. Put differently, these entities take care of the ZT-related communication and connection establishment to take place transparently to the application layer.

In ZT, when a client wants to access an enterprise resource, it first must resolve the resource's domain name. Then, before connecting, the PEP-Agent at the client communicates with the PDP in the control plane to authenticate and authorize itself in order to access the enterprise resource. Once the PDP has provided the appropriate

authorization token to the client, it can initiate the connection to the desired enterprise resource. The counterpart of the PEP at the enterprise resource, i.e., the PEP-Gateway, verifies[3] the tokens provided by the client and grants (or denies) access accordingly. This process ensures that the client has obtained its token in a legitimate way, the token is indeed generated for the client (i.e., to avoid any malicious actor to obtain and use authorization tokens from legitimate clients by any means, like stolen credentials), and whether it is still valid. Finally, the actual connection establishment occurs like in a traditional network. Clearly, the increased security measure imposed by the ZT entities will have significant impact on the TTFB.

With ZeroDNS, our aim is to simplify the authorization process by implementing/offloading most of the ZT procedures (e.g., token issuance, access-control within/to the DNS infrastructure. We implement the control plane elements on top a `NGINX` proxy that, besides load-balancing the requests to the DNS back-end servers, can also intercept each DNS response and extend it with the appropriate authorization tokens (see details later in §3). Therefore, clients' PEP-Agents become aware of whether they are authorized to connect to a service straight away when resolving its domain name, thereby significantly reducing the TTFB.

Note, ZT principles and policies (e.g., maintenance of authorization tokens) are expected to be adopted by every enterprise in a customized manner. ZeroDNS optimizes ZT's underlying architecture in a platform-agnostic manner; the actual authentication and authorization functions (e.g., PA, PEP) are out of scope of ZeroDNS.

2.4 Domain Name System

Every communication attempt to a resource requires obtaining the appropriate IP addresses first. This process is done through the decentralized Domain Name System (DNS, [47]) protocol. Every domain is maintained by the authoritative name server of the enterprise registering it. To avoid a massive amount of devices connected to the Internet overloading a single domain's *authoritative server* before visiting, DNS employs a hierarchical structure. In this structure, *recursive resolvers*[4] (typically run by big companies like Google) act as high-performing global "phone books" providing domain name resolution services (for everyone) with all domain records queried and cached so far. Similarly, every client also caches resolved domain data in its local *stub resolver* (e.g., at the operating system, in the web browser) to avoid additional DNS resolution processes for the same recently visited domains. Due to the dynamically changing nature of the Internet, however, every resource record has a relatively short (typically, an hour- to a day-long) expiration time (i.e., time to leave or TTL) for which the record is considered valid, hence cached. Once the TTL of a record expires, the domain has to be resolved again.

A client can be provided with several (recursive) DNS resolvers; it will query a domain according to the resolvers' priorities until the domain is resolved. If the domain cannot be resolved, the client receives an `NXDOMAIN` error (i.e., a non-existent domain as a reply); hence, it will not be able to connect to the resource using its domain name.

Since almost every resource is identified by domain names (instead of hard-to-remember IP addresses), DNS has become a critical infrastructure since its inception. However, it has been a plaintext protocol for all practical purposes [13], allowing malicious actors to track user activities (e.g., [37]), tamper with DNS (e.g., [9]), or even block certain queries for censoring purposes (e.g., in [7]). While some recent privacy-enhancing technologies (e.g., DNS-over-TLS [27], DNS-over-HTTPS [26]) have started to penetrate particular DNS ecosystems, they only provide encrypted communication. Fine-grained access control of DNS, similarly to other resources in ZT (i.e., who can resolve which domains), is still missing. With ZeroDNS, we make a step towards filling this gap.

3 ZERODNS ARCHITECTURE

Our proposed architecture is shown in (the top-most part of) Fig. 2. On the left-hand side, the existing DNS infrastructure is shown. According to the IANA considerations [28], a primary and a secondary DNS server are deployed. Both are realized (and configured) by the stock BIND9 DNS implementation [31]. These DNS back-end servers are the authoritative name servers of the enterprise's network.

In the middle, the ZeroDNS plugin is shown, which is implemented on top of `NGINX` reverse proxy using NJS, an `NGINX` specific Javascript engine [50]. ZeroDNS can be easily added to existing infrastructures already deploying `NGINX` proxies for other purposes, e.g., load-balancing, scalability, traffic engineering, logging, TLS offloading/termination [24][5].

The reverse proxy is configured to provide load-balancing capabilities for the DNS servers, i.e., DNS queries are evenly distributed among the back-end servers by relying on NGINX's built-in load-balancing algorithm. It also offers secure TLS tunnel endpoints for connecting clients (refer to the right-hand side of the figure). In particular, it provides standard DoT and DoH endpoints. For the latter one, a translation service is utilized [6] to "convert" DNS messages arriving via HTTP to traditional DNS wire format and vice versa. Furthermore, these endpoints are protected using mutual TLS, i.e., only authorized clients with appropriate X.509 certificates can connect to the reverse proxy to query a domain.

By these features, ZeroDNS provides and *end-to-end encryption* between the clients and the DNS proxy. Note that terminating (m)TLS sessions at the reverse proxy is a typical practice when deploying NGINX, especially when deployed for load-balancing purposes. Therefore, NGINX reverse proxy enables swift and transparent scaling-out of DNS back-end servers without dealing with multiple certificates, TLS session handling, etc. It also means that the back-end servers are configured to accept queries from the proxy *only*; Furthermore, since the type of communication between the NGINX proxy and the DNS back-end servers remains intact, ZeroDNS fosters brownfield deployment. To avoid imposing any security concern (e.g., man-in-the-middle attack, DNS poisoning),

[3] The verification process is implementation specific. Adequate verification information can be self-contained in the authorization token (e.g., with signed JWT token) or the PEP-Gateway might double-check the validity of the tokens with the PDP (e.g., via Online Certificate Status Protocol [25, 57]

[4] Note, an authoritative name server acts as a recursive name server for its clients requesting domains out of the authoritative domain.

[5] In environments without pre-existing `NGINX` modules, it can be easily installed (even in a Docker container [51]) and configured within a couple of steps [17].

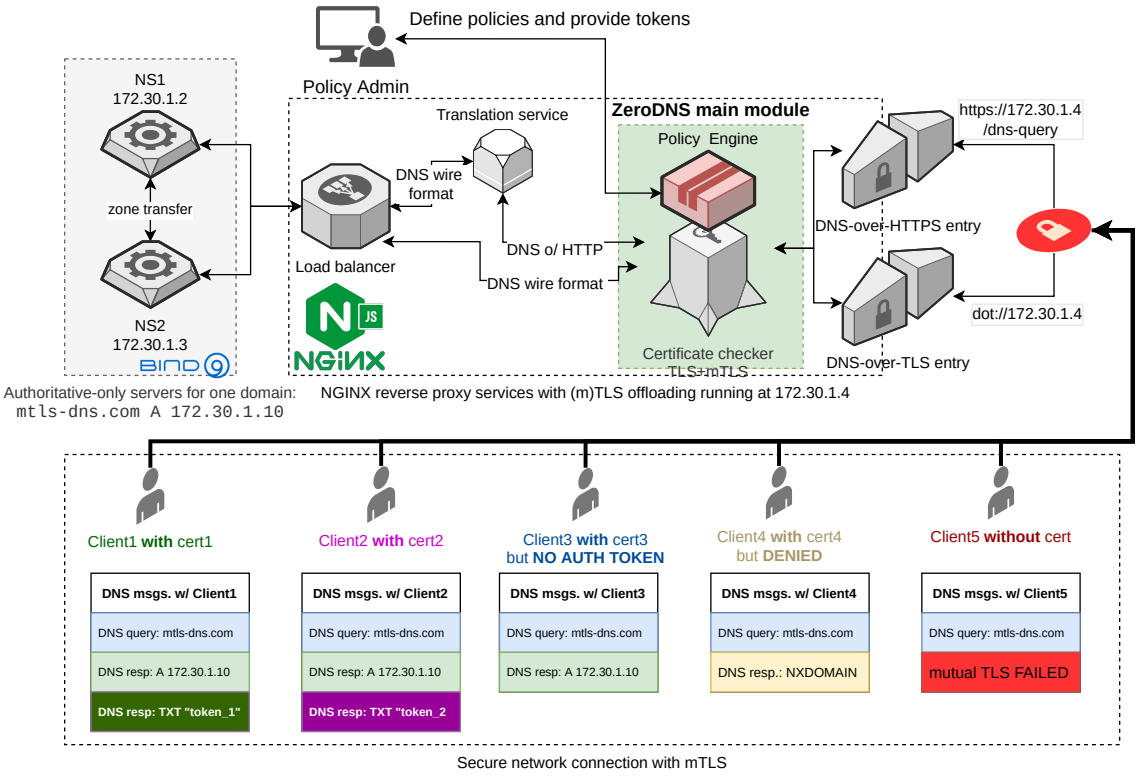


Figure 2: Architecture

ZeroDNS runs physically close (i.e., directly connected to) or on the primary back-end server.

The core of ZeroDNS is the main module denoted in the green box. The steps are the following.

1. The requesting client initiates a TLS connection towards ZeroDNS and authenticates itself via its X.509 certificate.
2. Then, the PE component of the main module fetches the corresponding policy, optionally caches it for future use, and in parallel, forwards the DNS query towards one of the DNS back-end servers to avoid any significant packet buffering at the proxy side, i.e., to avoid keeping the DNS query in the buffer until the policy lookup is done.
3. Once the corresponding DNS response from the DNS back-end servers is received by the proxy, the PE component of the main module evaluates whether the requester, besides possessing the proper client certificate, is granted permission to resolve the domain name (of a requested service).
4. After successful authorization, the DNS response is appended with the client-specific authorization tokens and sent to the client.
5. The client receives its custom authorization tokens straightaway after resolving the requested service's IP address.

Note that the authorization tokens for each client are generated and constantly updated by the PE and PA. The DNS proxy has access to this list of tokens wherein it can look up whether the client querying a given service's domain is authorized to visit.

Time-To-Live (TTL): Since the clients have to obtain the IP address of the requested service via DNS before every connection attempt, piggybacking DNS can reduce the TTFB between a client and a service. However, DNS records typically last longer than authorization tokens, i.e., authorization tokens may expire earlier than the DNS records[6]. If the DNS TTL is longer than the ZT authorization token's TTL, clients will have an IP address in their caches (i.e., in their stub resolver), but the authorization tokens would have already expired. In ZeroDNS, the DNS records' TTL is set (by the proxy when processing the corresponding DNS reply) to the lowest TTL value of the authorization tokens. This way, the clients have to query the given service's domain IP address via DNS every time the authorization tokens expire. Recall, this does not need any change to the DNS back-end servers or their fundamental logic; the TTL values are updated by the DNS proxy.

Authorization token TTL is set based on the ZT access policy, which is context dependent. If an authorization token expires, the client must obtain a fresh token (if it still desires access to the enterprise resource). Since we use ZeroDNS for both DNS resolution and as the policy decision engine, setting the DNS TTL to the ZT authorization token TTL will not impose overall additional communication costs. Naturally, however, the ZT-related communication now introduces additional overhead to the DNS infrastructure yet,

[6]The typical default TTL value for a DNS record is 12 hours or 24 hours [30], while authorization tokens in ZT last for minutes or even less for more critical resources

due to the load-balancing feature of NGINX, seamless scaling-out of the DNS back-end servers is straightforward.

Note, however, TTL values in the DNS responses are sometimes not honored by client applications, e.g., browsers might cache DNS responses at least for hard-coded timeframe, such as 15 minutes [48]. When such applications are required to access enterprise resources in a ZT environment using ZeroDNS, authorization tokens should either be set accordingly if acceptable (i.e., greater than or equal to this value), or seamless fallback to traditional ZT-based authentication methods should be provided.

3.1 Workflow

We give an overview of the steps taken when a client connects to a service available at a hypothetical domain, `https://www.mtls-dns.com`. There are different client types, shown at the bottom of Fig. 2.

1. Consider `Client1`, which has a client certificate issued and signed by the organization. Using one of the secured DNS end-points (i.e., DoT or DoH), `Client1` first sends a DNS query to resolve the corresponding service's domain's IP address.
2. After the client has verified the DNS proxy by the DNS proxy's certificate (signed by a trusted authority) and authenticated itself using its client certificate, the DNS proxy forwards the query to one of the DNS back-end servers.
3. While the DNS proxy awaits the response from the DNS back-end server, the proxy checks the provided policy (i.e., the list of tokens, access rights of the client) to see whether the client is allowed to query the domain name and whether there is any authorization token available for the querying client. In our preliminary implementation, for brevity, the list of tokens and the access policies are stored in a JSON file, which is read and loaded by the DNS proxy[7].
4. If `Client1` is authorized to use the service at `https://www.mtls-dns.com`, then the client-specific token, `token_1`, is appended to the DNS response `ANSWER` section as a `TXT` record[8]. The case for `Client2` depicts a similar picture except that it receives a different token, `token_2`. Suppose, for a client, there is no authorization token[9], e.g., in the case of `Client3`. In that case, the corresponding DNS response remains intact, hence the client only receives the original DNS response with the queried relevant information (e.g., `A` record with the IPv4 address of the service). Therefore, the client will need additional authentication and authorization to access the request service.
5. With `Client4`, it has the correct certificate, however the DNS reply is denied due to finer-grained access policies defined in the PE, i.e., `Client4` is not authorized or temporarily blocked. In particular, when the response from one of the DNS back-end servers is received (with the correct `A` record), the proxy alters it, and replace the `ANSWER` section with `NXDOMAIN`. Therefore, `Client4` will not be able to even connect to the service because it does not have the correct IP address.
6. Last but not least, in the case of `Client5`, since it has no appropriate client certificate, its request is denied straightaway as the mutual authentication fails.

[7]Thus, when the tokens are refreshed, an explicit notification is needed to be sent to the DNS proxy to reload the JSON file.

[8]Note, DNS has several other sections (e.g., auth. records, additional records), and possibilities to append the authorization tokens to (e.g., EDNS). For simplicity, we relied on the `ANSWER` section and `TXT` type.

[9]Assume a new service is under development, and to avoid interfering with the actual running infrastructure, access control is merely handled by the service itself.

3.2 Benefits of ZeroDNS

Next, we discuss on several benefits ZeroDNS provides.

Minimal modification to existing network infrastructure: In ZeroDNS, the zero trust control plane components, e.g., PA, PE, are introduced as plug-ins to the existing DNS infrastructure. The DNS communication, which precedes every connection, is used to piggyback client-specific authorization tokens. Thus, clients do not need to make a separate request to the PA to obtain a ZT authorization token. This also eliminates the need to have the PA as an additional stand-alone server.

By building on top of the ubiquitously used `NGINX` reverse proxy (cf. §3), not only are the original DNS services preserved, but also no additional change is required at the protocol level (i.e., ZeroDNS does not require any new protocol to be deployed).

Reduced ZT bottleneck: Our ZeroDNS plugin acts as a reverse DNS proxy. In particular, the reverse proxy behaves as an intermediate server that, by intercepting client requests, can forward them to the appropriate back-end servers (see implementation details in §3). Since, every domain requires at least two authoritative name servers according to the IANA technical requirements[28], a reverse proxy is useful to balance the load across these DNS back-end servers.

By utilizing the load-balancing capabilities of `NGINX` in ZeroDNS, we can optimize resource utilization, maximize throughput, reduce latency, defend against DoS attacks, and provide transparent fallback mechanisms for the DNS back-end [17, 24, 38].

Being true to ZT: In ZT, authentication and authorization are critical building blocks. However, existing DNS implements none of them. Anyone making a request, receives the IP address of the resolved domain name.

In ZeroDNS, requesting clients can only resolve domain names if the clients authenticate themselves. X.509 certificates are issued and deployed to each client enterprise-wide, and connections to the DNS proxy are protected using mTLS. Traditional ZT requires devices to be authenticated, so we use the same identity management. Access control is defined at the DNS proxy to check whether a client is allowed to resolve the domain name of a particular service.

Offloading TLS processing: Besides enforcing strong authentication and authorization, ZT requires each communication to be encrypted. While the security and the privacy aspects of the Domain Name System have received significant attention lately [11, 15, 26, 27, 53, 54], encrypted DNS request-response, especially within an enterprise setting, is not yet a reality. Aside from some disputed trade-offs between increased privacy and efficient network monitoring [33], technically adopting encryption to the DNS traffic can be challenging in an existing infrastructure. For instance, while `BIND` [31] and `Unbound` [52] support DNS-over-TLS (DoT, [27]) since 2018 [3, 32], `Simple DNS plus`, a resolving and authoritative DNS

server for Windows-based systems, supports DoT only since the end of 2021 [34]. Furthermore, if a network administrator prefers DNS-over-HTTPS (DoH, [26]), the number of compatible native implementation choices reduces even more.

Our DNS proxy implements typical application-level encryption using TLS, thereby offloading such security-related functions to the proxy and leaving the existing DNS infrastructure intact.

DNS server implementation-agnostic: Besides preserving existing infrastructure elements, it is beneficial if critical software components, e.g., the DNS back-end server, remain unchanged.

By offloading the TLS processing to our DNS proxy where all ZT components are realized, the existing, potential non-encrypted, DNS back-end servers do not need any upgrade; they can still operate according to the original infrastructure. Furthermore, since essential security mechanisms required by DoT or even DoH are provided by the DNS proxy, ZeroDNS is entirely DNS back-end server-agnostic. In other words, ZeroDNS only requires to have either encrypted (e.g., DoT) or traditional non-encrypted (i.e., DNS-over-UDP) access to the (authoritative) domain name servers irrespective of any implementation- and platform-specific details.

Piggybacking existing traffic: Introducing new servers with new protocols can quickly increase an enterprise's overall costs. In particular, migrating a traditional network into a ZT and introducing new entities to the network infrastructure (i.e., PA, PE, PEP), will cause additional network administration challenges, e.g., different access control lists, intra-domain routing changes, traffic re-directions. Furthermore, a new service (running on a new physical machine) increases the number of entities that can fail or be compromised, which eventually further increases the maintenance and management costs.

In ZeroDNS, by utilizing and extending (existing) `NGINX` proxy functions, we do not introduce any new physical or logical control plane entity, hence no additional network routing management is required. Furthermore, by piggybacking DNS traffic, network routes do not require any alteration, and troubleshooting (i.e., finding a root cause of a networking-related failure w.r.t. ZeroDNS) does not impose much additional efforts either. ZeroDNS, however, just like any ZT realization, requires the PEPs to assure the secure and authenticated connections (cf. §2.3).

Reduced Time-to-First-Byte (TTFB): TTFB refers to the time between the client application initiating a connection to a requested service and when the client receives the first byte of relevant information. Every connection starts with a DNS lookup, i.e., with the process of resolving the service's domain name. This connection imposes at least one RTT (Round-Trip Time) if made over unencrypted UDP channels, and a few more RTTs with increased reliability (e.g., over TCP) and additional security measures (e.g., over TLS). Let t_{UDP}^{DNS} denote the time required for domain name resolutions via unencrypted DNS communication, i.e., via DNS-over-UDP. Since in ZT environment, all services must rely on strong authentication, let, furthermore, t_{TCP}^{DNS}, t_{TLS}^{DNS}, and t_{mTLS}^{DNS} denote the elapsed time when domain name resolution is over TCP, over TLS, and subsequently, over mutual TLS (proposed by our ZeroDNS architecture), respectively[10].

[10]Note, the latter ones inherently requires all previous ones, e.g., mTLS connection requires TLS on top of TCP.

Traditionally, after resolving the domain name, the client initiates a TCP handshake to the service's resolved IP address, then a TLS handshake takes place to kick off a secure communication session. Additionally, mutual TLS might be enforced by the requested service. This multi-step connection establishment to the service is necessary regardless of whether the security model is based on ZT or not. Therefore, let us collectively refer to those steps as $t_{ALL}^{Service}$.

In a traditional network where *only* the content-hosting servers are protected and authenticated by TLS (and optionally by mTLS), the TTFB can be summarized as follows:

$$TTFB_{trad.} = t_{UDP}^{DNS} + t_{ALL}^{Service} \quad (1)$$

In the case of secured DNS communication according to recent standards [26, 27] and best practices [5, 14, 20], the above equation changes as follows:

$$TTFB_{trad.} = t_{TCP}^{DNS} + t_{TLS}^{DNS} + t_{ALL}^{Service} \quad (2)$$

In a zero trust environment, accessing any service is expected to require strong authentication, as well as authorization through the new control plane components (i.e., PA, PE), therefore, clients have to make additional round-trips increasing overall TTFB as follows:

$$TTFB_{zt} = t_{TCP}^{DNS} + t_{TLS}^{DNS} + t_{mTLS}^{DNS} + t_{TCP}^{ZTCP} + t_{TLS}^{ZTCP} + t_{mTLS}^{ZTCP} + t_{ALL}^{Service}, \quad (3)$$

where *ZTCP* refers to Zero Trust Control Plane.

In ZeroDNS, however, by piggybacking the DNS responses with the ZT authorization tokens, TTFB can be reduced close to the TTFB in traditional networks that have much looser security measures in general (i.e., to $TTFB_{trad}$):

$$t_{\text{ZeroDNS}} = t_{TCP}^{DNS} + t_{TLS}^{DNS} + t_{mTLS}^{DNS} + t_{ALL}^{Service} \quad (4)$$

Note, if traditional networks advocates strong authentication via mTLS for DNS communication, the TTFB of the ZT architecture via ZeroDNS is practically the same. For comprehensive sequence charts of the above-mentioned steps, refer to Fig. 5 and Fig. 6 in the Appendix (cf. §6).

4 EVALUATION

ZeroDNS provides Zero Trust access control leveraging the DNS infrastructure, while reducing connection establishment time by reducing the total number of round-trips required. However, the processing time at the proxy might offset the gains. In this section, through experiments in a simulated environments, we show that this processing time does not introduce additional significant time costs.

4.1 Methodology

ZeroDNS adds mTLS to DNS and uses the DNS for certain ZT control plane functions. In our measurements, we focus on the round-trip times of DNS and investigate to what extent ZeroDNS affects the overall DNS response times.

We simulate one client, which uses the `kdig` [39] command-line utility to resolve a domain name and we rely on the same tool's reported response time. Note, due to the nature of the command-line application, every time it is issued, the (m)TLS connection is established from scratch and the connect torn down once the response

is received. This helps us to measure the worst-case baseline indicators; any optimization, e.g., TLS session tickets [42], 0-RTT [58], would only affect the performance positively. The measurements are carried out in a lightweight containerized environment, where each entity (two stock authoritative BIND9 name servers[11], one stock `NGINX` proxy[12] with the ZeroDNS extensions, and one regular Debian Linux-based client[13]) runs in a separate Docker container and the underlying network substrate is provisioned by the default Linux networking stack (i.e., virtual Ethernet pairs [45], Linux bridge utilities [43], iptables [44]) Docker relies on. This containerized overlay network runs on top of a Ubuntu 20.04 LTS operating system, running on a relatively old Lenovo x240 Laptop with Intel Core i7-4600U CPU with 8GB of memory.

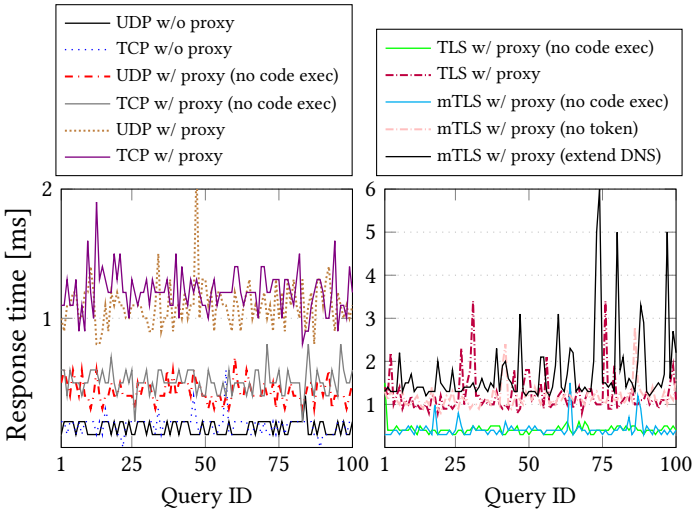
We differentiate several use cases based on the protocols relied on and the additional processing time they incur. `UDP w/o proxy` and `TCP w/o proxy` imply that the client directly sends the queries to one of the DNS back-end servers using UDP and TCP protocol, respectively. The protocols with the notion `w/proxy (no code exec)` represent response times using the given protocol when queries are sent to the `NGINX` reverse proxy; however, *no code execution* is involved. This is a typical scenario when the reverse proxy is used for load-balancing purposes only. In contrast, the protocols (e.g., `UDP, TCP, TLS`) denoted solely by `w/proxy` mean that the proxy also executes Javascript instructions for every query and response even if no packet mangling occurred, i.e., when queries and responses are parsed. Since the authorization tokens are based on the clients' certificates, we differentiate three different use cases for the mTLS protocol. Similar to previous cases, `mTLS (no code exec)` shows the response times when the proxy provides only a load-balancing feature on top of mutual TLS communication. In both `mTLS (no tokens)` and `mTLS (extend DNS)`, Javascript instructions are executed for every query and response. However, the DNS response packets are extended in the former case, as the connecting client will receive no authorization token(s). The latter case (`mTLS (extend DNS)`) represents the full-fledged scenario in ZeroDNS, where clients authenticate themselves via mutual TLS and, in response, the proxy extends the relevant DNS queries with the client-specific authorization token(s).

Our experiments measure the absolute and relative response times for 100 consecutive DNS queries sent from the client using all different protocols. For relative comparisons, if not stated otherwise, we take the average response time of the 100 measurement points. The y axes show the measured/averaged response times in milliseconds.

4.2 Baseline Measurements

To measure additional latency imposed by the proxy, we need baseline measurements. The baseline measurement represents the traditional use case when the DNS (back-end) servers are directly addressed (i.e., there is no additional proxy), and there is no communication overhead either (i.e., `UDP w/o proxy`). Then, we compare this baseline to the response times of other protocols in the order

[11] https://hub.docker.com/r/internetsystemsconsortium/bind9
[12] https://hub.docker.com/_/nginx
[13] https://hub.docker.com/r/cslev/debian_networking



(a) With and without proxy when no encryption is used.

(b) With proxy and encryption.

Figure 3: Absolute responses times for 100 queries using different protocols. Note the different scales of the y axes.

of overhead they "naturally" impose according to their additional complexity (cf. §4.1).

The results are depicted in Fig. 3. We observe that while TCP introduces at least two more round-trips due to the TCP handshake (i.e., TCP SYN/SYN-ACK/ACK), on average, the use of TCP does not introduce significant overhead to the response time compared to using UDP ($\sim 0.15ms$), providing an average response time of $0.175ms$. When we introduce the reverse proxy, the response times increase up to $\sim 0.433ms$ with UDP, and $\sim 0.5ms$ with TCP, respectively. When the proxy processes each packet, i.e., when the Javascript code is executed every time a query or response is forwarded, on average, the response time with UDP increases to $\sim 1.1ms$, and with TCP to $\sim 1.2ms$.

When it comes to encryption, we observe an unexpected phenomenon in the response times measured. The average response time with `TLS w/ proxy (no code exec)` is lower ($0.419ms$) than the same scenario without encryption, i.e., `TCP w/proxy (no code exec)`. When, in addition to the TLS encryption, the proxy also processes the DNS packets (i.e., `TLS w/ proxy`) the average response time increases to $1.21ms$; slightly above the similar case without encryption, i.e., above the response time measured in the case of `TCP w/ proxy`. While on average, the response time is higher with mutual TLS authentication, compared to sole TLS authentication, the increase is negligible ($0.419ms$ of TLS compared to $0.421ms$ of mTLS).

We consider the full-fledged scenarios, i.e., when using mTLS connection and code execution at the proxy side. In particular, when the connecting client is not (directly) authorized to the requested service, i.e., no tokens are added to the DNS response, the average response time becomes $1.116ms$. On the other hand, when DNS responses need to be extended with authorization information, the average response time turns out to be $1.72ms$.

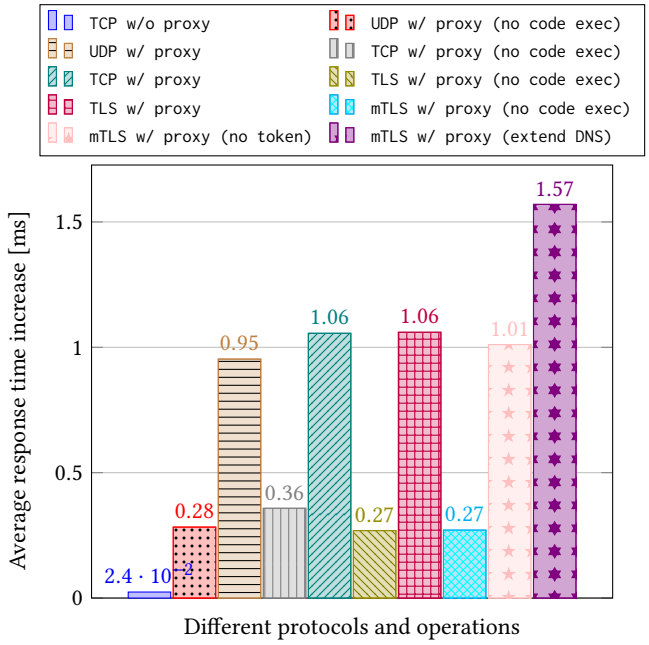


Figure 4: The average response times of all settings relative to the baseline.

We can conclude that the greatest impact on the response time, as expected, is when the reverse proxy has to process every DNS packet. However, the whole domain name resolution with the DNS response packet extension with authorization tokens requires, on average, no more than $2ms$.

4.3 Relative Response Times

Since the absolute response times heavily rely on the existing network services, comparing the response times of all protocols relative to the baseline scenario can provide better, infrastructure-independent comparison. To this end, the average response time of the 100 queries for `UDP w/o proxy`, i.e., $0.151ms$, is compared to the average response times of all other protocols. Therefore, Fig. 4 shows the deviance in the response time of the different protocols (i.e., the additional latency the other protocols impose).

We can observe that, on average, the introduction of the proxy element imposes around a $0.26 - 0.36ms$ penalty on the response time. The results also indicate that whether the proxy provides no (`UDP w/ proxy (no code exec)`, `TCP w/ proxy (no code exec)`), TLS (`TLS w/ proxy (no code exec)`), or mTLS (`mTLS w/ proxy (no code exec)`) security measures, it does not play a significant role; interestingly, the average indicators turn out to be slightly better for the encrypted cases[14]. Furthermore, one can see that when the proxy load-balances the traffic and processes each DNS packet *at the same time*, the response times for all protocols increase, on average, by about $1ms$. When the DNS responses require on-the-fly modifications and extensions with client-specific authorization tokens, the average response time further increases by $0.5ms$.

We can conclude that while introducing the DNS proxy element naturally imposes additional latency, it only adds at most $\sim 1.5ms$ latency to the average DNS response time for the client. Many enterprises already deploy (reverse) proxies for load-balancing purposes. Therefore, above-mentioned factor may further decrease by the average response time of the `w/ proxy (no code exec)` use cases resulting in an average additional latency of the ZeroDNS architecture of only $\sim 1.2ms$. Finally, since in ZeroDNS, client-specific authorization tokens are piggybacked in the DNS response, the additional latency will be mitigated by having significantly lesser number of additional round-trips.

5 DISCUSSION AND FUTURE WORK

In this section, we discuss further challenges and considerations for future work regarding ZeroDNS.

Client and Server Applications: In the ZT security model, the policies are enforced by the Policy Enforcement Points (PEP), deployed in the data plane (cf. §2.3). Practically speaking, these PEP points are deployed on the client systems (PEP-Agent), as well as on the servers (PEP-Gateway). These logical entities are responsible to carry out the authorization required by ZT. This provides a seamless migration to ZT as neither the client nor the server application requires any modification to their business logic. In ZeroDNS, we do not change the data plane and consequently, the functionality of the PEPs. ZeroDNS still relies on the existence of the PEPs; it only advocates the authentication and authorization to be done on the DNS protocol layer at the same time as the DNS query and response.

Multiple services behind the same IP: Different services reachable at different endpoints (e.g., REST API endpoints) may have different authorization tokens according to different access policies. However, they might still run behind the same IP address. If for one domain name, the DNS proxy supplies all tokens for all services for a given client, the DNS response might become too big (in packet size), resulting in an undesired DNS packet fragmentation. While fragmentation does not impose any significant security issues or processing overhead at the DNS proxy, DNS fragments should be avoided. Therefore, it is worth considering whether there is any straightforward way to provide extra information in the DNS query to always receive only the relevant authorization tokens to a given (set of) endpoint. EDNS0 extensions [62] may suit this demand. In EDNS, one optional resource record (termed as `OPT RR`) can be added to the request [62]. EDNS extensions have already been used for several purposes, e.g., specifying the UDP DNS payload size of the requester to avoid packet fragmentation, EDNS Client Subnet [12] (proposed in 2011) option used to optimize routing decisions when queries sent to a distant publicly available recursive DNS resolvers. Since the OPT RR supports a variable set of options expressed as `{attribute, value}` pairs [62], the given endpoint(s) in ZeroDNS could be encoded into this field and the DNS proxy can be amended to append authorization tokens accordingly.

Tokens in `TXT` records: We use the DNS `TXT` records to deliver the authorization tokens. While the maximum length of the DNS `TXT` records is 255 characters, typical authorization tokens (e.g., JWT

[14] When a fine-grained measurement is done in an environment completely running within one laptop only, such small (unexpected) deviance often occurs and is attributed to measurement error.

tokens) can exceed this length. In ZeroDNS, every token starts with an easily identifiable `prefix` as a (`tokentype::webservice->`) tuple. When the total length exceeds ($255-prefix_{length}$) characters, it is broken into multiple DNS `TXT` records (cf. Listing 1 in the Appendix). To ease parsing at the client-side, these subsequent segments of the same token do not have the prefix anymore. Hence, the client application can easily identify the start and the end of the authorization tokens.

Authoritative responses: By adding cryptographic signatures to DNS records, DNSSEC [64] can be used by clients to verify that a response comes from its authoritative name server and has not been tampered with. Clearly, modifying such responses in ZeroDNS (by adding tokens and manipulating the TTL) would result in failed signature verification. However, in ZeroDNS, the responses do not require the existence of DNSSEC. The security extensions in ZeroDNS (e.g., TLS endpoints) adequately verify the authenticity of the DNS server, i.e., the NGINX proxy, and provide end-to-end encryption to the communication channel. Moreover, the clients query the authoritative name servers, i.e., the NGINX proxy, directly, having no man-in-the-middle (e.g., a recursive resolver) involved. On the other hand, if a malicious actor would try to divert the DNS queries to a rouge server, due to the (lack of) the correct TLS certificate, the client will abort and the resolution will fail.

Note, for non-enterprise domains, ZeroDNS is not responsible for any ZT-related measures (i.e., authorization tokens). Hence, those queries can either be handled as usual, i.e., ZeroDNS acting as a recursive resolver (cf. §2.4), or clients can rely on additional third-party recursive resolvers for the same purpose.

Security consideration for shared IP addresses: Client systems do not have to rely on the DNS server (preset via the network administrator) for domain name resolution. They can be manually configured, and hostnames can be matched with IP addresses manually (e.g., `/etc/hosts` file in Linux systems). In general, services are usually deployed behind a gateway that offers several further services (e.g., DDoS protection, load-balancing). Such services can also provide an API gateway (e.g., Apigee [23], Kong [19]) that can process authorization tokens from the header of the requests. In ZeroDNS, even if the IP addresses are obtained from other sources than the ZeroDNS proxy, without the authorization tokens no client can connect to the corresponding service.

Bypassing the proxy: By definition, the proxy can be bypassed. For instance, in case of Fig. 2, if `Client4` figures out the DNS back-end servers IP addresses (e.g., `172.30.1.3`), modifies its network settings, it can bypass ZeroDNS proxy (located at `172.30.1.4`) and directly query the back-end servers. To avoid this, the DNS back-end servers' configuration must be modified to only allow queries from the ZeroDNS proxy's IP address.

Performance: If EDNS can be used to indicate which particular service(s) a client wishes to connect to, the DNS responses could be assured to not go beyond the allowed payload size limit, thereby avoiding more than one response packet received by the client. In certain circumstances (e.g., where not many services are deployed behind one IP address), receiving the authorization tokens for all of the services can improve the performance as no further query is needed for the other services.

Reverse proxy and `NGINX`: As mentioned in §3, ZeroDNS is implemented on top of `NGINX` reverse proxy. Originally, `NGINX` was developed as a web server in 2004 to resolve the so-called performance-related C10K problem [36]. Since then, due its high performance, `NGINX` has evolved to be a reverse proxy, HTTP cache, and load balancer and it is one of the most popular web servers among high-traffic websites [38]. In addition to its superior performance, ZeroDNS uses `NGINX` because its Application Programming Interface (API) allows developers to extend its functionality via NJS, a JavaScript engine specifically tailored to `NGINX`.

However, `NGINX` is not the only way to implement ZeroDNS. HAProxy [4] is another free versatile reverse proxy, offering high availability, load-balancing and proxying features. It support Lua scripting language to extend its basic packet processing capabilities [49], hence being another candidate to materialize ZeroDNS.

Similarly, Traefik [59] is another reverse proxy implementation that can automatically discover the correct configuration for the running services and supports every major cluster technology. From the perspective of ZeroDNS, Traefik can be extended with plugins written in Go language [60].

6 CONCLUSION

Zero Trust (ZT) security framework addresses many long-standing security issues of traditional perimeter-based network security. ZT removes the implicit trust in the network and requires strong authentication, authorization, and encryption for each connection, irrespective of the physical location of the entities.

In this paper, we address three problems of ZT. First problem is that the DNS infrastructure, which is the critical entity in every network, does not adhere to the ZT principles, i.e., anyone can access the DNS and get a response. Second problem is that ZT's authorization procedures require new entities in the data and the control plane to authorize and verify access requests. These new entities can result in changes in preferred network routes (hence requiring additional traffic engineering), as well as introduce potential bottlenecks. Third problem is that ZT adds additional time cost, increasing the time to first byte (TTFB).

We have proposed ZeroDNS, a better realization of ZT. In ZeroDNS, the authorization and authentication entities are realized and enforced within the DNS infrastructure. This also reduces the required number of round-trips, potentially decreasing TTFB. To include the robust security measures of ZT to DNS, ZeroDNS enforces mutual TLS protocol for the DNS communication for authentication, and ensures that only permitted clients with valid certificates can query domain names. We have implemented ZeroDNS on top of `NGINX` proxy, typically used as a load-balancer in enterprise settings, providing a DNS back-end server-agnostic solution. We have evaluated the impact of ZeroDNS on the response times in a simulated environment, and we have found that the impact is minimal.

REFERENCES

[1] A10 Networks. [Accessed: Sep 2022]. What is IPv4 Exhaustion? Online Glossary of Terms, https://www.a10networks.com/glossary/what-is-ipv4-exhaustion/.

[2] A10 Networks. Jun 2022 [Accessed: Sep 2022]. Enterprise Perspectives 2022: Zero Trust, Cloud, and Remote Work Drive Digital Resiliency. Enterprise report, https://links.a10networks.com/NDE3LUdVQi05OTUAAAGE_zvj7YLRpVIB8AdLlqEceIT3UZEVlXclrkm6DG8ii00cR609-QZ5G5TQRu29igpHR0x8lwM=.

[3] Daniel Aleksandersen. Jun 2018 [Accessed: Sep 2022]. Actually secure DNS over TLS in Unbound. Ctrl.blog, https://www.ctrl.blog/entry/unbound-tls-forwarding.html.

[4] Mitchell Anicas. May 2014 [Acceessed: May 2022]. An Introduction to HAProxy and Load Balancing Concepts. Digital Ocean Tutorial, https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts.

[5] Jim Black. Jan 2022 [Accessed: Sep 2022]. We Need to Encrypt DNS: Here's Another Compelling Reason Why. Akamai Blogpost, https://www.akamai.com/blog/security/we-need-to-encrypt-dns-heres-another-compelling-reason-why.

[6] Mark Boddington. Feb 2022 [Accessed: Sep 2022]. Using NGINX as a DoT or DoH Gateway. NGINX blog post, https://www.nginx.com/blog/using-nginx-as-dot-doh-gateway/.

[7] Heather Brown, Emily Guskin, and Amy Mitchell. Nov 2012 [Accessed: Oct 2020]. The Role of Social Media in the Arab Uprisings. Pew Research Center Journalism & Media, https://www.journalism.org/2012/11/28/role-social-media-arab-uprisings/.

[8] Cloudflare. [Accessed: Jun 2022]. What is mutual TLS (mTLS)? Online, https://www.cloudflare.com/en-gb/learning/access-management/what-is-mutual-tls/.

[9] Cloudflare. [Accessed: Oct 2020]. What is DNS cache poisoning? | DNS spoofing. [Online], https://www.cloudflare.com/learning/dns/dns-cache-poisoning/.

[10] Cloudflare. Dec 2021 [Accessed: Sep 2022]. Cloudflare One. At a glance brochure, https://www.cloudflare.com/static/e9ea5dfaa69c554cc1cbaa7f3e441acf/Cloudflare_One_at_a_glance.pdf.

[11] Comcast. Jun 2020. Xfinity Internet Joins Firefox's Recursive Resolver Program, Committing to Customer Privacy Protection. Press release, https://corporate.comcast.com/press/releases/comcast-xfinity-internet-firefox-trusted-recursive-resolver-program-customer-privacy.

[12] C. Contavalli, W. van der Gaast, D. Lawrence, and W. Kumari. 2016. *Client Subnet in DNS Queries.* RFC 7871. RFC Editor.

[13] Levente Csikor, Himanshu Singh, Min Suk Kang, and Dinil Mon Divakaran. 2021. Privacy of DNS-over-HTTPS: Requiem for a Dream?. In *IEEE European Symposium on Security and Privacy*. IEEE Computer Society, Los Alamitos, CA, USA, 252–271. https://doi.org/10.1109/EuroSP51992.2021.00026

[14] S. Dickinson, B. Overeinder, R. van Rijswijk-Deij, and A. Mankin. 2020. *Recommendations for DNS Privacy Service Operators.* BCP 232. RFC Editor.

[15] DNSCrypt project. [Accessed: Sep 2022]. DNSCrypt version 2 protocol specification. [Online], https://dnscrypt.info/protocol/.

[16] efficient iP. Mar 2019 [Accessed: Sep 2022]. Zero Trust: Verifying beyond perimeters, DNS security is key. Blogpost, https://www.efficientip.com/zero-trust-dns-security/.

[17] Justin Ellingwood. Nov 2014 [Accessed: Sep 2022]. Understanding Nginx HTTP Proxying, Load Balancing, Buffering, and Caching. DigitalOcean blogpost, https://www.digitalocean.com/community/tutorials/understanding-nginx-http-proxying-load-balancing-buffering-and-caching.

[18] ETSI GS NFV 002. Oct 2013. Network Functions Virtualization (NFV); Architectural Framework v1.1.1. http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf.

[19] Paul Fisher. Mar 2022 [Accessed: Sep 2022]. Kong Gateway 2.8: Increase Security and Simplify API Management. Product Releases, https://konghq.com/blog/kong-gateway-2-8.

[20] Sean Gallagher. Apr 2018 [Accessed: Sep 2022]. How to keep your ISP's nose out of your browser history with encrypted DNS. arsTechnica Blogpost, https://arstechnica.com/information-technology/2018/04/how-to-keep-your-isps-nose-out-of-your-browser-history-with-encrypted-dns/.

[21] Robert Gibb. Jun 2016 [Accessed: Sep 2022]. What is Time to First Byte? Stackpath Blogpost, https://blog.stackpath.com/time-to-first-byte/.

[22] Evan Gilman and Doug Barth. 2017. *Zero Trust Networks.* O'Reilly Media Inc, 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[23] Google. [Accessed: Sep 2022]. What is Apigee? Google Cloud Docs, https://cloud.google.com/apigee/docs/api-platform/get-started/what-apigee.

[24] Mike Hadlow. May 2013 [Accessed: Sep 2022]. The Benefits of a Reverse Proxy. DevOps Zone - Interview, https://dzone.com/articles/benefits-reverse-proxy.

[25] Mike Hathaway. Feb 2020 [Accessed: Sep 2022]. What is OCSP and how does it work? ascertia Blogpost, https://blog.ascertia.com/what-is-ocsp-and-how-does-it-work.

[26] P. Hoffman and P. McManus. 2018. *DNS Queries over HTTPS (DoH).* RFC 8484. Internet Engineering Task Force. http://www.rfc-editor.org/rfc/rfc8484.txt

[27] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. 2016. *Specification for DNS over Transport Layer Security (TLS).* RFC 7858. Internet Engineering Task Force. 1–19 pages. http://www.rfc-editor.org/rfc/rfc7858.txt

[28] IANA. Feb 2020 [Accessed: Sep 2022]. Technical requirements for authoritative name servers. Online: https://www.iana.org/help/nameserver-requirements.

[29] Infoblox. May 2020 [Accessed: Sep 2022]. DNS Security is Critical to Zero Trust Network Architecture. Blogpost, https://blogs.infoblox.com/security/dns-security-is-critical-to-zero-trust-network-architecture/.

[30] IONOS. May 2022 [Accessed: Sep 2022]. DNS TTL best practices: Understanding and configuring DNS TTL. Configuration guide, https://www.ionos.com/digitalguide/server/configuration/understanding-and-configuring-dns-ttl/.

[31] ISC. [Accessed: Sep 2022]. Bind9. Online, https://www.isc.org/bind/.

[32] ISC. [Accessed: Sep 2022]. Bind9.17.7 - Release notes. Online, http://ftp.iij.ad.jp/pub/network/isc/bind9/9.17.7/doc/arm/html/notes.html#notes-for-bind-9-17-7.

[33] ISPreview. Sep 2019. Firefox Says – NO DNS Over HTTPS (DoH) by Default for UK. Blog post, https://www.ispreview.co.uk/index.php/2019/09/firefox-says-no-dns-over-https-doh-by-default-for-uk.html.

[34] JH Software. 2021 [Accessed: Sep 2022]. New in Simple DNS Plus. Online, https://simpledns.plus/kb/194/new-in-simple-dns-plus-v9-0.

[35] Michael Jones, John Bradley, and Nat Sakimura. 2015. JSON Web Token (JWT). RFC 7519. https://doi.org/10.17487/RFC7519

[36] Dan Kegel. 2014 [Accessed: Sep 2022]. The C10K problem. Online, http://www.kegel.com/c10k.html.

[37] Dae Wook Kim and Junjie Zhang. 2015. You Are How You Query: Deriving Behavioral Fingerprints from DNS Traffic. In *Security and Privacy in Communication Networks*, Bhavani Thuraisingham, XiaoFeng Wang, and Vinod Yegneswaran (Eds.). Springer International Publishing, Cham, 348–366.

[38] Kinsta. Jan 2022 [Accessed: Sep 2022]. What Is Nginx? A Basic Look at What It Is and How It Works. Glossary, https://kinsta.com/knowledgebase/what-is-nginx/.

[39] KNOT DNS. [Accessed: Sep 2022]. kdig – Advanced DNS lookup utility. Online, https://www.knot-dns.cz/docs/2.6/html/man_kdig.html.

[40] Diego Kreutz, Fernando Ramos, Paulo Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2014. Software-Defined Networking: A Comprehensive Survey. *ArXiv e-prints* 103 (06 2014). https://doi.org/10.1109/JPROC.2014.2371999

[41] Chenta Lee. Dec 2021 [Accessed: Sep 2022]. Zero Trust and DNS Security: Better Together. SecurityIntelligence Blogpost, https://securityintelligence.com/posts/zero-trust-dns-security/.

[42] Zi Lin. Feb 2015 [Accessed: Sep 2022]. TLS Session Resumption: Full-speed and Secure. Cloudflare Blog post, https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/.

[43] Linux manpages. Aug 2021 [Accessed: Sep 2022]. bridge(8) — Linux manual page. Online, https://man7.org/linux/man-pages/man8/bridge.8.html.

[44] Linux manpages. Aug 2021 [Accessed: Sep 2022]. iptables(8) — Linux manual page. Online, https://man7.org/linux/man-pages/man8/iptables.8.html.

[45] Linux manpages. Aug 2021 [Accessed: Sep 2022]. veth(4) — Linux manual page. Online, https://man7.org/linux/man-pages/man4/veth.4.html.

[46] Microsoft. Jan 2021 [Accessed: Sep 2022]. Lessons learned in engineering Zero Trust networking. Blogpost, https://www.microsoft.com/en-us/insidetrack/lessons-learned-in-engineering-zero-trust-networking.

[47] P. Mockapetris. 1987. *Domain names - concepts and facilities.* STD 13. RFC Editor. http://www.rfc-editor.org/rfc/rfc1034.txt http://www.rfc-editor.org/rfc/rfc1034.txt.

[48] NetworkComputing. Mar 2005 [Accessed: Sep 2022]. A GSLB Reality Check. Online, https://www.networkcomputing.com/data-centers/gslb-reality-check.

[49] Adis Nezirovic. May 2019 [Accessed: Sep 2022]. 5 Ways to Extend HAProxy with Lua. HAProxy blog, https://www.haproxy.com/blog/5-ways-to-extend-haproxy-with-lua/.

[50] Nginx. [Accessed: Jun 2022]. NJS Scripting Language. Online, https://nginx.org/en/docs/njs/.

[51] NGINX Docs. [Accessed: Sep 2022]. Deploying NGINX and NGINX Plus on Docker. Online, https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-docker/.

[52] NLNetLabs. [Accessed: Sep 2022]. Unbound - About. Online, https://www.nlnetlabs.nl/projects/unbound/about/.

[53] T. Pauly. 2020 [Accessed: Sep 2022]. Enable encrypted DNS. WWDC 2020 video, https://developer.apple.com/videos/play/wwdc2020/10047.

[54] R. Prakash. 2020 [Accessed: Sep 2022]. Build trust through better privacy. WWDC 2020 video (11:57), https://developer.apple.com/videos/play/wwdc2020/10676.

[55] Eric Rescorla. 2022. *The Transport Layer Security (TLS) Protocol Version 1.3.* Internet-Draft draft-ietf-tls-rfc8446bis-04. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-tls-rfc8446bis-04 Work in Progress.

[56] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. 2020. Zero Trust Architecture. NIST Special Publication 800-207,https://doi.org/10.6028/NIST.SP.800-207.

[57] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Dr. Carlisle Adams. 2013. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960. https://doi.org/10.17487/

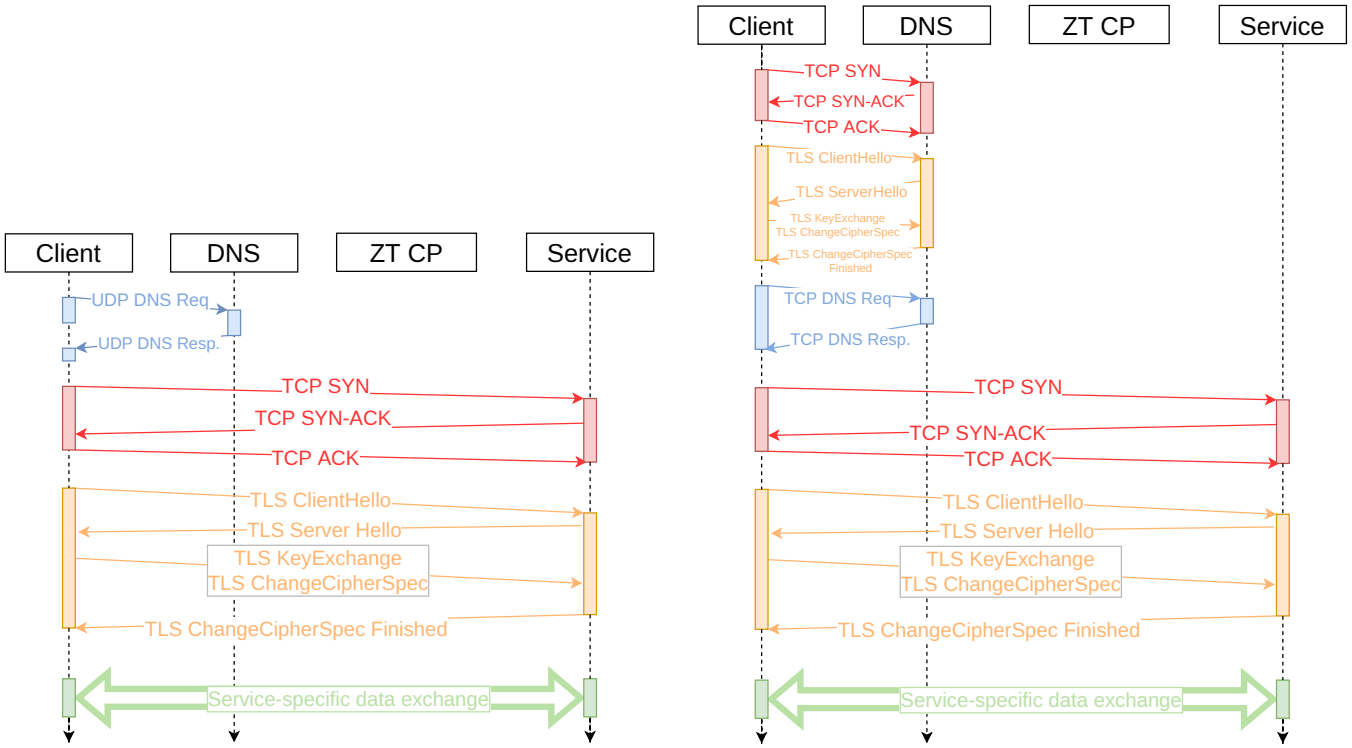
RFC6960
[58] Nick Sullivan. Mar 2017 [Accessed: Sep 2022]. Introducing Zero Round Trip Time Resumption (0-RTT). Cloudflare Blog post, https://blog.cloudflare.com/introducing-0-rtt/.
[59] Traefik Labs. [Accessed: Sep 2022]. Concepts. Online, https://doc.traefik.io/traefik/getting-started/concepts/.
[60] Traefik Labs. [Accessed: Sep 2022]. Plugins and Traefik Pilot. Online, https://doc.traefik.io/traefik/plugins/.
[61] International Telecommunication Union. Oct 2021. Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks. Standard, https://www.itu.int/rec/T-REC-X.509.
[62] Paul Vixie. 1999. *Extension Mechanisms for DNS (EDNS0).* RFC 2671. RFC Editor. http://www.rfc-editor.org/rfc/rfc2671.txt http://www.rfc-editor.org/rfc/rfc2671.txt.
[63] Rory Ward and Betsy Beyer. 2014. BeyondCorp: A New Approach to Enterprise Security. *;login:* Vol. 39, No. 6 (2014), 6–11.
[64] S. Weiler and D. Blacka. 2013. *Clarifications and Implementation Notes for DNS Security (DNSSEC).* RFC 6840. Internet Engineering Task Force. 1–21 pages. http://www.rfc-editor.org/rfc/rfc6840.txt

APPENDIX

Below, we show the sequence charts of the most important steps when establishing a connection to a service in a traditional network using unencrypted (cf. Fig. 5a) and encrypted DNS (cf. Fig. 5b), and in ZT using the default ZT mechanisms (cf. Fig. 6a) and ZeroDNS (cf. Fig. 6b).

Observe that in ZeroDNS (cf Fig. 6b), the TTFB of client applications can be close to the one in traditional networks using adequate encryption.

Note, since the actual steps after establishing the connection to a service is implementation-specific (i.e., data might be pushed/read after connection establishment, or further credentials are needed), similar to §3.2, we denote those subsequent steps as "Service-specific data exchange" for brevity. Furthermore, any additional mutual TLS authentication with the service is also omitted from the sequence charts. Similarly, in Fig. 6, the actual ZT authorization steps are not detailed for easier comprehension, i.e., they are collectively shown as `ZT authorization` (in Fig. 6a) and `ZT Auth.` and `ZT AuthZ tokens` (in Fig. 6b). Finally, for the same purpose (i.e., as being implementation-specific), the steps of the PEP at the service's site is not detailed either. Recall, however, that a PEP at the service site might make additional message exchanges with the ZT control plane for verification purposes, or the authorization tokens sent by the client contains the necessary information to do the same (i.e., verifying the signature is sufficient).



(a) Traditional way with DNS over unencrypted UDP.

(b) Traditional way with encrypted DNS.

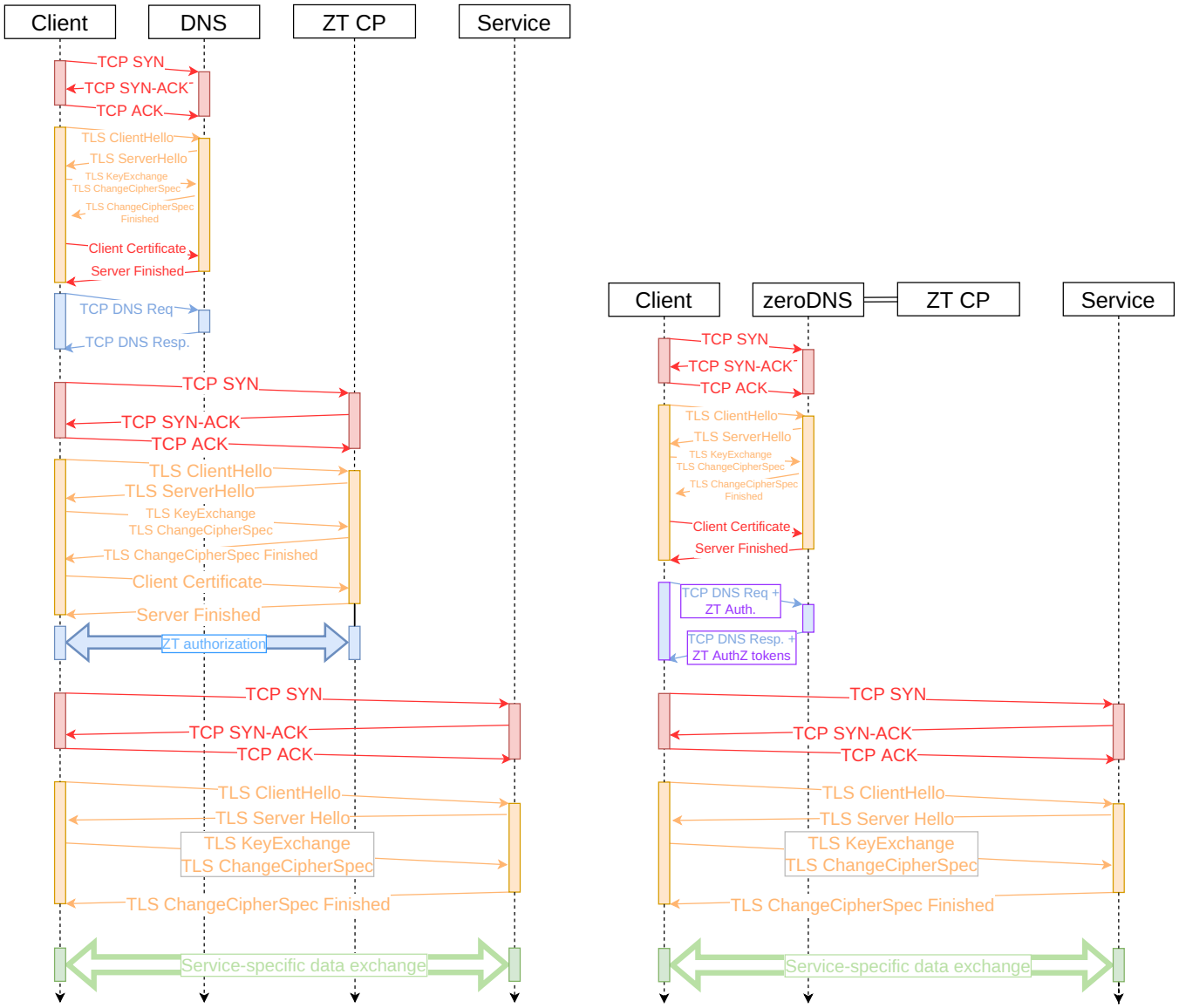
Figure 5: Sequence charts of the most important steps when establishing a connection to a service in a traditional network using *(a)* unencrypted DNS over UDP, and *(b)* encrypted DNS over TCP/TLS.

```
;; ANSWER SECTION:
mtls-dns.com.          55   IN   A 172.30.1.10
mtls-dns.com.          60   IN   TXT "JWT:::webservice1-->eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.
     eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWUsImlhdCI6MTUxNjIzOTAyMiwiZ
     W1haWwiOiJhZG1pbkBhZG1pbi5jb20iLCJyb2xlcyI6ImFkbWluICwgdXNlciwgc3lzYWRtaW4iLCJmdXJ0aGVyX3Rva2  VuIjoid2dobm"
mtls-dns.com.          60   IN   TXT "hkZ3VvNHduaG9uZ2Y0OHcwdDM0bmVmYWVkbmZxb2lvZWFuZiJ9.
     FL1LhOYjSWkskEMWMla1niWBOjzCzNHtJ6SPSk0mdJpIYrDDRHLQcAdqfPaxwUr0K_gvOReyqnCEsDdHgiG3gg"
mtls-dns.com.          55   IN   TXT "JWT:::webservice2-->user1_token"
```

Listing 1: Example of an ANSWER section of a modified DNS response in ZeroDNS

```
{
  "tokens":
  {
    "CN=Client1":
    {
      "webservice1":
      {
        "token":"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFt
        ZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWUsImlhdCI6MTUxNjIzOTAyMiwiZW1haWwiOiJhZG1pbk
        BhZG1pbi5jb20iLCJyb2xlcyI6ImFkbWluICwgdXNlciwgc3lzYWRtaW4iLCJmdXJ0aGV yX3Rva2V
        uIjoid2dobmhkZ3VvNHduaG9uZ2Y0OHcwdDM0bmVmYWVkbmZxb2lvZWFuZiJ9.FL1LhOYjSWkskEMW
        Mla1niWBOjzCzNHtJ6SPSk0mdJpIYrDDRHLQcAdqfPaxwUr0K_gvOReyqnCEsDdHgiG3gg",
        "ttl":60
      },
      "webservice2":
      {
        "token":"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFt
        ZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWUsImlhdCI6MTUxNjIzOTAyMn0.VFb0qJ1LRg_4ujbZoR
        MXnVkUgiuKq5KxWqNdbKq_G9Vvz-S1zZa9LPxtHWKa64zDl2ofkT8F6jBt_K4riU-fPg",
        "ttl":55
      }
    },
    "CN=Client2":
    {
      "webservice1":
      {
        "token":"eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFt
        ZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWUsImlhdCI6MTUxNjIzOTAyMn0.VFb0qJ1LRg_4ujbZoR
        MXnVkUgiuKq5KxWqNdbKq_G9Vvz-S1zZa9LPxtHWKa64zDl2ofkT8F6jBt_K4riU-fPg",
        "ttl":66
      }
    }
  }
}
```

Listing 2: Example JSON of a simple policy in ZeroDNS. Clients are identified by their client certificates and tokens are defined for each service a client has access to.



(a) Default Zero Trust architecture.

(b) ZeroDNS.

Figure 6: Sequence charts of the most important steps when establishing a connection to a service in ZT using *(a)* the default ZT mechanisms, and *(b)* in ZeroDNS.