Avoiding VPN Bottlenecks: Exploring Network-Level Client Identity Validation Options

Yu Liu(✉) and Craig A. Shue

Worcester Polytechnic Institute, Worcester, MA 10609, USA
{yliu25,cshue}@cs.wpi.edu

Abstract. Virtual private networks (VPNs) allow organizations to support their remote employees by creating tunnels that ensure confidentiality, integrity and authenticity of communicated packets. However, these same services are often provided by the application, in protocols such as TLS. As a result, the historical driving force for VPNs may be in decline. Instead, VPNs are often used to determine whether a communicating host is a legitimate member of the network to simplify filtering and access control. However, this comes with a cost: VPN implementations often introduce performance bottlenecks that affect the user experience.

To preserve straightforward filtering without the limitations of VPN deployments, we explore a simple network-level identifier that allows remote users to provide evidence that they have previously been vetted. This approach uniquely identifies each user, even if they are behind Carrier-Grade Network Address Translation, which causes widespread IP address sharing. Such identifiers remove the redundant cryptography, packet header overheads, and need for dedicated servers to implement VPNs. This lightweight approach can achieve access control goals with minimal performance overheads.

Keywords: Virtual private networks · Access control · Software-defined networking · Residential networks · Carrier-grade NAT

1 Introduction

Virtual private network (VPN) protocols are often used by organizations to allow remote users to access the organization's network as if they were on-site. These protocols have been used for decades [3] and were designed for an Internet that needed cryptography to protect the confidentiality, integrity, and authenticity of communication payload. In essence, VPNs allow organizations to treat the traffic from a remote worker the same as that from a local worker.

In recent years, the deployment of end-to-end cryptography has grown substantially, with over 90% of web servers supporting the TLS/SSL application-layer protocol [41]. When remote users access an HTTPS server through a VPN,

X. Yuan et al. (Eds.): QShine 2021, LNICST 402, pp. 280–300, 2021.
https://doi.org/10.1007/978-3-030-91424-0_17

the traffic is encrypted and authenticated by TLS between the application-layer endpoints and is again encrypted and authenticated between the VPN termination points. Since confidentiality, integrity, and authenticity can be reasonably assured by either of the protocols, having both is redundant. Further, VPN deployments often come with performance overheads since 1) VPN servers are usually an aggregation point for network traffic and their limited resources can cause network congestion [22,45], 2) redundant protocol headers use more space in each packet, and 3) VPN licenses, which cost more than $2 million annually for a large company, can be expensive [47].

The traditional motivations for deploying a VPN [36,38,39] are to 1) provide data with confidentiality and authenticity, 2) manage network activities at the remote endpoint, and 3) simplify access control. Application-layer security achieves the first goal. We address the second goal in Sect. 2.5, where we discuss better management through endpoint filtering. We explore the third goal, access control, throughout this work. Importantly, 67% of organizations are exploring VPN alternatives for security, maintenance, and fiscal reasons [54].

Organizations often implicitly use a host's position within a network perimeter as a factor in determining whether to trust the host or not. As examples, the configuration instructions for web servers [37,40], email servers [12,33,34], and firewalls [9] often describe how administrators can configure the systems to permit only traffic within an organization's IP prefix. Organizations may also use network address translation (NAT) to assign private, unroutable addresses to hosts and devices so that the infrastructure cannot be reached by outsiders without traversing NAT devices that could enforce policy [5].

While organizations can employ such address filtering within their networks, it may be impractical to do so with remote users. Internet Service Providers (ISPs) often use dynamic addresses for their customers, with lease times that vary greatly [6]. Residential users often deploy NAT in their home networks to share addresses, so authorizing a particular user's IP address would allow others at the same residence access to the organization's resources. Even worse, some ISPs have adopted a competing technology called carrier-grade network address translation ("carrier-grade NAT" or CGN). CGN is used in 92% of cellular networks [1]. Some of these carriers plan to use 5G cellular networks to provide residential network connectivity, with some providers estimating 30 million home networks will be connected in this fashion [11]. In some cases, hundreds of users share an IP address [2] and in others, a single customer's traffic may be simultaneously associated with multiple public IP addresses.

Organizations may use VPNs to mitigate the problems that accompany address sharing. The VPN tunneling approach allows an organization to provide remote access to systems that are highly sensitive or have weaker protections, such as printers, Supervisory Control and Data Acquisition (SCADA) or Internet of Things (IoT) infrastructure, or other embedded devices. However, a heavyweight VPNs approach may not be necessary to achieve these goals.

Organizations need a quick and simple way for a network-level identifier to validate a remote end user. This validation can be a quick, "first-factor"

authenticator that provides evidence that the connecting machine or network is likely legitimate. This factor can work with other authentication factors, such as application-layer credentials, on the server endpoint. It must be cooperatively used by both the end-user and the organization to avoid abuse by malicious parties. While identifiers have been used by network providers to identify clients in the past, such as "super cookie" deployments in cellular networks [13], the use of the factor in authentication requires it to be cross-application, dynamic, and under the associated user's control.

With a "less may be more" perspective, we explore a practical and deployable approach to allow end-users to create dynamic network-level factors for access control. Our contributions include:

- **Dynamic Identifier Insertion:** Our software-defined networking (SDN) approach leverages endpoint programs and modified residential routers to insert application-agnostic identifiers into network flows. This approach allows organizations to determine if a user is connecting from a known device or network location while introducing minimal latency overheads that affect only the first packet in each network flow.
- **Gateway and Endpoint Validation:** Using our implementation, built using the popular `iptables` tool, organizations can validate clients at either a gateway or endpoint, providing functionality similar to a VPN without unnecessary performance overhead. This method can verify each flow in around 90 ms (of which only around 0.8 ms is spent at the validator). Unlike a VPN, it adds no overhead after the handshake. We eliminate the need for a VPN server and its associated CPU bottleneck, resulting in a 2.5–2.9 times increase in throughput for clients.

2 Background and Related Work

This section discusses background and related work on CGN, software-defined networking, user identity, and techniques to encode and transmit identifiers.

2.1 Carrier-Grade NAT (CGN) and Address Sharing

As IPv4 addresses availability became scarce, Internet Service Providers (ISPs) started deploying CGN to share IPv4 addresses among subscribers and to minimize disruption during the transition from IPv4 to IPv6. CGN utilizes both network-level addresses and transport-layer port numbers to map traffic to the appropriate end-user. The number of public addresses needed for a given number of end-users can be estimated with formulas. Some guides suggest sharing 30,517 public IP address among 1,000,000 subscribers [2]. In measuring CGN behavior, Richter et al. [1] observed that some CGNs used the same public IP address and varying transport layer ports for subsequent TCP sessions from the same subscriber. In Netalyzr [6], the authors found that more than 60% of the TCP sessions for a given subscriber used different public IP addresses. These guides

and measurements show that providers use highly dynamic, and widely shared, public IP addresses for their subscribers.

With their Revelio tool, Mandalari et al. [7] performed Internet measurements and found around 10% of ISPs deployed CGN. With web server logs and multiple measurement points, Livadariu et al. [8] estimated that around 4.1k of the 17.4k ASes they measured deployed CGNs. Richter et al. [1] found that while only 13.3% of non-cellular ASes use CGNs, 92% of cellular networks use them [1]. In their plans to deploy 5G cellular connectivity, some providers estimate they will serve 30 million residential networks through 5G [11].

2.2 Software-Defined Networking (SDN)

The software-defined networking approach separates the data plane and control plane for network traffic, often using a centralized network controller. The OpenFlow protocol [30] allows a network controller to alter data structures in switches and routers to enable inspection and arbitrary forwarding of packets. The Open vSwitch [19] tool can be used to enable the OpenFlow protocol on a router. In OpenFlow, a controller can intercept packets from a network device via a `PACKET_IN` message. To authorize a packet, with possible alteration, the controller replies with a `PACKET_OUT` message. A controller can also command switches to cache certain rules through `FLOW_MOD` messages, allowing switches to process subsequent packets without controller involvement. This avoids causing performance overheads in subsequent packets in a flow.

2.3 Host Identity and Reputation Systems

Address sharing introduces challenges for a wide range of applications relying on public IP addresses. For example, enterprise-grade firewalls often utilize public IP addresses in policies [20]. Efforts to mitigate DNS amplification attacks use IP addresses in response rate limiting [29]. IP reputation systems, which are used by major email providers, are often used to determine the threat associated with incoming email messages [12], such as Microsoft's SmartScreen technology in Outlook [33] or Gmail's delivery rate throttling [34]. For websites, Cloudflare identifies users with a bad IP reputation and challenges them with CAPTCHAs [35]. Such tools may mistakenly assume that IP addresses change infrequently and are unlikely to be shared. This leads to false negatives when attackers move across IP addresses and false positives when innocent people happen to use an IP address previously involved in an attack [10].

Komu et al. [31] investigated methods to separate the functionalities of "locator" and "identifier" from network addresses. The locator can be used to find a host while a separate long-term identifier is associated with the system. This separation is important for mobile hosts or for times when addresses change. HIP is a protocol to maintain persistent identity even with dynamic IP addresses [27]. HIP uses a public key to identify end-hosts and uses IPSec for packet tunneling. Since HIP requires HIP-aware gateways to forward packets to the correct destination, the deployment of HIP requires infrastructure changes.

DeCusatis et al. [56] introduce a TCP-based access control mechanism using first packet authentication. That work uses an ephemeral four-byte value in the TCP protocol that is used for access control and must be established for each interaction. That work only evaluates token evaluation. In contrast to our work, it does not explore token creation, insertion, storage, or protocols for conveying these values in a way that allows longer-term, cross-protocol use. Other work has proposed persistent identifier, but provided only an abbreviated analysis of performance impacts in limited deployment scenarios [46]. Our work provides the necessary information and analysis for a practical implementation.

At the application layer, web applications can track user identities with cookies, supported by browsers. Unfortunately, cookies are application-specific and only work with web traffic. In some cases, network providers create persistent identifiers, called "super cookies," to identify systems at the device-level [28]. These super cookies are outside the end-user's control. They enable tracking that could violate end-user privacy. These deployments resulted in fines for some ISPs [13]. Our approach is mindful of these potential privacy concerns. A key design goal is to allow end-users to have control over persistent identifiers while supporting multiple applications, which we discuss in Sect. 3.4.

Organizations often use virtual private network (VPN) protocols, such as IPSec [3], to authenticate remote users and then leverage the VPN server's position inside a local area network (LAN) to provide access to LAN resources. As aggregation points, VPN servers can become throughput bottlenecks since they must be involved in the entire network flow and use cryptography to encrypt and authenticate traffic, even if the application-layer already offers that support (e.g., in HTTPS or SSH). Application-layer software typically lacks options to configure IPSec tunnels for specific flows or destinations. Instead, current VPN software works across all applications on a device-wide basis. Often, all network traffic from a host in a VPN is forwarded to the VPN server, which increases overhead and decreases throughput. Hauser et al. [57] propose an SDN extension to IPSec for programmable data planes. In practice, VPNs can increase organization costs [47], reduce performance [22], create single points of failure [21], and add complexity [44]. In our approach, we avoid these limitations.

2.4 Mechanisms to Encode Application-Agnostic Identifiers

Protocols such as TCP and IP support options for communicating information, such as identifiers and authenticators. Options in the IP header can be used for all transport layer protocols, rather than just TCP. However, intermediary routers may drop packets with IP options they do not support [17,18].

Prior work has examined using "shim" layers between the IP header and transport layer headers to encode data [26]. IPSec does so using the ESP or AH headers to encapsulate protected traffic [24,25]. Special-purpose shims have the downside of requiring support from endpoints and the risk that they will be discarded by firewalls or routers that do not understand the shim layer headers. However, the IP-in-IP tunneling technique, standardized in RFC 1853 [15], essentially provides a second IP header as a shim layer. The use of an IP-in-IP

shim gives us a straightforward way to add options in a backwards-compatible manner. We can insert a second IP header in front of the original transport layer header using the standardized approach. Mobile IP [14] uses this same technique.

2.5 Motivations and Perspectives with VPN Deployments

Commonly cited reasons for organizations to use VPNs with their employees include [38,39]: 1) to build a communication tunnel with confidentiality, integrity, and authenticity via cryptography, 2) to control communication to remote systems, and 3) to provide authentication to achieve simplified access control. In this section, we explore these techniques and describe how they may be affected by changing Internet trends. We also note recent changes in VPN planning.

Confidentiality, Integrity, and Authenticity. Web traffic comprises a majority of Internet communication. Currently, 90% of HTTP traffic is protected by TLS [4,41]. TLS supports common business applications, such as remote desktop, file transfer, and remote terminals. Further, studies show that over 98% of printers support Internet Printing Protocol (which supports encryption and authentication) [42]. Most network traffic is protected by application-layer cryptography, which achieves confidentiality, integrity and authenticity.

Some legacy protocols or devices may not support cryptography. However, organizations may use the reverse proxy model [43] to protect such devices by creating application-specific security tunnels without requiring VPNs.

Organizations Pursuing VPN Alternatives. In a recent report [54], Zscaler indicates that 67% of companies are seeking alternatives to VPNs. In addition to performance issues, companies expressed concerned about 1) the changing role of VPNs with pandemic-related work-from-home patterns, 2) increased VPN infrastructure's impact on organizational architecture, which makes maintenance more complex and expensive, and 3) attackers who are increasingly using VPNs to gain access to corporate networks via social engineering and malware. Given this context, 77% of companies have indicated an interest in using a zero-trust model to manage remote access for their employees instead. Our approach aligns with these corporate goals.

3 Approach: Indicating Authenticity Validation

Inspired by Kerberos and HTTP cookies, we explore a token-based identity approach. A remote authenticator distinguishes legitimate and unauthenticated users via a token provided by the user and device. We describe the goals of such a system and how they differ from the robust authentication present in end-to-end applications. We then describe our threat model and the techniques we use.

3.1 Design Goal: Evidence Supporting Legitimacy

VPN servers can robustly authenticate their remote VPN gateways. For gateway-to-gateway VPNs that interconnect two LANs, this approach may not uniquely identify the connecting end-user. The VPN server may be able to uniquely identify the connecting end-user if the remote VPN gateway runs on the remote user's endpoint. However, the VPN server does not share that identification with the endpoints that the user then connects to through the VPN server. The VPN server often performs NAT to proxy the connection between the remote user and server, but the IP addresses may be randomly selected from the available IPs. The server endpoint may be able to infer that the end-user authenticated with the VPN server by determining if the remote IP belongs to an IP address associated with the VPN server's pool. This is a useful, but relatively weak authenticator because it lacks a unique identifier or strong authenticity guarantees.

Our approach aims to re-build the authentication between a remote user and an organization network and the weak authenticator available to a remote server.

3.2 Threat Model

Our approach enables clients to provide an application-agnostic device-level authenticator. This authenticator is not designed to be authoritative about the identity of a client. Instead, it is a quick "first-pass" authenticator that can be used to separate out "likely legitimate" traffic from completely unknown traffic. It can be used in combination with application-layer authentication (e.g., as a mechanism to address brute-force guessing on SSH servers). Enterprises can also calculate reputation based upon these identifiers, with better reputation leading to better services, as incentives.

Our approach is designed to effectively defend against "on-the-side" attackers, such as a user who is not on the network path, but who might share IP address with a client (e.g., behind the same CGN). We deploy identifiers that protect against any brute-force guessing. However, "on-path" adversaries can inspect the inner IP header, observe the identifiers, and misuse the identifier information. We rely on the application layer to provide robust authentication to defeat such powerful on-path adversaries.

3.3 Leveraging Authentication Servers

In our approach, we create a mechanism that allows an organization to authenticate its remote users using a lightweight device-level authentication factor. The end user can authenticate to the organization using a pre-existing authentication system, such as a web-based authentication page. This authentication system can use multi-factor authentication to robustly verify the end-user. Upon successful verification, the authentication system provides a token that can be used by the user's device as an authentication factor. If verification is unsuccessful, the server simply does not provide a token.

Our approach requires both the device and authentication server to automatically determine each others' support for the protocol, as well as for the other systems that are able to support the device-level authenticator. We enable automated deployment discovery using specially-crafted DNS records. When a client requests certain DNS records associated with the organization's domain (e.g., `A` records for www.example.com), the DNS server provides a `TXT` record in the Additional Records section of the response indicating the authentication server that is authorized to create authentication tokens for the domain. The `TXT` record also indicates which servers at the organization support the authentication scheme by listing public-facing IP address or CIDR prefixes.

3.4 Using OpenFlow to Manage Tokens

To avoid requiring support for the authentication mechanism in each application, we use an SDN technique to engage in the protocol on the client's behalf. An OpenFlow agent in the client's network, which could be on the client endpoint device itself (e.g., via Open vSwitch) or on a network gateway (e.g., a residential router), intercepts new flow requests and directs them to a SDN controller. The controller examines the related DNS responses for any `TXT` records that indicate support for the protocol. It also manages the authentication factors on behalf of the end-user on the device or on multiple devices in the network.

Our approach is designed to grant users full control over their identifiers. Unlike the "super cookies" approach [28], our system allows users to configure their identifies and choose when to use tokens with a remote party. The OpenFlow controller can allow users to manage the entries (e.g., via a web page). Based on the user's configuration and the destination of each flow, the controller determines whether to insert identifiers.

We use the standardized IP-in-IP tunneling approach [15] to create a "shim" layer. This creates two IP headers, allowing the outer IP header to be processed normally by routers while the inner IP header contains options that might otherwise result in the packet being dropped. We use those IP options to communicate the token that provides evidence of authentication.

When a client first interacts with an authorized authentication server, the OpenFlow agent intercepts the initial packet in the flow and sends it to the OpenFlow controller. The controller modifies the packet to signal that the client supports the scheme and sends it back to the OpenFlow agent, which then transmits it to the authentication server. Since the DNS records signal the server's support for the approach, we can construct packets that require the server to engage in custom parsing. The controller alters the packet to insert the IP shim layer, resulting in an IP-in-IP packet. In the inner IP header, the controller includes an option indicating that the client supports the protocol. Upon successful login, the authentication server replies with its own IP-in-IP packet, with the authentication data contained in an option field in the inner IP header. The OpenFlow agent elevates this response to the OpenFlow controller, which extracts the authentication factor, removes the IP shim header, and orders the OpenFlow agent to send the decapsulated packet to the client application.

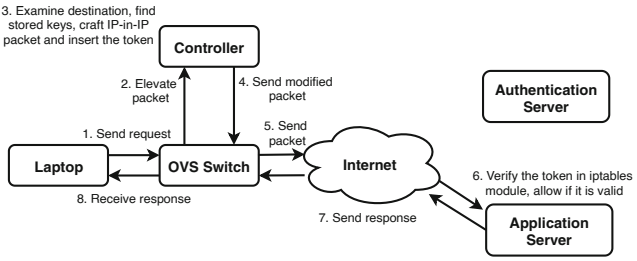


Fig. 1. The process for the client to authenticate to the application server.

When a client subsequently creates a new network flow to a server that supports the scheme, the OpenFlow agent intercepts the request and elevates it to the controller (Fig. 1). The controller again performs the necessary alteration of the packet to create the IP-in-IP shim that contains the pre-determined authentication data. The controller then returns the modified packet to the OpenFlow agent for transmission to the server. The server, or a gateway or middlebox on the path to the server, processes any packets containing the IP-in-IP shim to verify and strip the authentication data. In doing so, the server or middlebox can record that the flow is verified and, depending on policy, allow the flow where unverified flows may be denied.

Importantly, the inner IP header addresses can be used by a gateway or middlebox validator to implement NAT translations that allow a remote user to have local IP addresses in the same manner as VPN servers. However, unlike VPN servers, the inner IP header only needs to appear in the initial exchange to create the appropriate NAT mapping to translate the remote machine's address.

4 Implementation

We implement our system in a home network to allow us to evaluate its security and performance. As shown in Fig. 2, we explore the modifications that would be needed to include a regular client, an authentication server (e.g., a single sign-on identity provider), an application server (e.g., a relying party), and an SDN controller to coordinate it all.

We run our client in a virtual machine (VM) hosted on a Thinkpad S3 laptop with four cores and 8 GBytes RAM. We create two other VMs on a Macbook Pro laptop, with four cores and 16 GBytes RAM, for applications and authentication servers respectively. Each VM has one core and 4 GBytes memory. We configure a physical TP-Link Archer C7 router with OpenWrt and the Open vSwitch module as our SDN switch. The VMs are bridged through laptop interfaces and get DHCP services from the router. All the physical devices are located in the same home network. However, we configure our Floodlight SDN controller on a remote network machine with two cores and 4 GBytes memory.

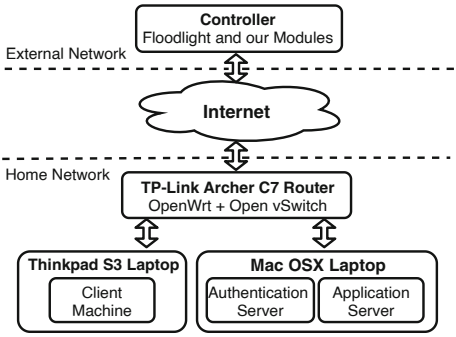


Fig. 2. Our experiment architecture.

4.1 Identity Provider Interactions

In Sect. 3.4, we described how the OpenFlow switch and controller cooperate to detect that the application and authentication servers support the approach. The Kerberos approach provides a shared-key architecture for identity providers to construct keys for relying parties [16].

The SDN controller learns about the authentication server through DNS records and signals the client's support via an IP-in-IP shim. When the authentication server replies, it includes a shared secret that the client can use to authenticate itself.

As with Kerberos, each application server pre-shares a unique key with the authentication server [55]. The authentication server uses that `applicationServerSecretKey`, along with a unique identifier for the user and a nonce value it generates, to produce a shared secret that is a one-way hash of these values (i.e., `clientKey=SHA224(uniqueIdentifier || nonce1 || applicationServerSecretKey)`). The authentication server then sends both the `clientKey` value and the concatenation of the `uniqueIdentifier` and `nonce1` value to the client.

The authentication server communicates the key and identifier by crafting an IP-in-IP packet. The inner IP header includes an option field in which both the `clientKey` and `uniqueIdentifier` are encoded. The protocol field differs between the two IP headers (the outer header indicates the protocol is another IP header, while the inner header indicates the transport protocol used), but otherwise the two headers contain identical values.

The OpenFlow switch elevates packets with IP-in-IP shims to the controller, allowing the controller to obtain the `clientKey` and the `uniqueIdentifier`.

4.2 Application Server Interactions

In our implementation, when the client initiates a connection to the application server, the OpenFlow switch elevates the request to the OpenFlow controller. The controller consults its database, determines that a token is needed, and

creates an IP-in-IP shim. In the inner header, it creates an IP option that contains the user identity, the authentication server's nonce (`nonce1`), the client's own nonce value (`nonce2`), and a SHA224 value constructed from the concatenation of the identity, `nonce2`, and `clientKey` (i.e., `SHA224(uniqueIdentifier || nonce2 || clientKey)`). The token we construct is 37 bytes total (5 bytes for the user ID, 2 bytes for the `nonce1`, 2 bytes for `nonce2`, and 28 bytes for the SHA224 output). The controller sends this modified packet back to the switch for transmission using an OpenFlow `PACKET_OUT` message.

The application server must parse the IP-in-IP message, validate the token, and then remove it before delivering it to the actual destination application. To do this, we develop an open source `iptables` module using the Xtables-Addons framework [32]. This module efficiently performs the interception and validation before the packet reaches the destination application.

Our functionality is divided into an `iptables` match module that specifies user-defined conditions. It passes any matching packets on to a target module for processing. We configure our match module to examine any IP-in-IP packets. For all such packets, it searches for our specific IP option type inside the inner IP header. If found, it parses the option to obtain the unique identifier and nonces. The match module then uses the client-supplied information and the key shared by the application server with the authentication server to calculate the corresponding `clientKey` (i.e., `clientKey=SHA224(uniqueIdentifier || nonce1 || applicationServerSecretKey)`). The application server then constructs a SHA224 digest using this `clientKey`, the `uniqueIdentifier`, and the client's nonce (i.e., `SHA224(uniqueIdentifier || nonce2 || clientKey)`) and compares it with the SHA224 value that is contained within the IP option. If the digests match, it knows the client interacted with the authentication server to obtain the `clientKey`. The match module only returns true if a match is found.

We next implement a target module that is used if the match module successfully validates a packet. Our target module modifies the `skb` buffer, which is the data structure used in Linux for packet processing. The target module must decapsulate the packet to remove the shim. It does so by removing the inner IP header and IP options, updating the protocol field in the outer IP header and recalculating the checksum. It then sends the packet to the application for processing. This allows the destination to validate the communication across applications. Importantly, the `iptables` tool can be run on a middlebox or on the application server itself to avoid bottlenecks.

5 Evaluation: Security and Performance

We evaluate our approach from both a security and performance perspective using the configuration depicted in Fig. 2. We focus on the performance of the token validation and shim layer operations between the client and application server, since these same operations are used in the interaction between the client and authentication server.

5.1 Security Evaluation

To assess our approach, we simulate the authentication process by issuing new network requests to the organization network, where our `iptables` modules are deployed on the servers. The application servers deny packets without validation by default. Only the packets approved by our match module can pass through `iptables` rules and be received by the services that run on the server.

Table 1. Result of effectiveness evaluation. We performed experiments 20 times for three scenarios: 1) network request without a token, 2) network request with an invalid token, and 3) network request with a valid token.

Result	No token	Invalid token	Valid token
Access allowed	0	0	20
Access denied	20	20	0

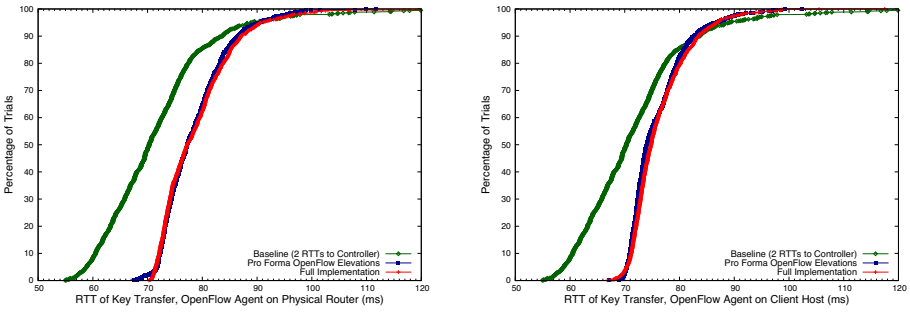
The first experiment setup simulates network queries without providing a token. As shown in Table 1, all 20 requests were successfully blocked by the default deny rule set up on the application server. In our second scenario, we enable the OpenFlow and `iptables` modules we implemented. We craft client request packets that contain invalid tokens. As shown in Table 1, all 20 requests failed to reach the applications. In our third scenario, we also enable our OpenFlow and `iptables` modules on both client and server side. We craft proper packet headers and the first packets of these flows contain valid tokens. As shown in Table 1, all such requests were approved by our `iptables` module.

5.2 Network Delay Overhead Evaluation

Our approach affects only the first exchange in each flow. The controller elevation to insert the shim, and the `iptables` processing to validate and remove the shim, are only needed on the initial message from the client to the server. Subsequent packets in the flow are not modified and or inspected by our custom `iptables` module. Those packets will proceed through standard packet processing. Therefore, the only significant overhead in our approach is incurred in the initial round-trip, so we focus our measurements accordingly.

During key transfer, the client initiates a TCP connection to the authentication server. Since the authentication server program is essentially unchanged, we use a simple echo reply to omit its overhead. For our measurements, we transmit a TCP `SYN` packet to a port without an associated application server, resulting in a TCP `RST` packet that refuses the connection. This simple exchange allows us to monitor any overheads at the OpenFlow controller and `iptables` modules to signal support for the protocol, encode keys into packets, and extract those keys. The client sends 1,000 TCP requests and measures the round-trip time (RTT) that includes all the overheads.

Our evaluation examines two deployment scenarios: 1) where the OpenFlow agent runs on a separate, physical router for a local network deployment and 2) where the OpenFlow agent runs on the client machine itself, in which the endpoint natively supports the use of a controller for persistent identity. Our router-based experiment uses a TP-Link Archer C7 router (see Fig. 2).



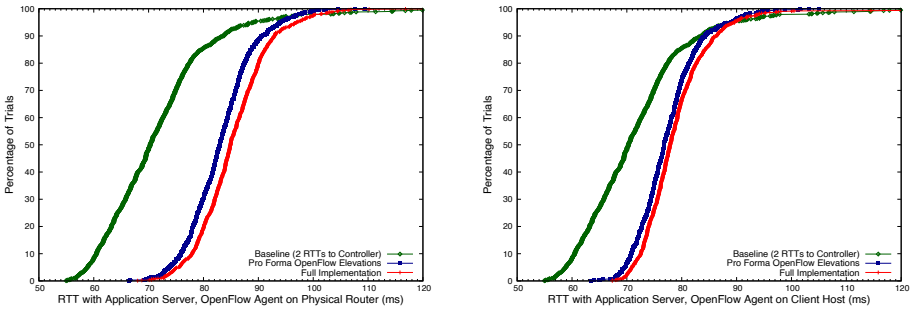
(a) Key Transfer via SDN Router Agent (b) Key Transfer via SDN Client Agent

Fig. 3. Round trip time for key transfer (1,000 trials) (Color figure online)

During the key transfer between the authentication server and client SDN controller, two elevations to the OpenFlow controller are required. Since this includes significant propagation delay to and from the controller, we measure that RTT and refer to it as the baseline in our method. In Fig. 3a, we explore the scenario where the OpenFlow agent runs on a physical router. In the diagram, the leftmost (green) line indicates the baseline case of two RTTs with the controller. The 90th percentile is 83 ms. The middle (blue) line shows a control experiment measuring end-to-end RTT for the client to the server using *pro forma* OpenFlow elevations, where the OpenFlow agent is configured to elevate each packet for approval, but the controller simply approves each packet without changes (i.e., the controller simply approves packets using `PACKET_OUT` messages without using `FLOW_MOD` rules). In the pro forma exchanges, no packet encapsulation or `iptables` verification occurs. In this scenario, the 90th percentile of the RTT is 86 ms. In our full key transfer implementation (the rightmost, red line), the 90th percentile is 87 ms. The similarity of the results of the pro forma and full implementations indicate that the overheads for encapsulation, `iptables`, and the OpenFlow controller are not significant.

We show the results where the OpenFlow agent runs on the client host in Fig. 3b. The baseline remains same. In the pro forma scenario, the 90th percentile is 82 ms. When we enable the full key transfer functions, the 90th percentile is 83 ms. The OpenFlow overheads are lower when the client runs the OpenFlow agent rather than a router. However, the overheads of encapsulation, `iptables`, and controller processing remain similar.

Next, we evaluate the overhead introduced by the key validation functionality between the client and the application server. In this experiment, our client



(a) Verification via SDN Router Agent (b) Verification via SDN Client Agent

Fig. 4. RTT for application server validation (1,000 trials) (Color figure online)

sends 1,000 UDP packets to the server. We measure the end-to-end delay from transmission to response, so each trial contains two packet elevations: 1) the elevation to the SDN controller for IP-in-IP encapsulation and 2) for the SDN controller to processes the UDP response.

In Fig. 4a, the green line (leftmost) again shows the baseline of two RTTs where the 90th percentile RTT is 82 ms. The blue line (middle) shows the overhead of the pro forma scenario in which 90% of the results have less than 90 ms delay. The red line (rightmost) shows our full implementation, which introduces 93 ms of delay or less for 90% of the trials. When we move the OpenFlow agent to the client machine, the pro forma 90th percentile drops to 84 ms and the full implementation drops to 86 ms, as shown in Fig. 4b. We again see that the full implementation has only modest overheads over a basic OpenFlow elevation approach. Further, the time spent at the validator in the full implementation was around 0.8 ms, indicating the verification overheads are low.

Importantly, the latency overheads incurred here occur only on the first round-trip between the client and the application server for each flow. Since they do not affect ongoing flows, they are unlikely to have a major impact on the end-user's experience. A subsequent optimization, to insert a `FLOW_MOD` during the first encapsulation, would cut the propagation time in half, reducing the RTT by roughly 40 ms in these experiments. End-users could further reduce their delay by hosting the controller closer to the client, such as in the LAN or in nearby ISP-hosted data centers.

5.3 VPN Server Throughput Comparison

As we will discuss in Sect. 6.2, our approach aims to improve bandwidth performance when an enterprise deploys its VPN server on a general-purpose machine. In this section, we simulate real environments and design experiments to create that bottleneck. We demonstrate the extent to which our approach can remove the bandwidth bottleneck associated with many VPN deployments. We explore five scenarios with some employing TLS, VPNs, and our approach.

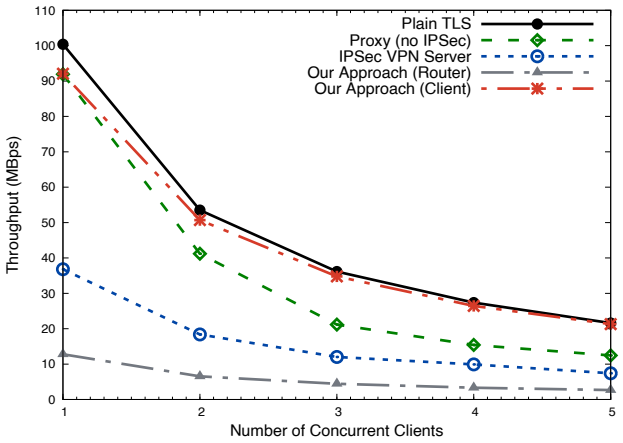


Fig. 5. Median client throughput in different security tool deployment scenarios.

In these experiments, we use `Strongswan` as our VPN software and `apache2` to host a large file for download via HTTPS. For each case, we run a varying number of clients concurrently to determine the per client throughput in each scenario. We measure the time used for downloading for each client, as well as CPU usage on the servers. In the experiments, we explore up to five clients since this degree of parallelism is sufficient to expose bottlenecks and demonstrate trends. Except as noted, we transfer a 1 GByte file in our trials. Each data point is the result of 30 trials.

In Fig. 5, we show the results of these scenarios. The first scenario, `Plain TLS`, represents a baseline in which the enterprise network has a TLS server that clients directly access for file transfer. This essentially represents an upper-bound on performance capabilities of the system. The traffic is constrained by the web server's ability to send traffic. In the second scenario, `Proxy (no IPSec)`, we forward the network traffic through an Ubuntu server VM that simply proxies traffic (i.e., forwards it using IP addresses) without any additional services (like IPSec). The throughput decreases in this scenario since the gateway host starts to constrain throughput and the CPU on the gateway vary from 35.41% to 95.83% utilization as we increase number of concurrent clients. In the `IPSec VPN Server` scenario, we enable IPSec on the Ubuntu proxy server and clients use the IPSec tunnel to reach the web server. This scenario represents an enterprise configuration in which organizations host their VPN services on a generic server. The CPU use climbs on the VPN server to around 100% usage and the result is a decrease in bandwidth from the baseline by around 65%. Compared to the 21.6 MBps in the baseline, the VPN server is only able to provide 7.4 MBps throughput per client during a five concurrent client scenario.

Next, we explore our approach using SDN support in either a consumer-grade router or in the endpoint itself. We first enable our approach in a consumer-grade router in the `Our Approach (Router)` scenario. This scenario does not

require the gateway server from the second or third scenarios and reflects the architecture as described in Sect. 3. The residential router scenario results show the challenges of repurposing hardware with limited computational capabilities. While it can deliver 12.8 MBps for a single client our tests of a 100 MByte file, it exhausts the router's computational resources (since it uses the general purpose CPU for OpenFlow forwarding lookups). While we explore up to five concurrent clients for a consistent presentation of results, these client-side routers would likely only service a single user at a time and do not act as an aggregation point, unlike the VPN server. With more capable residential routers, the computational limits would be less likely to constrain performance.

In our final scenario, `Our Approach (Client)`, we explore the approach where the SDN agent runs in software on the client, allowing us to characterize the performance implications of running the SDN agent on the residential router. When running on the client, the SDN functions no longer serve as a performance bottleneck (CPU usage at the client does not exceed 31%). This approach yields a performance improvement of roughly 2.5 to 2.9 times the throughput of an IPSec VPN. The performance decrease of the SDN approach verses the baseline ranges from 1% to 8%. Accordingly, with endpoint software, clients can attain far better throughput than VPNs and approximate the baseline.

5.4 Packet Header Overhead

VPNs may have to combine multiple protocols together to support some clients. A standardized implementation for this combines IPSec with L2TP [23]. When used with ESP and preshared secrets, the combined packet headers and trailers for the two protocols amounts to around 92 bytes (40 bytes for L2TP with UDP, 20 bytes for an encapsulated IP header, 16 bytes for the ESP header, 2 bytes for padding, and 14 bytes for the ESP trailer and authentication data). This can reduce the maximum transmission unit (MTU) for payload in many networks from 1500 to 1408, which is around a 6.1% reduction in payload per packet. This overhead occurs in each packet in the flow.

In our approach, we use packet encapsulation on the first packet sent from a client to an application server. Our IP-in-IP shim uses 20 bytes for the inner IP header, with an additional 40 bytes for our IP option, for a total of 60 bytes of overhead. Unlike VPN traffic, this overhead only applies to the first packet in a flow. For applications using TCP, this overhead would apply to the TCP SYN packet. Since those packets do not carry payload, our approach would often avoid the MTU complications present in VPN protocols.

6 Discussion

Our approach focuses on mechanisms that eliminate the need for VPN software by providing application servers with evidence that a client has been successfully authenticated. We now explore how a similar concept could be used with other kinds of services. We also explore scenarios in which we compare our system with VPN deployments and the role it can play in addressing bandwidth bottlenecks.

6.1 A Second-Factor Service for Public Infrastructure

For sensitive transactions, organizations with public-facing services can minimize risk by using multiple sources of evidence. For example, financial institutions may authenticate an end-user in multiple ways to minimize the risk in financial transactions. These forms of validation can include username and password, browser cookies, the use of one-time passcodes via SMS messaging or applications, or answers to secret questions.

Some organizations try to reduce risk by identifying a user's location. They may leverage databases that map IP addresses to geographical location and thereby prevent authentication attempts, or require more robust verification, when a user's location changes by a configured distance. Unfortunately, such mechanisms may be less effective when CGN is widely deployed.

Our approach can offer a lightweight, secondary factor that simply indicates if a client is located within a given source network (e.g., inside the LAN serviced by a given residential router) or if it is the same physical device (e.g., for a laptop or mobile device that changes networks). In such circumstances, the SDN controller can effectively act like a password manager by tracking secondary factors for a user across infrastructure.

6.2 Impact of VPN Server Provisioning

Enterprise networks may apply different architectures to deploy their VPN services. Enterprise networks may deploy their VPN services on their network gateways or devices that handle all the network's traffic. These devices can be purpose-built for VPNs. For example, the Cisco AST 1000 Series Embedded Processors achieves IPSec throughput up to 78 Gbit/s [48,49]. The downside associated with in-line hardware is that the service subscription for VPNs and the system's capital costs can be considerable [47].

Enterprises can also host VPN servers inside their networks using existing server infrastructure or other physical machines. Most commodity servers lack the hardware designed for VPN services. Pudelko et al. [53] indicate open source VPN servers on commodity servers have poor throughput compared to dedicated hardware. A Windows 2008 server can achieve gigabit throughput [50]. Other servers can achieve higher throughput [51] and a multi-core Linux machine can achieve 6.1 Gbps [52].

Our in-lab environment experiments in Sect. 5.3 show the potential bottlenecks associated with VPN gateways on general purpose systems. These results demonstrate the ability of our approach to remove such bottlenecks.

7 Conclusion

This work explores the roles that VPNs play in organizational security. With the rise of application-layer encryption and authentication, the secure tunneling features of VPNs are increasingly redundant. However, VPN tunnels are still

useful in simplifying perimeter-based access control by allowing authenticated remote users to bypass perimeter policies and interact with insider infrastructure.

We analyze the access control capabilities of VPNs and propose a new lightweight method. It requires no additional network infrastructure and removes the performance bottleneck on a VPN server, as well as eliminating redundant encryption and unnecessary packet header overheads. Our method is based on the SDN paradigm and gives clients the choice to implement a persistent identity on a per-application basis. We create `iptables` modules, which are required on the server side, to support our protocol. Our evaluation results show the method to be effective and lightweight.

Acknowledgements. This material is based upon work supported by the National Science Foundation under Grant No. 1651540.

References

1. Richter, P., et al.: A multi-perspective analysis of carrier-grade NAT deployment. In: ACM Internet Measurement Conference, pp. 215–29 (2016). https://doi.org/10.1145/2987443.2987474
2. Carrier-Grade-NAT (CGN) Deployment Considerations (2021). https://tools.ietf.org/id/draft-nishizuka-cgn-deployment-considerations-00.html
3. Atkinson, R.: Security Architecture for the Internet Protocol. RFC 1825, Internet Engineering Task Force (1995)
4. Sandvine Releases 2019 Global Internet Phenomena Report (2019). https://www.sandvine.com/press-releases/sandvine-releases-2019-global-internet-phenomena-report
5. Rekhter, Y., Moskowitz, B., De Groot, G.: Address Allocation for Private Internets. RFC 1597, Internet Engineering Task Force (1994)
6. Kreibich, C., Weaver, N., Nechaev, B.: Netalyzr: illuminating the edge network. In: ACM Internet Measurement Conference, p. 246 (2010). https://doi.org/10.1145/1879141.1879173
7. Mandalari, A., Lutu, A., Dhamdhere, A., Bagnulo, M., Claffy K.: Tracking the Big NAT across Europe and the U.S. ArXiv:1704.01296 (2017). arXiv.org, http://arxiv.org/abs/1704.01296
8. Livadariu, I., Benson, K., Elmokashfi, A., Dhamdhere, A., Dainotti, A.: Inferring carrier-grade NAT deployment in the wild. In: IEEE Conference on Computer Communications, pp. 2249–2257 (2018). https://doi.org/10.1109/INFOCOM.2018.8486223
9. Global Security Appliance Market Share 2012–2020 (2021). https://www.statista.com/statistics/235347/global-security-appliance-revenue-market-share-by-vendors/
10. Cloudflare Blocking My IP? (2021). https://community.cloudflare.com/t/cloudflare-blocking-my-ip/65453
11. Verizon to Launch 5G Residential Broadband Services in up to 5 Markets in 2018 (2021). https://www.verizon.com/about/news/verizon-launch-5g-residential-broadband-services-5-markets-2018

12. Amazon Simple Email Service Classic (2021). https://docs.aws.amazon.com/ses/latest/DeveloperGuide/
13. FCC Fines Verizon $1.35 Million over 'Supercookie' Tracking. https://www.theverge.com/2016/3/7/11173010/verizon-supercookie-fine-1-3-million-fcc
14. Perkins, C.E.: Mobile IP. IEEE Commun. Mag. **35**(5), 84–99 (1997). https://doi.org/10.1109/35.592101
15. Simpson, W.: IP in IP Tunneling. Request for Comments, RFC 1853, Internet Engineering Task Force (1995)
16. Neuman, C., Ts'o, T.: The Kerberos Network Authentication Service (V5). RFC 1510, Internet Engineering Task Force (1993)
17. Craven, R., Beverly, R., Allman, M.: A middlebox-cooperative TCP for a non end-to-end internet. In: ACM SIGCOMM Conference, pp. 151–162 (2014). https://doi.org/10.1145/2619239.2626321
18. Gont, F., Atkinson, R., Pignataro, C.: Recommendations on Filtering of IPv4 Packets Containing IPv4 Options. Request for Comments, RFC 7126, Internet Engineering Task Force (2014)
19. Open VSwitch (2021). https://www.openvswitch.org/
20. Cisco-Security-Manager-4-1 (2021). https://www.cisco.com/c/en/us/obsolete/security/cisco-security-manager-4-1.html
21. Bommareddy, S., Kale, M., Chaganty, S.: VPN Device Clustering Using a Network Flow Switch and a Different Mac Address for Each VPN Device in the Cluster. US6772226B1 (2004). https://patents.google.com/patent/US6772226B1/en
22. Coronavirus Challenges Remote Networking (2021). https://www.networkworld.com/article/3532440/coronavirus-challenges-remote-networking.html
23. Booth, S., Zorn, G., Patel, B., Aboba, B., Dixon, W.: Securing L2TP Using IPsec. Request for Comments, RFC 3193, Internet Engineering Task Force (2001)
24. Atkinson, R., Kent S.: IP Authentication Header. RFC 2402, Internet Engineering Task Force (1998)
25. Kent, S., Atkinson R.: IP Encapsulating Security Payload (ESP). RFC 2406, Internet Engineering Task Force (1998)
26. Nordmark, E, Bagnulo, M.: Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533, Internet Engineering Task Force (2009)
27. Moskowitz, R., Nikander P.: Host Identity Protocol (HIP) Architecture. RFC 4423, Internet Engineering Task Force (2006)
28. Estes, A.: The Dangers of Supercookies (2011). https://www.theatlantic.com/technology/archive/2011/08/dangers-supercookies/354297/
29. MacFarland, D., Shue, C, Kalafut, A.: Characterizing optimal DNS amplification attacks and effective mitigation. In: Passive and Active Measurement Conference, pp. 15–27 (2015). https://doi.org/10.1007/978-3-319-15509-8_2
30. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008). https://doi.org/10.1145/1355734.1355746
31. Komu, M., Sethi, M., Beijar, N.: A survey of identifier-locator split addressing architectures. Comput. Sci. Rev. **17**, 25–42 (2015). https://doi.org/10.1016/j.cosrev.2015.04.002
32. Netfilter/Iptables Project Homepage - The "Xtables-Addons" Project (2021). https://www.netfilter.org/projects/xtables-addons/index.html
33. Troubleshooting (2021). https://sendersupport.olc.protection.outlook.com/pm/troubleshooting.aspx
34. Prevent Mail to Gmail Users from Being Blocked or Sent to Spam - Gmail Help (2021). https://support.google.com/mail/answer/81126

35. Understanding the Cloudflare Security Level (2021). https://support.cloudflare.com/hc/en-us/articles/200170056-Understanding-the-Cloudflare-Security-Level
36. Malis, A., Lin, A., Heinanen, J., Gleeson, B., Armitage, G.: A Framework for IP Based Virtual Private Networks. RFC 2764, Internet Engineering Task Force (2000)
37. Access Control - Apache HTTP Server Version 2.4 (2021). https://httpd.apache.org/docs/2.4/howto/access.html
38. Benefits Of A VPN (2021). https://www.forbes.com/sites/tjmccue/2019/06/20/benefits-of-a-vpn/
39. Benefits of a VPN You Might Not Know About (2021). https://us.norton.com/internetsecurity-privacy-benefits-of-vpn.html
40. Dynamic IP Denylisting with NGINX Plus and Fail2ban (2021). https://www.nginx.com/blog/dynamic-ip-denylisting-with-nginx-plus-and-fail2ban/
41. Google Transparency Report (2021). https://transparencyreport.google.com/https/overview?hl=en
42. CUPS Plenary (2021). https://ftp.pwg.org/pub/pwg/liaison/openprinting/presentations/cups-plenary-may-18.pdf
43. What Is a Reverse Proxy Server? (2021). https://www.nginx.com/resources/glossary/reverse-proxy-server/
44. Francisco, Shaun Nichols in San. Corporate VPN Huffing and Puffing While Everyone Works from Home over COVID-19? You're Not Alone, Admins (2021). https://www.theregister.com/2020/03/11/corporate_vpn_coronavirus_crunch/
45. Comparing TCP performance of tunneled and non-tunneled traffic using OpenVPN (2021). https://www.os3.nl/_media/2010-2011/courses/rp2/p09_report.pdf
46. Liu, Y., Shue, C.: Beyond the VPN: practical client identity in an internet with widespread IP address sharing. In: IEEE Conference on Local Computer Networks, pp. 425–428 (2020). https://doi.org/10.1109/LCN48667.2020.9314846
47. Savings Calculator, Pulse Secure (2021). https://www.pulsesecure.net/savings-calculator/
48. Raumer, D., Gallenmuller S., Emmerich, P., Mardian L., Carle, G.: Efficient serving of VPN endpoints on COTS server hardware. In: IEEE International Conference on Cloud Networking (Cloudnet), pp. 164–169 (2016). https://doi.org/10.1109/CloudNet.2016.25
49. Cisco ASR 1000 Series Embedded Services Processors Data Sheet (2021). https://www.cisco.com/c/en/us/products/collateral/routers/asr-1000-series-aggregation-services-routers/asr-1000-series-embedded-services-ds.html
50. IP Security Features. Intel Ethernet Server Adapters (2021). https://docplayer.net/20618334-Ip-security-features-intel-ethernet-server-adapters.html
51. Han, S., Jang, K., Park, K, Moon, S.: PacketShader: A GPU-Accelerated software router. In: ACM SIGCOMM Conference, p. 195 (2010). https://doi.org/10.1145/1851182.1851207
52. Dobrescu, M., et al.: RouteBricks: exploiting parallelism to scale software routers. In: ACM Symposium on Operating Systems Principles, p. 15 (2009). https://doi.org/10.1145/1629575.1629578
53. Pudelko M., Emmerich, P.: Performance analysis of VPN gateways. In: IFIP Networking Conference (Networking), pp. 325–333 (2020)
54. VPN Risk Report - Cybersecurity Insiders|Industry Report (2021). https://info.zscaler.com/resources-industry-reports-vpn-risk-report-cybersecurity-insiders

55. Initial Credentials - MIT Kerberos Documentation (2021). https://web.mit.edu/kerberos/krb5-latest/doc/appdev/init_creds.html
56. DeCusatis, C., Liengtiraphan, P., Sager, A., Pinelli, M.: Implementing zero trust cloud networks with transport access control and first packet authentication. In: IEEE International Conference on Smart Cloud (SmartCloud), pp. 5–10 (2016). https://doi.org/10.1109/SmartCloud.2016.22
57. Hauser, F., Haberle, M., Schmidt, M., Menth, M.: P4-IPsec: site-to-site and host-to-site VPN With IPsec in P4-Based SDN. IEEE Access **8**, 139567–139586 (2020). https://doi.org/10.1109/ACCESS.2020.3012738