OnionVPN: Onion Routing-Based VPN-Tunnels with Onion Services

Sebastian Pahl
sebastian.pahl@hof-university.de
Hof University of Applied Sciences
Hof, Germany

Daniel Kaiser
dkaiser.lu@protonmail.me
Status Research & Development GmbH
Zug, Switzerland

Thomas Engel
thomas.engel@uni.lu
University of Luxembourg
Esch-sur-Alzette, Luxembourg

Florian Adamsky
florian.adamsky@hof-university.de
Hof University of Applied Sciences
Hof, Germany

ABSTRACT

Virtual Private Networks (VPNs) provide confidentiality and hide the original IP address. Although many VPN providers promise not to record user activity, several media reports of data breaches show that this is often not true. Tor, on the other hand, allows anonymous communication using onion routing and takes privacy and anonymity seriously, but at the cost of performance loss. What is missing is a sweet spot between VPNs and anonymization networks that supports bulk downloads and video streaming but provides countermeasures against untrusted VPN providers and Autonomous System (AS)-level attackers.

In this paper, we present OnionVPN, an onion routing-based VPN tunnel, that provides better bulk transfer performance than Tor and offers additional security features over a VPN: (1) intermediate VPN nodes see only encrypted traffic, (2) protection against AS-level attackers with a new path selection algorithm, and (3) onion services with a novel cryptographic NAT traversal algorithm using the Noise protocol framework. We analyze 118 VPN providers, systematically compare them to our requirements and show that OnionVPN is currently possible with three VPN providers. An alternative to Tor for bulk traffic could relieve the Tor network and provide a better experience for other users who need higher privacy and anonymity features.

CCS CONCEPTS

• **Security and privacy → Network security**.

KEYWORDS

vpn, onion-routing, anonymisation networks

ACM Reference Format:
Sebastian Pahl, Daniel Kaiser, Thomas Engel, and Florian Adamsky. 2024. OnionVPN: Onion Routing-Based VPN-Tunnels with Onion Services. In *Proceedings of the 23rd Workshop on Privacy in the Electronic Society (WPES '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3689943.3695043

WPES '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1239-5/24/10
https://doi.org/10.1145/3689943.3695043

1 INTRODUCTION

Virtual Private Network (VPN) protocols such as Internet Protocol Security (IPsec), OpenVPN[1], and WireGuard [1] provide an encrypted network tunnel to other endpoints. Commercial providers offer off-the-shelf VPN solutions to avoid setting up a VPN server. There are manifold reasons for using such services. First, it provides an encrypted tunnel, which is especially helpful in untrusted environments (e. g., free WLAN). Second, VPN providers can be used to bypass geo-blocking or government firewalls. Third, VPN protocols offer higher throughput than anonymization networks such as Tor and the Invisible Internet Project (I2P).

Without full control of a VPN server, the user must trust both the VPN provider and the data center where the server is located. Commercial VPN providers typically promise not to record any user activity. However, several media reports [2]–[5] from data breaches show that this is often untrue. Ramesh *et al.* [6] also show that traffic leaks during VPN tunnel failure are possible and expose sensitive user data. In a recent work, Xue *et al.* [7] revealed a design flaw of most VPN providers that allowed an adversary to force the victim to send the traffic outside the VPN tunnel. Both attacks would not be possible with OnionVPN.

On the other hand, anonymization low-latency networks such as Tor [8] and I2P [9] offer high anonymity, but with a performance loss. The scientific community works hard to improve the throughput of Tor [10]–[14], but there are still performance issues such as the cross-circuit interference (CCI) problem [15]. Additionally, through its circuit scheduling algorithm, Tor prefers bursty traffic, such as web browsing, over bulk traffic, such as file-sharing [16]. The use of file-sharing protocols (e. g., BitTorrent) over Tor can potentially compromise the anonymity of the users. Tor only works with TCP based services.

What is missing is a sweet spot between VPN and anonymization networks, which supports bulk downloads but provides countermeasures against untrusted VPN providers and Autonomous System (AS)-level attackers. In this paper, we present OnionVPN, which offers better bulk transfer performance to Tor, with additional security and privacy properties to a VPN from an anonymization network. We use modern network separation technology (e. g., Linux namespaces) to improve security and avoid data leaks. Our contributions are twofold: First, we present a ready-to-use open-source Proof of concept (PoC) that implements onion routing with

[1] https://openvpn.net/

Linux namespaces and WireGuard. Second, we present a theoretical architecture and ecosystem similar to Tor that would include onion services. Concretely, we make the following contributions in this work:

(1) We developed an open-source PoC of OnionVPN based on WireGuard and with Linux network namespaces.
(2) We analysed 118 VPN providers and showed that three are suited for the use for OnionVPN and six providers partially.
(3) We evaluated the performance of OnionVPN and showed that it outperforms Tor for bulk traffic.
(4) We developed a new path selection algorithm to avoid AS-level attackers.
(5) We show with our PoC that OnionVPN can have onion services as well.
(6) We designed a cryptographic Network address translation (NAT) traversal algorithm using the Noise protocol framework [17] that we used for onion services in OnionVPN.
(7) We designed an theoretical ecosystem for OnionVPN, that allows a wider adoption.

Outline. Section 2 provides background on VPN and onion routing. In Section 3, we describe our threat model. We divided the paper into two major parts: The first one is described in Section 4, in which we describe the technical details of OnionVPN and evaluate it in Section 5. The second one is Section 6, in which we describe a possible ecosystem for OnionVPN. Section 7 contains the discussion and limitations of OnionVPN and Section 8 discusses related work. We conclude in Section 9.

2 BACKGROUND

This section provides background on Tor and VPN, particularly WireGuard.

2.1 WireGuard

Most encrypted tunnel protocols nowadays are based on either Transport Layer Security (TLS) or IPsec. For example, OpenVPN is based on TLS 1.2 and uses a user-space TUN/TAP solution. TUN/-TAP are virtual network devices that are defined by software, in this case, OpenVPN. While TLS 1.2 has a long history of security and privacy issues [18], IPsec suffers from its complexity [19]. WireGuard, on the other hand, is a novel VPN protocol operating on Layer 3. WireGuard uses UDP as the basis; any in-order delivery and reliability of packets is the protocol's responsibility that goes over the tunnel. It is designed to run in the Linux Kernel[2], so it does not have to copy packets multiple times, which improves the performance significantly compared to user-space TUN/TAP solutions.

Complexity is an enemy of security. Therefore, WireGuard focuses on simplicity. The WireGuard implementation in the Linux Kernel has less than 4000 Lines of Code (LoC). WireGuard's protocol is based on the Noise Framework [17] and formally verified [20]–[22]. WireGuard does not support cipher and protocol agility, where sender and receiver negotiate the cipher suite. If security problems have been found in the underlying primitives, all endpoints must update to a newer version. WireGuard uses Curve25519 for Elliptic Curve Diffie-Hellman (ECDH), ChaCha20 [23] and Poly-1305 [24] for authenticated encryption, BLAKE2 [25] for hashing, and SipHash2-4 [26] for hashtable keys. Static public keys (static identities) must be manually exchanged *a priori*, similar to Secure Shell (SSH). Optionally, an additional pre-shared key can be used to make WireGuard post-quantum secure. Recently, Hülsing *et al.* [27] introduced a post-quantum WireGuard variation, which provides post-quantum confidentiality and authentication.

WireGuard has additional security properties. Due to the previously exchanged static identities, a WireGuard instance does not respond to any non-encrypted messages. This property makes an exposed WireGuard endpoint stealthy protecting it from protocol probing or port scanning. Additionally, all packet fields have a fixed length, which hinders parsing related memory corruption attacks.

[2]Besides the Kernel implementation, there are also user-space implementations in Rust, Go, and Haskell. This allows WireGuard to run on other Operating Systems (OSs) such as Windows, MacOS, iOS, and Android.

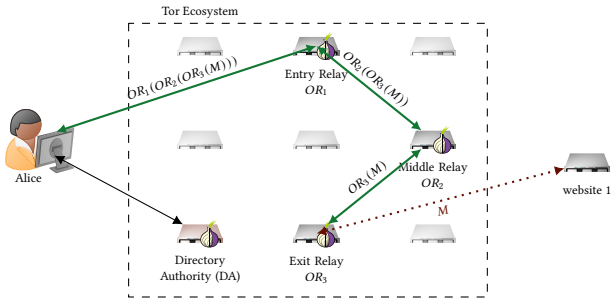


Figure 1: Example of how Onion Routing works in Tor. Alice's Tor client first contacts the Directory Authority (DA) to get a list of Onion Routers (ORs). Then it encrypts her message M three times in reverse order and sends it to the first Tor relay, also referred to as the Tor entry relay or Guard. The straight green line contains traffic that Tor encrypts, and the dotted red line is traffic that might or might not be encrypted.

2.2 Onion Routing

Onion Routing was first introduced by Goldschlag *et al.* [9]. It works by encrypting a message multiple times, like the layers of an onion, as depicted in Figure 1. This encrypted message is sent to a network of relays. Each relay removes one layer of encryption and forwards the message to the next hop until all layers have been removed. Usually, the message passes three relays. In backwards direction, each relay adds its layer of encryption until the message reaches the original sender.

There is no direct link between the sender and the destination. Therefore, it provides an anonymous communication channel. Only the entry hop knows the sender, and only the exit hop sees the cleartext message and knows the destination. Nowadays, most web traffic is TLS protected. In this case the exit node cannot see contents but protocol metadata.

2.3 Onion Services

Usually, only the user is anonymous to the destination; in the case of Onion Services, the destination is also anonymous to the user. Tor has deployed onion services since 2004 [28]. To connect to a service, users need a so-called onion address [29], which is a base32-encoded URL with the public identity key of the service, a version field and a checksum, which ends with `.onion`. Additionally, a piece of software that understands such a URL.

A service publishes introduction points in a distributed hash table (DHT) to each point they build a circuit. A client uses a URL, goes to this DHT and asks for these points. Before connection to the services it choses a rendezvous point and builds a circuit to it. Afterwards, it chooses an introduction point, builds a circuit to it and introduces itself. The client sends the rendezvous point and a secret string to the service. The service builds a circuit to this point. The rendezvous point verifies the client's secret. If the secret matches, both circuits will be connected, and both parties will be able to talk to each other.

Besides hiding the IP address of the destination, onion services provide:

(1) *Service Authentication*: If a user connects to an onion service, it can be sure that the URL and data from the service belongs to it. An adversary cannot pretend to be the service without the user noticing.
(2) *End-to-end encryption*: After the client and service are connected to the rendezvous point the traffic is encrypted. The encryption protects both ends, and no traffic leaves the Tor network. But an attacker can still fingerprint an circuit [30] and can distinguish between a normal and service circuit, but defenses against this attack exists [31].
(3) *NAT punching*: An onion service does not need to open a port on a public interface. It initiates communication with the Tor network and offers the onion service inside of the network. No port forwarding or NAT punching needed.
(4) *Censorship resistance*: A client with knowledge of the onion address can look up the defined introduction points in the distributed hash table. Multiple relays that change over time store this information. However, without the onion address, these relays cannot find out for which service they provide this information. It is only possible to censor certain services if all the relays involved work together.
(5) *Client traffic is not trackable*: Every time a client connects to an onion service, a new rendezvous point is exchanged. This makes it hard to track traffic from clients to a particular service.

3 THREAT MODEL

We consider a similar threat model as Tor, where it is impossible to defend against a global passive adversary but an adversary with a limited network view, such as:

(1) A *Malicious Provider* that can monitor all traffic. With OnionVPN, providers do not have any personally identifiable information (PII), and every hop on the path has a distinct provider, which makes it impossible to correlate sender and receiver directly to each other. Except the provider used for entry can observe the real IP address of the user. We can fully defend against such an adversary.
(2) A malicious AS inherits the capabilities of a *Malicous Provider*. In contrast to a provider, an AS can be on both ends of the network path, which enables traffic correlation attacks, but our path selection algorithm limits or prevents this possibility.

In the event of a malicious provider or AS, OnionVPN offers sender anonymity and with onion services receiver anonymity. The threat model excludes providers and ASs that are controlled by the same entity, which are referred to as siblings. Because this property breaks the assumption that the adversary cannot observe both ends. We also exclude fingerprinting attacks from adversaries, that e. g., could identify the web browser or OS of the sender. Tor Browser[3] or Mullvad Browser[4] protect against browser fingerprinting attacks. Other types of fingerprinting attacks like circuit or website fingerprinting are not easily preventable. We also do not consider censorship-resistance.

4 TECHNICAL DETAILS OF ONIONVPN

OnionVPN allows the creation of multi-hop circuits based on VPN protocols. The only requirement for the VPN protocol is to transport itself, which applies to all popular VPN protocols, including IPsec, OpenVPN, and WireGuard. We are not aware of any VPN protocol that does not have this property. This makes it possible to nest a tunnel inside another tunnel. For example, if there are three relays, one can build a tunnel to the first hop, then over this hop a second tunnel to the next and so on. We assume intermediate relays run only standard software.

A naive approach is to define a set of firewall rules to route packets through a chain of three network interfaces. Each interface is a configured tunnel of a VPN protocol. The packet must go through the interface of the exit relay, then the interface of the middle relay and then the interface of the entry relay. Although this is possible, it is increasingly challenging to manage and error-prone. A minor misconfiguration (e. g., IP address collision) could leak sensitive user information [7], [32]. To solve this problem, our PoC makes use of Linux network namespaces. WireGuard [33], IPSec (strongSwan [34]), and OpenVPN integrate into the Linux network namespaces infrastructure. But IPSec and OpenVPN need virtual Ethernet (`veth`) interface pairs to chain the namespaces together. Other OSs do not have network namespaces like Linux but could use firewall rules instead.

First, we describe how we build a circuit with Linux network namespaces.

4.1 Building a Circuit

Every process that is executed, by default, run in the default network namespace called the *root namespace*. A WireGuard interface remembers the network namespace in which it was created. All outgoing packets will arrive in the initial namespace if we move the WireGuard interface to another network namespace. For example, to build a one-hop tunnel, the following steps are necessary,

[3] https://www.torproject.org/de/download/
[4] https://mullvad.net/de/download/browser/windows

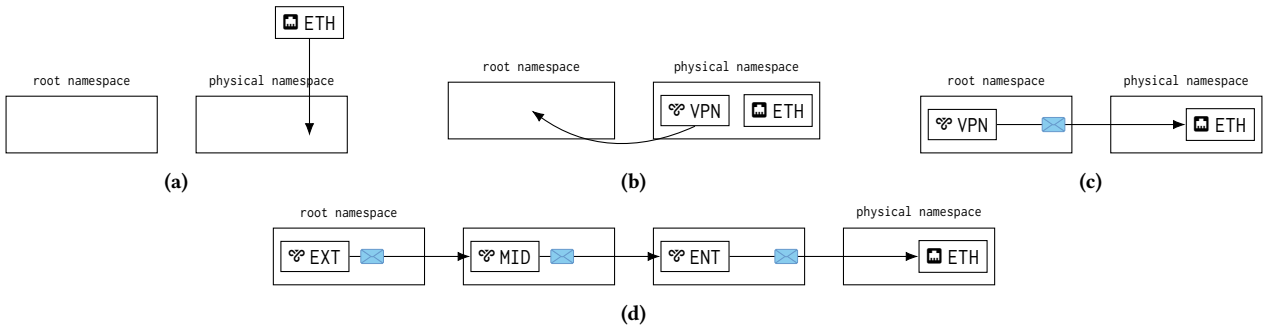


Figure 2: Example of how a circuit is build with WireGuard-based OnionVPN. In *(a)*, we move the physical network interface in its own network namespace. In *(b)*, we create the VPN inside the physical network namespace and move it then to the root namespace. In *(c)*, every process can tunnel their network traffic through the VPN interface. In *(d)*, the whole three-hop circuit is shown.

which are depicted in Figure 2: First, we move all physical interfaces to a new namespace, which we name *physical* (see Figure 2a). Second, we create a WireGuard interface in the physical namespace and add a default route to it. Third, we move it to the root namespace (see Figure 2b). When the WireGuard interface is in the root namespace, all processes can tunnel their traffic through that interface. Fourth, every outgoing packet arrives in the physical namespace and goes through the routing table of this namespace (see Figure 2c).

For a three-hop tunnel, we create two additional namespaces, *entry* and *middle*. Instead of moving the first WireGuard interface to the root namespace, we move it to the *entry* namespace. Then, we create a WireGuard interface in the *entry* namespace and move it to the *middle* namespace. Then, we create a WireGuard interface in the *middle* namespace and move it to the *root* namespace. All running processes must use the interface in *root* namespace to communicate with the Internet (see Figure 2d). WireGuard itself plans to implement onion routing, but at the time of writing, it is still on their to-do list [35]. If they do, we only need the *physical* and *root* namespace to isolate processes from the physical interfaces.

4.2 Path Selection

AS-level attackers are a real threat to the Tor network [36]–[38]. If an adversary can observe both ends of the path, a traffic correlation attack can deanonymize users. Tor's path selection algorithm tries to prevent such a situation and, due to relay weights, also improves the user's performance. However, it does not actively prevent a path from entering or exiting a destination with distinct Autonomous System Number (ASN) [39] since Tor's path selection algorithm does not take ASs into account.

Before describing our path selection algorithm, we analyze the AS diversity of Tor and VPN providers to get a better understanding.

4.2.1 AS Diversity of Tor and VPN Providers. We collected the IP addresses of 40 000 proxy servers from ten VPN providers and Tor. We selected VPN providers based on the following criteria: the number of available servers and if the VPN provider offers a server list. The result is the following list of VPN providers: CyberGhost, ProtonVPN, Surfshark, Windscribe, OVPN, iVPN, Mullvad, NordVPN, PIA, and Ivacy. CyberGhost, ProtonVPN, Surfshark and Windscribe do not have publicly available server lists. We use a domain enumeration provider[5] to get an estimated list. For CyberGhost, we only get 10 000 servers, which limits the data source. Ivacy claims to have over 6500 servers, but its support site lists only 300. We have the complete server list for OVPN, iVPN, Mullvad, NordVPN and PIA. For Tor, we can use the latest consensus at that time (2023-09-15 at 00:00).

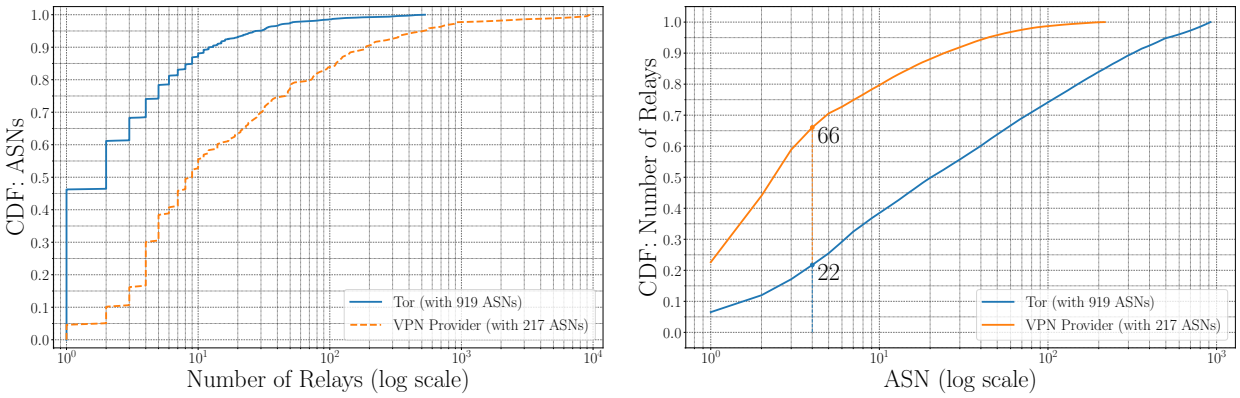
For each server with a unique IP address, we assign one ASN. We use the ASN database from the University of Oregon and their Route Views Archive Project[6]. We compare the ASN and country diversity for Tor and the VPN providers. Tor has 918 different ASNs, while all VPN providers combined have only 216. Similarly, for 16 bit subnet prefixes, where Tor has 2158 and VPN providers have 771. The top four ASNs for VPN providers contain 66 % of all servers. In contrast to Tor, where the amount of all relays belonging to the top four ASNs make up 22 % of all servers. Compare ASN diversity in Figure 3.

The country diversity is better for the VPN providers, but to resolve the IP address to country we use a database. Keep in mind, that such databases are not necessarily reliable [40]. According to our data collection (see Section 5), most VPN servers are in the US.

4.2.2 Algorithm. Due to the low diversity, our path selection algorithm must be AS-aware; therefore, we adapted an AS-aware path selection algorithm from Edman and Syverson [39].

VPN providers' AS diversity is not as good as Tor's, and every provider can be considered one overlay AS. In that sense, an overlay AS is very similar to what Tor calls relay families. VPN providers have more servers than Tor. In the case of Mullvad (Swedish VPN provider) each server has on average 10 Gbit/s [41]. The available bandwidth per server does not vary as much, making it unnecessary to select servers by bandwidth weight or relay weight. Instead, they can be selected uniformly at random after considering other attributes. We will show later in Section 5.2 a detailed

[5] https://www.abuseipdb.com/whois/
[6] ftp://archive.routeviews.org/route-views4/bgpdata/2023.10/RIBS/rib.20231005.1000.bz2



(a) (b)

Figure 3: Comparison of the ASNs diversity of Tor and ten VPN providers. Tor has 918 and the VPN providers have 216 ASNs. Figure 3a compares the distribution of the number of relays per ASN. It shows that Tor utilizes more ASNs with fewer relays. On the other hand, VPN-Prvoider have fewer ASNs but more relays in a single AS. Figure 3b compares the cumulative number of relays per ASN. The top four ASNs contain 66 % of all servers compared to Tor's top four ASNs, which amount to 22 % of all relays.

performance measurement. In contrast to Tor, every server is an exit relay.

Each network path must consist only of sets of non-overlapping ASs or overlay ASs. At least the set from client to entry and exit to destination should not intersect. This makes it impossible for one entity to observe and compromise both ends. It is necessary because of the low AS diversity and top-heavy providers with many servers.

ASs path inference or measuring is a computationally expensive operation [39]. In the Tor network, measuring the network path from a particular relay to a destination is only possible if you are the operator. Due to how OnionVPN works, measuring the network forward path is possible but not the backward path. However, it can be inferred by a complete AS path inference algorithm or an approximation.

The path selection algorithm is shown in Listing 1. The algorithm first selects a VPN provider from the set of VPN providers uniform at random. The algorithm selects an entry node from that provider and removes it from the list. The same procedure applies to the middle and exit nodes. The lambda function `has_overlap()` implements a AS path inference algorithm [39], [42].

```
def path_selection_default(nodes: Dict, has_overlap, client_ip,
    destination_ip):
    nodes = nodes.copy()
    providers = list(nodes.keys())

    # (1) Select an entry node
    entry_provider = random.choice(providers)
    entry_node = random.choice(nodes[entry_provider])
    providers.remove(entry_provider)

    # (2) Select a middle node
    middle_provider = random.choice(providers)
    middle_node = random.choice(nodes[middle_provider])
    providers.remove(middle_provider)

    # (3) Select an exit node
    while True:
        exit_provider = random.choice(providers)
        exit_node = random.choice(nodes[exit_provider])
        if has_overlap(client_ip, entry_node, exit_node,
    destination_ip):
            continue
        else:
            break

    return [entry_node, middle_node, exit_node]
```

Listing 1: Simplified Path selection Algorithm in Python.

4.3 Onion Services

A feature of Tor is onion services [43], where both client and server are anonymous. This section describes how onion services can be integrated into OnionVPN.

Due to no additional software on the relays, connecting two endpoints yields the same challenges that two peers have to establish a direct connection on the Internet. One problem that might occur is that the client and server are behind a router with NAT. The number of intermediate hops does not matter. Usually, VPN providers enable users to offer services by port forwarding. To do so, the user connects to a relay and opens a port via API call (or the provider's web interface) to forward incoming traffic to a service that is listening locally.

In the Tor network, connecting clients and services is not directly visible to an outside observer, but there are circuit fingerprinting attacks [30] to distinguish onion services from standard circuits. This can help to deanonymize users or operators. In our case, this is directly observable when connecting both circuit ends, because we use a WireGuard tunnel which would reach a non-default port of the VPN provider's relay.

4.3.1 Port Forwarding. Port forwarding is the easiest way to connect the client and server, and it requires no additional software.

Unfortunately, VPN providers that have supported port forwarding no longer offer it because of abuse [44], [45]. Abuse is easily possible because a web server can be reached from a normal web browser with only a DNS entry. An attack [46] against port forwarding is known to leak the actual IP address of a user. This attack is rendered useless if the VPN is implemented via Linux network namespaces. Next, we describe our NAT traversal algorithm to enable onion services in OnionVPN.

4.3.2 Introduction Point Proxy. Without port forwarding, we need another way to bypass NAT. Other protocols, such as WebRTC and Voice-over-IP (VoIP), use Interactive Connectivity Establishment (ICE) [47]. However, ICE is not suitable for our needs. First, ICE needs an already established connection to signal the necessary information, which we do not have with onion services. Second, ICE could reveal the actual IP address when not using Linux network namespaces. Third, if symmetric NAT[7] [48] is used on one side, ICE would evade to a Traversal Using Relays around NAT (TURN) server. For these reasons, we have developed a NAT traversal algorithm, which we call Introduction Point Proxy (IPP), that can be used for onion services. Our design goal is to allow both ends to connect to each other but make arbitrary data exchange difficult during the connection phase over IPP. Client and service need to exchange at least the following information:

- The IP address and port of the IPP; and; public key of the service.
- The public IP address, port and public key of the client.

Our protocol is based on the Noise Framework [17] and we use `Noise_{IK,KK}_25519_ChaChaPoly_BLAKE2s`. These are two protocol names from Noise which only differ in the handshake pattern and comparable to TLS cipher suites. Each part separated by an underscore allows the use of a different algorithm, e. g., the first is for the handshake and the last for the hash function. `Noise` is just a constant prefix. `{IK,KK}` are our handshake patterns. `IK` is used for the IPP and `KK` is used for the peer-to-peer communication between the two hops. If the first letter is an *I*, then the public static key of the sender will be immediately transmitted to the destination in cleartext. If the first letter is a *K*, then the public static key of the sender is known to the destination. If the second letter is a *K*, then the public static key of the destination is known to the sender. Depending on the handshake pattern, the payload security and identity hiding properties are different. For these two patterns the handshake requires one message from the sender and one from the destination. Both have the same payload security properties. `25519` is the elliptic curve that is used for the ECDH. `ChaChaPoly` provides authenticated encryption. `BLAKE2` is the hash function to ensure data integrity.

The first message from the source (service or client) to the IPP is prone to a Key-Compromise Impersonation (KCI) if the IPPs long-term private key has been compromised, as well as the first message of the second handshake from client to service if the client is compromised. This is not a problem in these two cases because clients create a new key pair for each connection attempt, and service and IPP should update their key pairs regularly. The same applies if we look at it from the point of view of the destination, but additionally, the message is prone to a replay attack. The second message from the destination (IPP or service) to the source (client or IPP) has weak forward secrecy if the sender's long-term private key has been compromised. This is the same situation as the first message. After that, the transport phase starts with authentication resistant to KCI and strong forward secrecy. You can find the data flow of IPP in Figure 4:

(1) At first, the service does an *IK* handshake through OnionVPN hops with the IPP. We assume a list of IPPs, each with a public key available to the service. It will transmit its static public identity key to the proxy immediately (*I*), and due to the list it knows, the public key of the proxy (*K*).
(2) It will instruct the proxy to listen for client connections. This logic is implemented on the application layer, not on the transport layer. The proxy will use the service's public key an identifier for the service. This public key is authenticated due to the Noise protocol. The service will wait for client connections and publish its connection information as a URL and/or in a directory service.
(3) The client will use the connection information from the service to connect to the proxy. It also does an *IK* handshake with the proxy. We assume a list of proxies, each with a public key available to the client.
(4) The client will send an application-level connect command to the proxy using the service's public key.
(5) The proxy will send the service's public IP address and port to the client.
(6) The proxy will send the client's public IP address, port, and public key to the service.
(7) Both client and service will send empty UDP packets to each other to punch a hole through their NAT.
(8) Both will perform another Noise *KK* handshake. To ensure that both parties have the same view of the connection, each handshake packet must contain all prologue information: IP address and port of the service and IP address, port, and public key of the client. Each peer validates the received information with its own view and fails the handshake if there is a difference. This mechanism is supported by the Noise protocol.
(9) Further information can now be exchanged via the established peer-to-peer channel.
(10) Each party configures a WireGuard interface over the same connection. Everything afterwards is service-dependent, but the communication will take place over a WireGuard tunnel. Both ends must configure a keep-alive signal that sends a packet in certain intervals to keep the tunnel alive. With WireGuard, both interfaces can configure `persistent_keepalive=25`, to send a UDP packet every 25 s.

A malicious IPP can deny the service, but that is not preventable. Such a service must be removed from the available list. After a client's connect attempt, it can send an alternate IP address, port, or public key to the service. This is also not preventable and is effectively a Denial-of-Service attack. It can send an arbitrary IP address

[7] Symmetric NAT is a network configuration that assigns a unique external IP address and port combination to each outbound connection from a device, making it difficult for unwanted incoming data to reach that device.

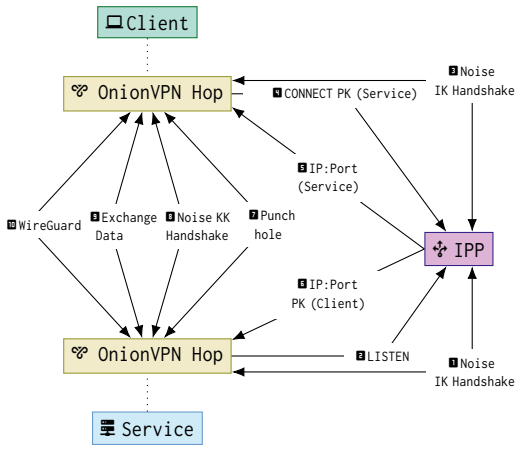


Figure 4: Shows a data flow graph of our NAT traversal algorithm IPP.

and port to the client. However, the client can detect this during the second peer-to-peer handshake (see step 8 in Figure 4) because we assume the service's public key is known to the client over a separate trusted channel. If the second handshake succeeded, the client can be sure that it connected to the correct service.

4.3.3 Internal Introduction Point Proxy. The IPP can be located inside of the VPN provider's network. This way, a malicious actor cannot abuse the IPP without accessing the VPN network first. On the other hand, this requires client and service to be connected to the same VPN provider. This approach is closer to how Tor offers onion services.

5 EVALUATION

This section describes the evaluation of the performance of OnionVPN and the suitability of VPN providers.

5.1 Suitable VPN Providers

There are hundreds of VPN providers, but only a subset meets our needs. OnionVPN has the following anonymity and technical requirements for a suitable VPN provider:

(1) *No PII required to get access*: An account is just a random number or just a username and password.
(2) *At least one anonymous payment option*: Payment is possible via cash, voucher or privacy-preserving cryptocurrency (like Monero, Dash or Zcash).
(3) *Registration and payment is possible via Tor*.
(4) *They support WireGuard*: This is important for our technical implementation but not a general limitation.
(5) *Available server list*: A list of all servers with each server's public key and IP address. This list is important for an offline path selection algorithm (see Section 4.2).

Other properties could be included, such as not logging unnecessary data, giving law enforcement access to their network, or trying not to mislead users with properties that a VPN cannot have. However, these are not directly measurable or ambiguous and will be excluded.

Table 1: List of all VPN providers that fully and partially suit our requirements for using OnionVPN.

Provider	Suitable	Countries	# of Server	Server List	WG	Access	Payment
IVPN	●	35	81	✓	✓	Random ID	Cash, Monero
OVPN	●	20	100	✓	✓	Credentials	Cash, Monero
Mullvad	●	41	667	✓	✓	Random ID	Cash, Monero, Voucher
Anonine	◐	32	43	✓	✓	E-Mail	Monero
Hide.me	◐	49	66	✗	✓	E-Mail	Monero
AirVPN	◐	23	250	✗	✓	E-Mail	Monero
ProtonVPN	◐	67	2914	✓	✓	E-Mail	Cash
VPN.ac	◐	21	N/A	✗	✓	E-Mail	Monero
Windscribe	◐	63	N/A	✗	✓	Credentials	Cash, Monero

In order to answer these questions we evaluate various providers. As a starting point, we use a list from a review website[8] and add other providers that are not yet included. We collect information about the provider by visiting each individual website.

Our list contains overall 118 VPN providers. Appendix A contains Table 2 that shows a list of all providers. We had to exclude 54 providers because they do not exist anymore, their website does not work, they offer only a mobile application or do not offer their service to non-business customers. Only three fulfill our needs: Mullvad, IVPN and OVPN, see Table 1. The first two offer usage without asking for PII by utilizing a random account number as an identifier and supporting anonymous payment methods (Cash and Monero). The last provider asks for a self-chosen username/password combination but no further PII and supports Monero. These providers have a filled circle in the table. Providers with a half-filled circle nearly support all requirements, e.g. ProtonVPN has a server list, WireGuard and an anonymous payment method but requires an email address for their account.

The most common payment methods are credit cards and PayPal, 39 support some non-privacy-preserving cryptocurrency, only nine support anonymous payment methods, 15 have a server list available, and 33 support WireGuard.

5.2 Performance Evaluation

This section evaluates OnionVPN's performance in a local and Internet environment. The local environment helps us understand the upper limits and removes performance differences between VPN providers. The Internet environment helps us understand the practical performance implications.

5.2.1 Experimental Setup and Methodology (Internet). In this setup we will use our PoC, which we named *Vad*, that is implemented in Python and supports multiple VPN providers. It implements our Path Selection Algorithm, can build circuits with multiple hops and supports Onion Services. The latter implements only the simplest connection method with IPP, more details in Section 6.1.2. You can find the source code at: https://github.com/iisys-sns/vad.

We use this PoC with three VPN providers, namely NordVPN, ProtonVPN, and Mullvad. We selected those VPN providers according to the technical requirements from Section 5.1. However, we ignore the anonymity requirements for these experiments because we emphasize the number of servers to evaluate the performance.

[8] https://vpn-services.bestreviews.net/vpn-providers

The client uses our path selection algorithm from Section 4.2. Additionally, we measure Tor against OnionVPN in specific scenarios to get further insides.

Every 5 min our measurement script builds a new circuit. The circuit consists of three random servers from the selected VPN provider in random order. WireGuard does not connect to another server unless we transfer data, so we send an ICMP echo request (ping) to activate the circuit. Afterwards, we record ten pings to a measurement server in Germany and run a traffic generator called `tgen` with a model comparable to the performance measurements of the public Tor network [49] and Shadow simulations [50]. Our measurement server is running Ubuntu 20.04 LTS with Kernel version 5.4.0-173 and has a 2.5 Gbit/s Internet connection.

We use the `tgen` model to take five measurements, each 1 min apart from the other, with the same circuit, similar to Shadow [50]. However, the public Tor network [50] takes one measurement every 5 minutes. Each of our measurements transfers a file with a random size of 50 kB, 1 MB, and 5 MB. We chose those file sizes to compare our results with the public Tor metrics. Even though we chose a random size, we ensured each file size was measured with the same probability.

However, the public Tor network and Shadow simulations do not use an equal probability. They transfer smaller file sizes more often. We use the same parsing and analysis tools from `tgentools` as Tor and Shadow. Additionally, we measured the goodput (the actual useful data without overhead) over 60 s with `iperf3` for 6 d every 12 min.

5.2.2 Experimental Setup and Methodology (Local). For our local measurements, we use a server with Arch Linux, Kernel 6.9.7-1, and an 8-core AMD Ryzen 7 3700X. The measurements are similar to the Internet measurements but take place in an emulated network with Linux network namespaces. For Tor, we use a 0.1 % network that is generated by `tornettools` in version 2.0.0 and uses only one client with seven relays. For OnionVPN, we configure a static circuit.

We build the network and then execute a script 30 times that performs the following actions: Run a traffic generator called `tgen` with a model comparable to the performance measurements of the public Tor network [49] and Shadow simulations [50]. We run the model 10 times, each measurement transfers a file of 50 kB, 1 MB, 5 MB, 10 MB, and 20 MB with equal probability. We use `iperf3` to measure the goodput over 60 s.

Tor and OnionVPN must connect or build the circuit during the first measurement. Tor will build a new circuit every ten minutes, but Tor can pre-build these and it will not have any influence on the measurements. OnionVPN has a static circuit, but with either `tgen` or `iperf3`, a new TCP connection will be used for every measurement. Tor might have a slight advantage here because it could use an already established connection between two ORs and the CCI problem is not present. Using Shadow simulations for Tor and OnionVPN would be better, but WireGuard is kernel-based, making it difficult to simulate. There are user-space implementations, but whether they would be compatible with Shadow is unclear.

We measure the following scenarios: no circuit latency and packet loss to get a theoretical upper limit; circuit round-trip time (RTT) of 352 ms and no packet loss; and; circuit RTT of 352 ms and 0.1 % packet loss. The latency is derived from the median latency from our Internet measurements. The packet loss is evenly distributed across all packets and the value is based on Verizon measurements [51].

5.2.3 Results and Discussion. Figure 6 shows the local performance measurements. First, the measurements in Figure 6a show that the upper bound for Tor with its end-to-end congestion control [52] can reach around 300 Mbit/s, whereas OnionVPN can reach up to 1100 Mbit/s. Second, the rest of the measurements in Figure 6b–h show that OnionVPN is faster than Tor, independent of the file size. However, if we add latency to our local measurements, one can see in Figure 8 that OnionVPN is faster for larger files starting with 5 MB. If we add packet loss, OnionVPN performance is overall worse than Tor's; see Figure 9. As a reminder, Tor builds its circuits with *cascading* TCP connections from client to entry node, from entry to middle, from middle to exit and from exit to destination. OnionVPN utilizes a *single* TCP connection to the destination, which is nested inside three WireGuard UDP packets. Thus, window size increases faster in Tor since we have multiple TCP connections between ORs; therefore, the window updates show a faster effect. Due to the end-to-end congestion control [52], Tor no longer has a fixed window size. However, the local measurements ignore network load and CCI [15], negatively impacting Tor's performance.

Figures 5e–g show the results for Internet measurements. OnionVPN is slower while transferring 50 kB but faster in the case of 1 MB and 5 MB (see Figure 5f and 5g).

It is slower for 50 kB because Tor has cascading TCP connections, while we only have one TCP connection from client to destination. A cascading TCP setup can scale its congestion window faster because of the faster feedback from its intermediate nodes. In our case, we need to wait for a complete circuit RTT until we can scale our congestion window, which results in slower transfer times for smaller files.

Our measurements with `iperf3` (goodput measurements) over 60 s in Figure 5a show that this is also true for larger transfers. These measurements show a median goodput of around 100 Mbit/s; 10 % of transfers have more than 300 Mbit/s. This is in contrast to Figure 5c, which shows goodput up to 2.5 Gbit/s, but this only measures short-time performance. Unfortunately, we have no comparison with the public Tor network since the Tor metrics project does not have performance measurements with more than 5 MB. Overall, OnionVPN is faster for larger transfers.

Tor has six vantage points for performance measurements in Germany, China, and the USA. We have only one vantage point in Germany. It needs to be clarified how well these vantage points from Tor are connected. Our vantage point has a 2.5 GB/s connection; we assume this is the same as Tors. Most of Tor's relays are located in Germany, and most of our servers are in the USA. Despite these differences, the circuit round trip time matches primarily except for the worst 8 % (see Figure 5d).

6 ONIONVPN ECOSYSTEM

The Tor community is doing a fantastic job establishing an ecosystem that involves thousands of volunteers. Tor is free of charge, therefore removes any payment traces that we have in OnionVPN. If a user wants to use a VPN provider, she/he needs to register an

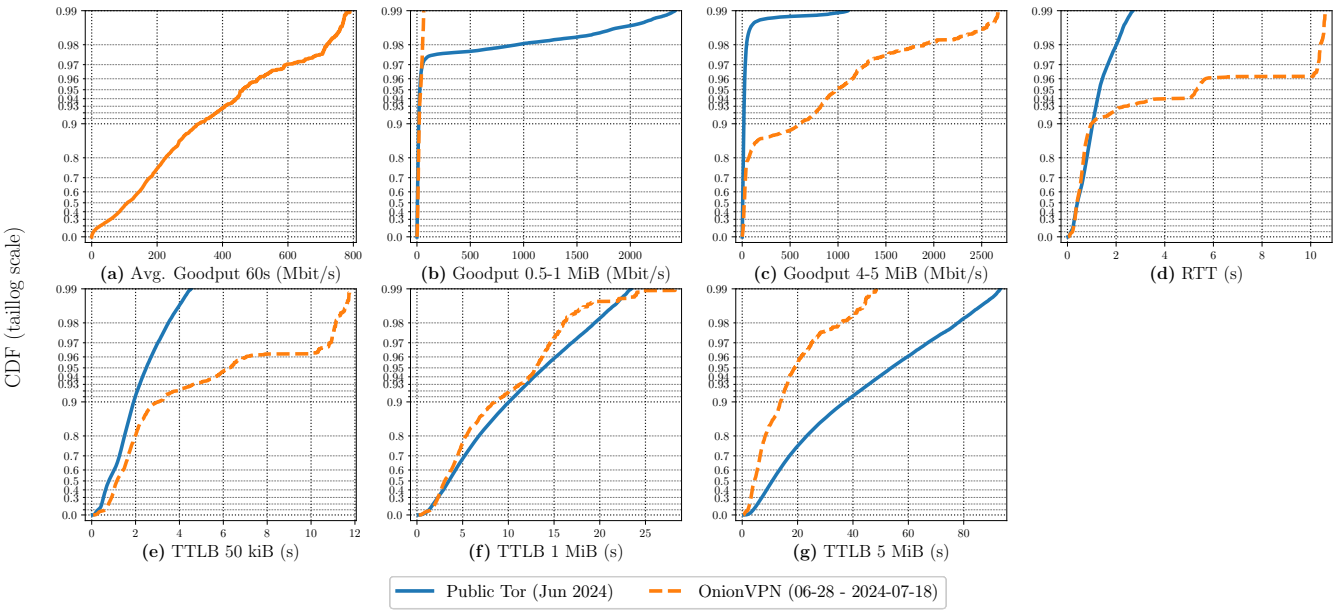


Figure 5: Internet measurements of Tor versus OnionVPN. (a) The Cumulative distribution function (CDF) of the average goodput over 60 s measured with `iperf3`. The goodput CDF while downloading between (b) 0.5 MB to 1 MB and (c) 4 MB to 5 MB; and; (d) the CDF of the circuit round trip time. The CDF transfer time in seconds of (e) 50 kB, (f) 1 MB and (g) 5 MB.

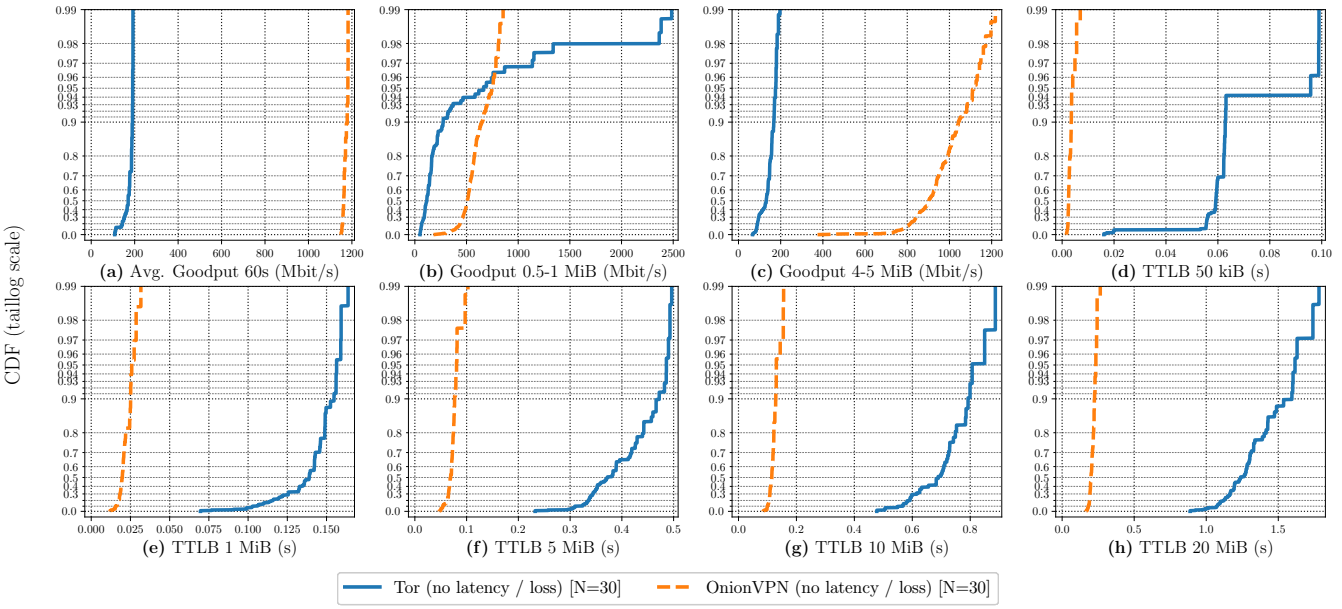


Figure 6: Performance measurements that compare Tor against OnionVPN in a local setup. This setup has no latency and shows the theoretical upper bound for both on one system. The first plot show the goodput over 60 s measured with `iperf3`. The next two plots the goodput between transferring 0.5 MB to 1 MB and 4 MB to 5 MB measured with `tgen`. Afterwards the time to last byte (TTLB) of various file sizes.

account. An account might allow one or more devices access to the service. Even in the best case where it is only an account number, it forms a long-term pseudonym. The user will make regular payments to extend access. That makes it easier for a malicious provider to track a user. We will introduce a system that removes this long-term pseudonym and makes it usable for OnionVPN. In the best case, VPN providers would support such a system. This system would also improve the situation for regular VPN users.

Instead of having a long-term account, users buy a public key slot using an anonymous payment method. Your device will generate the private key, and the public key will be inserted into this slot during the payment process. The public key is now the account number. This key pair does not allow direct access to the VPN service. Instead, it provides another empty public key slot that will allow access. We will call the first key pair *long term key pair* or *account key pair* and the second *ephemeral key pair* or *device key pair*. In the simplest case, both have the same lifetime. The account key pair allows us to create or update our device key pair. If we update our device key pair, the public key will be pushed to the VPN provider's proxy servers, which will give us access.

6.1 Onion Services

This section describes how onion services could be integrated into the OnionVPN ecosystem. We have already shown the feasibility of onion services inside OnionVPN in Section 4.3. We will present different alternatives that have different benefits and weaknesses. The biggest difference to Tor is that the relays themselves do not have supporting software, and access to them is expensive. VPN providers could support OnionVPN, but that would require an agreement on a standard interface.

This section is organized as follows: In the first section, we describe how a client can find an onion service. The following three subsections use the technical connection methods and describe the whole process with introduction and rendezvous points. See Figure 7.

6.1.1 URL and Directory Service. Without a directory service, the relevant connection data must be encoded into the URL. This would at least include the IP address and port of the introduction point and the service's public identity key. This scheme is only viable for short-lived services with a limited user base. For a broader user base, a Tor-like directory service is necessary, where only the public identity key is encoded into the URL. Onion addresses or URLs for services in Tor have a public key, a checksum and a version field in a base32-encoded string which ends with `.onion`.

6.1.2 Simplest Implementation. A straightforward implementation uses port forwarding as a connection method. This is not much different if we use a normal VPN, but it would have only one relay. In our case, there could be multiple hops. This relay plays the role of introduction and rendezvous point. See number one inside Figure 7a. The first step the service needs to do is to tell the client where it is listening for connections; this can be done by encoding all necessary parameters into a URL. In the second step, the client must connect to the services using these parameters. If port forwarding is not available, it is also possible to use IPP as a connection method. See number two inside Figure 7b.

The disadvantage is that there is only one known location; it allows an adversary to easily monitor traffic, censor the service or launch a Denial-Of-Service attack. But it has advantages over a normal VPN. It uses multiple hops, which hides the location of the service, and every NAT on the way does not matter.

6.1.3 Separate Introduction Point and Rendezvous Point. In this section, we will use IPP as a connection method (see Section 4.3.2) since port forwarding is no longer widely supported. Separating introduction and Rendezvous Point (RP), see Figure 7c, makes it harder to monitor client traffic. If they built a new circuit or reused an old circuit, it makes it easier or harder for an adversary. Tor does not reuse circuits for e. g., RP, but in our case, it makes sense to think about it because building a circuit costs money, because of path selection algorithm with three hops or three distinct provider. To access each provider you need to pay a fee. The cost is limited to the number of suitable providers.

If we would build a new circuit for RP, this circuit is only used for a short amount of time. Reusing a circuit means that only one circuit or only a few circuits are used for an extended amount of time. If both built a new circuit, this is comparable to Tor. If the client reuses a circuit, then the same service could track a client, and if many services collude, they could assign the communication to one client. If only the service reuses a circuit, an adversary could easily monitor the traffic of many clients. If both reuse circuits, an adversary can also easily separate clients from each other and track them over a long period of time.

Other benefits of this separation are: It provides location hiding of the service due to multiple hops. During the introduction and rendezvous phase, all communication is end-to-end encrypted. In the rendezvous phase, the service and client communicate over a WireGuard tunnel, so the service's underlying protocol does not matter and is not directly visible to an outside observer. The client can be sure it communicates with the correct service in any phase, and during the introduction phase, the client gives the service its public identity key. Afterwards, all communication is end-to-end, authenticated, and peer-to-peer. The IPP helps overcome any NAT on the way, as long as it is not symmetric.

6.1.4 Key Blinding and Introduction Points. Key blinding means that every Introduction Point has is own blinded pair and its locations change over time without relying on a distributed hash table on the relays themselves. We will use Tor's Key Blinding scheme and the algorithm to find the responsible Onion Service Directory in the hash ring [43]. As already mentioned, we do not store any data on the relays; we use this algorithm instead to find the location of our Introduction points.

This algorithm requires the following input arguments. Three global scalar values; a shared random value (*SRV*); the time period (*TP*); and how many introduction points (*IPNUM*). The *SRV* is just a nonce that changes with each *TP*. *TP* is just an integer that specifies in which period we are, it does not say anything about how long a period is, in the case of Tor, it changes every 24 hours. *IPNUM* specifies the minimum number of Introduction Points a service listens on. Additionally, we need a list of introduction points, each with an identity key pair. All of this data can be stored in a Directory Service.

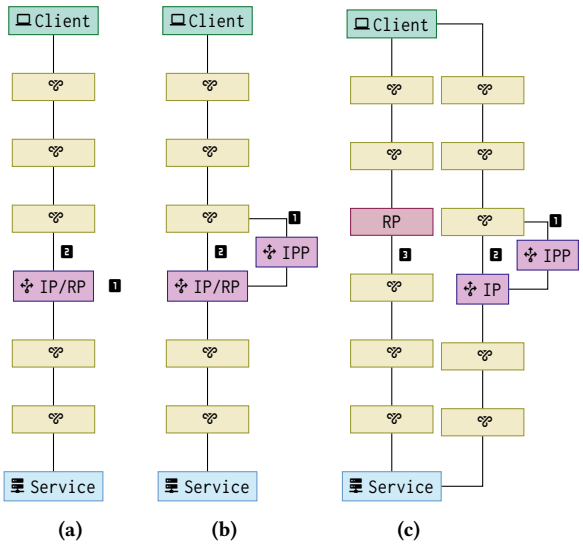


Figure 7: Three examples of how client and service can connect. The first image is with port forwarding, where the introduction and rendezvous points are not separated. The second image is the same as the first but with the IPP. The last example separates the introduction and rendezvous point, and uses the IPP.

Each service has its identity key pair and up to *IPNUM* blinded key pairs derived from its initial pair. It needs two sets of blinded key pairs, one for the current *TP* and one for the last *TP*, so clients that do not have a current consensus can still connect. The service will take the list of Introduction Points and calculate for each a hash that includes its public identity key, *SRV* and *TP*. It sorts this list by the hash value to get an ordered list of Introduction Points. For each of its blinded key pairs, it calculates a hash over the blinded public key, *IPNUM* and *TP*, and finds the closest match in the ordered list of Introduction Points. Each match is an Introduction Point where it will listen for clients.

A client has only the public identity key of a service. It gets this key through the URL (see Section 6.1.1) but can easily calculate the blinded key. This key can do the same calculation as the service and can find all Introduction Points. The downside of this approach is that Introduction Points have only a derived key pair rather than a randomly generated one. It could be expensive for service operators, especially for a Tor-like setup. Finding a balance between cost and anonymity is crucial.

6.2 Key Selling Onion Service

If VPN providers would not widely support the key selling scheme, which was dissected at the beginning (see Section 6), it is possible to write an onion service that is reachable over Tor. Usually, providers give users an account that is usable with multiple devices. Each device has its key pair. If a user wants access to a VPN provider, they can generate a new key pair, go to that service and request to insert their public key as one device. The service needs to check if an account for that provider is available and has a free device slot. If it has not a free device slot, it needs to buy a new account. After that, it can request a small fee from the user and allow the usage for a fixed time frame. The public key will be inserted if the user pays it using an anonymous payment method. After that, a user has access.

7 DISCUSSION AND LIMITATIONS

Our intention is not to develop an alternative to Tor. However, bulk downloads are a problem for the Tor network and often do not need Tor's privacy and anonymity features. Thus, OnionVPN provides an alternative for users who use the Tor network to download large files or use video streaming. This relieves the Tor network and provides a better experience for other users who need privacy and anonymity features. Next, we will discuss some limitations.

Choosing a provider uniform at random, as OnionVPN does in the first step of the algorithm, creates a load balancing problem. Smaller providers will get disproportionate amounts of traffic to the number of servers or bandwidth they have. On the other hand, an increase in traffic will result in an increase in revenue, which will in turn allow them to extend their capacity.

We only consider three providers as suitable: Mullvad, OVPN and iVPN. Due to the low AS diversity it is possible that AS's and Internet exchange points (IXes) are heavily connected to each other and there might not be an ideal network path without intersections. A stochastic path selection algorithm that minimizes the risk of compromise as suggest by Nithyanand *et al.* [53], could help here. Unfortunately this kind of path selection algorithm is not attractive for VPN provider's business models, because it prefers smaller providers and might help them grow. However, OnionVPN-support might help a provider to acquire new users or increase reputation.

Because access to the relays is expensive, it makes it difficult to find a good balance in favor of anonymity.

8 RELATED WORK

8.1 Virtual Private Networks

Some VPN providers support a two-hop solution, which they offer under various names such as "Double VPN" [54], "Secure Core" [55], or "Multihop" [56]. In the case of Mullvad, the packet gets encrypted by WireGuard for the exit hop and sent to the entry hop, which forwards the packet. However, not in an onion-like fashion where it would be encrypted twice for each hop. It does not work across providers. The description of NordVPN is unclear, but it looks similar to what Mullvad does.

VPN providers offer articles on how to use "Onion over VPN" or "VPN over Tor" [57]–[59]. Some providers also directly support these as features in their applications. A VPN server is the first or last hop with a Tor circuit. There is no onion routing itself with the VPN. One or two-hop VPN solutions will always be faster than a three-hop solution.

Recently, Nym published NymVPN [60] for beta testing, which has a fast and anonymous mode. The fast mode is comparable to the existing two-hop solutions but uses onion routing. The anonymous mode uses two gateway nodes and three mix network hops

with a novel onion encryption. Since NymVPN was released during the writing of this paper, we could not compare the performance, and we will leave this for future work.

8.2 Anonymity Networks

Anonymity networks can have lower, low, or high latency. In the first category, are LAP [61], Dovetail [62], and HORNET [63]. These systems have high throughput, are implemented at the network level rather than as an overlay network, are primarily stateless, and offer limited anonymity guarantees. The second category contains systems like I2P [9] and Tor [8], which are implemented as overlays, have lower throughput, are not stateless, can defend against an adversary with a limited view and control of the network, and are practical and usable. The last category is mixed networks, to which, for example, Nym [64] belongs. These have very low throughput but can defend against a global adversary.

Our work is closely related to Onion Routing [9] and belongs to the low latency category. Currently, two practical, usable networks implement this concept with different goals.

Tor [8] based on Onion Routing [9] is an overlay network that is primarily designed for anonymous low-latency communication, like web-browsing. It has around 8000 relays that are operated by volunteers. A circuit through the network goes over three hops. Each circuit is bidirectional and each message has multiple layers of encryption, like an onion.

I2P [9] is a closed peer-to-peer overlay network that uses the Internet, but usually, no communication to endpoints outside the network occurs. Volunteers provide relays. I2P's support for garlic routing, a variant of onion routing, is characterized by unidirectional tunnels. This means that two tunnels exist from the sender to the receiver, a unique approach that sets I2P apart. Messages have multiple layers of encryption, like an onion, but are bundled with other messages, like a garlic bulb. It can defend against the same adversary as Tor but makes traffic correlation attacks more difficult.

8.3 Path Selection

A user can be de-anonymized if an adversary can observe both ends of a Tor circuit. IXes and ASs are in a position to do that, but Tor's path selection algorithm is not AS-aware. This problem is not exclusive to Tor; all low-latency anonymity networks have this problem.

A user can be de-anonymized if an adversary can observe both ends in a low-latency anonymity network. Certain network-level attackers on the level of AS or IXes can do that. Tor's path selection algorithm is not aware of these adversaries. Previous work used path selection algorithms as countermeasures for Tor that are AS-aware [39], [65]–[68] or measure the extend of the problem [36], [38], [53], [69]. Nithyanand *et al.* [53] shows that without counter measures 37 % to 85 % of paths could be compromised.

Edman and Syverson [39] evaluated their path selection algorithm by choosing relays by *unique ASN*, *unique Country*, or *unique ASN path*. They found out that the first two are similarly effective. The latter builds upon complete path inference between the client and the entry node and between the exit node and the destination. The network path between both can cross different ASs and at least both paths must have non-intersecting sets of ASs. This is complicated by routing asymmetry, where packets travel differently forwards and backwards. A complete AS path inference algorithm is computationally complex, so they only proposed a heuristic. A tunable path selection algorithm was introduced by Akhoondi *et al.* [65] which "*reduces the median latencies by 25 % while also decreasing the false negative rate of not detecting a potential snooping AS from 57 % to 11 %*".

9 CONCLUSION AND FUTURE WORK

In this paper, we presented OnionVPN, which brings onion routing to standard VPN protocols and protects against untrusted VPN providers. We analysed 118 VPN providers and found that only three would be suited and six partially suited for using OnionVPN. Using network namespaces, we avoid data leaking [7], and with our AS-aware path selection algorithm, we protect against AS-level attackers. We have shown that onion services are possible with OnionVPN and developed a cryptographic NAT traversal algorithm to bypass NAT. Our performance measurements show that OnionVPN is slower than Tor for smaller file transfers, but for bigger file transfers, OnionVPN outperforms Tor. Thus, OnionVPN provides an alternative for users who use the Tor network to download large files but do not need the same level of privacy and anonymity Tor provides. That could also relieve the Tor network and provide a better experience for Tor users who need privacy and anonymity from Tor. Additionally, we have presented an OnionVPN ecosystem in which Onion services could implemented on a large scale. If VPN providers do not support OnionVPN, we propose self-hosted solutions, such as a key selling proxy, as an alternative.

The following points we declare as future work: (1) Conduct a more comprehensive analysis of the path selection algorithm with more VPN providers, an ASN inference algorithm and added bandwidth or load weights. Due to the low AS diversity, it is still unclear how many circuits can exist without an AS observing both ends or how high the compromise rate is. (2) Create a simulation of an OnionVPN network that is informed by real distributions to better understand performance characteristics. (3) Find a balance between cost and anonymity, e. g., Onion Services with separate introduction and rendezvous points, where the service builds always a new circuit to the rendezvous point, give better anonymity but also cost more. (4) Gain a better understanding of the implications of the proposed alternative services in the ecosystem. These services introduce complexity which could be exploited to easier de-anonymize users.

ACKNOWLEDGMENTS

The European Union partly funded this work under grant number ROF-SG20-3066-3-2-2. We would like to thank Dominik Thalhammer for the idea of connecting multiple namespaces and performing preliminary performance measurements, Daniel Urban for the preliminary performance measurements for OnionVPN vs. Tor, and Daniel Forster, Katharina Schiller, Christopher Pahl, and Elvin Eardley DeSouza for reading the early drafts and providing feedback to us.

REFERENCES

[1] J. A. Donenfeld, "WireGuard: Next generation kernel network tunnel," in *Proceedings 2017 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2017, ISBN: 978-1-891562-46-4. DOI: 10.14722/ndss.2017.23160. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/ (visited on 07/08/2024).

[2] "Report: No-Log VPNs Reveal Users' Personal Data and Logs," vpnMentor. (Jul. 10, 2020), [Online]. Available: https://www.vpnmentor.com/blog/report-free-vpns-leak/ (visited on 01/16/2021).

[3] J. Youngren. "Hidden VPN Owners Unveiled: 101 VPNs Run by 23 Companies," VPNpro. (Jan. 5, 2021), [Online]. Available: https://vpnpro.com/blog/hidden-vpn-owners-unveiled-97-vpns-23-companies/ (visited on 07/08/2024).

[4] D. Ruiz. "21 million free VPN users' data exposed." (Mar. 3, 2021), [Online]. Available: https://blog.malwarebytes.com/cybercrime/privacy/2021/03/21-million-free-vpn-users-data-exposed/ (visited on 07/08/2024).

[5] C. Durward. "LimeVPN Website Taken Down by Hacker? Customers Sensitive Information Hacked and Sold on Telegram." (Jun. 29, 2021), [Online]. Available: https://www.privacysharks.com/well-known-vpn-provider-in-security-breach-sensitive-information-hacked-and-sold-on-dark-web/ (visited on 07/08/2024).

[6] R. Ramesh, L. Evdokimov, D. Xue, and R. Ensafi, "VPNalyzer: Systematic Investigation of the VPN Ecosystem," in *Proceedings 2022 Network and Distributed System Security Symposium*, San Diego, CA, USA: Internet Society, 2022, ISBN: 978-1-891562-74-7. DOI: 10.14722/ndss.2022.24285.

[7] N. Xue, Y. Malla, Z. Xia, C. Pöpper, and M. Vanhoef, "Bypassing Tunnels: Leaking VPN Client Traffic by Abusing Routing Tables," presented at the 32nd USENIX Security Symposium, 2023, pp. 5719–5736, ISBN: 978-1-939133-37-3.

[8] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router:" Tech. Rep., 2004. [Online]. Available: https://svn-archive.torproject.org/svn/projects/design-paper/tor-design.pdf (visited on 07/08/2024).

[9] D. Goldschlag, M. Reed, and P. Syverson, "Onion Routing for Anonymous and Private Internet Connections," *Communications of the ACM*, vol. 42, pp. 39–41, 1999.

[10] R. Jansen, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "Never Been KIST: Tor's Congestion Management Blossoms with Kernel-Informed Socket Transport," in *Proceedings of the 23rd USENIX Security Symposium*, 2014, p. 16.

[11] R. Jansen, M. Traudt, J. Geddes, C. Wacek, M. Sherr, and P. Syverson, "KIST: Kernel-Informed Socket Transport for Tor," *ACM Transactions on Privacy and Security (TOPS)*, no. 1, pp. 1–37, 2018. DOI: 10.1145/3278121.

[12] M. AlSabah *et al.*, "DefenestraTor: Throwing Out Windows in Tor," in *Proceedings of the 11th on Privacy Enhancing Technologies Symposium (PETS)*, ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 134–154. DOI: 10.1007/978-3-642-22263-4_8.

[13] T. Wang, K. Bauer, C. Forero, and I. Goldberg, "Congestion-Aware Path Selection for Tor," in *Proceedings of the 16th International Conference on Financial Cryptography and Data Security (FC)*, A. D. Keromytis, Ed., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2012, pp. 98–113, ISBN: 978-3-642-32946-3. DOI: 10.1007/978-3-642-32946-3_9.

[14] M. Perry. "Congestion Control Arrives in Tor 0.4.7-Stable! | Tor Project." (May 4, 2022), [Online]. Available: https://blog.torproject.org/congestion-contrl-047/ (visited on 07/08/2024).

[15] S. Pahl, F. Adamsky, D. Kaiser, and T. Engel, "Examining the Hydra: Simultaneously Shared Links in Tor and the Effects on Its Performance," in *Proceedings on Privacy Enhancing Technologies*, vol. 2023, 2023, pp. 268–285. DOI: 10.56553/popets-2023-0081.

[16] C. Tang and I. Goldberg, "An Improved Algorithm for Tor Circuit Scheduling," presented at the 17th ACM Conference on Computer and Communications Security (CCS), Oct. 4, 2010, pp. 329–339. DOI: 10.1145/1866307.1866345.

[17] Perrin, Trevor, "The Noise Protocol Framework," Tech. Rep., 2018, Revision 34. [Online]. Available: https://noiseprotocol.org/noise.pdf (visited on 11/18/2020).

[18] D. Stebila. "List of Attacks from recent years on TLS." (2020), [Online]. Available: http://files.douglas.stebila.ca.s3.amazonaws.com/files/research/presentations/tls-attacks/tls_attacks_table.pdf (visited on 12/11/2020).

[19] N. Ferguson and B. Schneier, "A Cryptographic Evaluation of IPsec," p. 28, 1999.

[20] Donenfeld, Jason A. and Milner, Kevin, "Formal Verification of the WireGuard Protocol," Tech. Rep., 2018, Revision b956944. [Online]. Available: https://www.wireguard.com/papers/wireguard-formal-verification.pdf (visited on 11/18/2020).

[21] B. Dowling and K. G. Paterson, "A Cryptographic Analysis of the WireGuard Protocol," in *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS 2018)*, vol. 10892, Springer International Publishing, 2018, pp. 3–21. DOI: 10.1007/978-3-319-93387-0_1.

[22] B. Lipp, "A Mechanised Computational Analysis of the WireGuard Virtual Private Network Protocol," Masterthesis, Karlsruhe Institute of Technology, May 23, 2018. [Online]. Available: https://bit.ly/3mU8jNZ.

[23] D. J. Bernstein, "ChaCha, a variant of Salsa20," in *Workshop Record of SASC*, 2008. [Online]. Available: https://cr.yp.to/chacha/chacha-20080128.pdf.

[24] D. J. Bernstein, "The Poly1305-AES Message-Authentication Code," in *Fast Software Encryption*, H. Gilbert and H. Handschuh, Eds., Berlin, Heidelberg: Springer, 2005, pp. 32–49, ISBN: 978-3-540-31669-5. DOI: 10.1007/11502760_3.

[25] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "BLAKE2: Simpler, Smaller, Fast as MD5," in *Applied Cryptography and Network Security*, vol. 7954, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 119–135, ISBN: 978-3-642-38979-5. DOI: 10.1007/978-3-642-38980-1_8.

[26] J.-P. Aumasson and D. J. Bernstein, "SipHash: A Fast Short-Input PRF," in *Progress in Cryptology - INDOCRYPT 2012*, Berlin, Heidelberg: Springer, 2012, pp. 489–508, ISBN: 978-3-642-34931-7. DOI: 10.1007/978-3-642-34931-7_28.

[27] A. Hülsing, K. Ning, P. Schwabe, F. Weber, and P. R. Zimmermann, "Post-Quantum WireGuard," in *2021 2021 IEEE Symposium on Security and Privacy (SP)*, IEEE Computer Society, May 2021, pp. 511–528. DOI: 10.1109/SP40001.2021.00030.

[28] L. Overlier and P. Syverson, "Locating hidden servers," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*, Berkeley/Oakland, CA: IEEE, 2006, 15 pp.–114, ISBN: 978-0-7695-2574-7. DOI: 10.1109/SP.2006.24. [Online]. Available: http://ieeexplore.ieee.org/document/1624004/ (visited on 07/08/2024).

[29] "Tor rendezvous specification - version 3." (Feb. 5, 2024), [Online]. Available: https://raw.githubusercontent.com/torproject/torspec/main/rend-spec-v3.txt (visited on 07/08/2024).

[30] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit fingerprinting attacks: Passive deanonymization of tor hidden services," presented at the 24th USENIX Security Symposium, 2015, pp. 287–302, ISBN: 978-1-939133-11-3.

[31] G. Kadianakis, T. Polyzos, M. Perry, and K. Chatzikokolakis, *Tor circuit fingerprinting defenses using adaptive padding*, Jan. 11, 2022. DOI: 10.48550/arXiv.2103.03831. arXiv: 2103.03831[cs].

[32] T. Bui, S. P. Rao, M. Antikainen, and T. Aura, *Client-side vulnerabilities in commercial VPNs*, Dec. 10, 2019. arXiv: 1912.04669[cs]. [Online]. Available: http://arxiv.org/abs/1912.04669 (visited on 07/08/2024).

[33] "Routing & Network Namespace Integration." (Feb. 5, 2024), [Online]. Available: https://www.wireguard.com/netns/ (visited on 07/08/2024).

[34] "Strongswan in linux network namespaces." (Feb. 8, 2024), [Online]. Available: https://docs.strongswan.org/docs/5.9/howtos/nameSpaces.html (visited on 07/08/2024).

[35] "Project todo - wireguard." (Nov. 13, 2023), [Online]. Available: https://www.wireguard.com/todo/ (visited on 07/08/2024).

[36] S. J. Murdoch and P. Zieliński, "Sampled traffic analysis by internet-exchange-level adversaries," in *Privacy Enhancing Technologies*, N. Borisov and P. Golle, Eds., vol. 4776, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 167–183, ISBN: 978-3-540-75550-0. DOI: 10.1007/978-3-540-75551-7_11. [Online]. Available: http://link.springer.com/10.1007/978-3-540-75551-7_11 (visited on 07/08/2024).

[37] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: Traffic correlation on tor by realistic adversaries," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, Berlin, Germany: ACM Press, 2013, pp. 337–348, ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516651. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2508859.2516651 (visited on 10/16/2023).

[38] J. Juen, A. Johnson, A. Das, N. Borisov, and M. Caesar, "Defending tor from network adversaries: A case study of network path prediction," 2, vol. 2015, Jun. 1, 2015, pp. 171–187. DOI: 10.1515/popets-2015-0021.

[39] M. Edman and P. Syverson, "As-awareness in tor path selection," in *Proceedings of the 16th ACM conference on Computer and communications security - CCS '09*, Chicago, Illinois, USA: ACM Press, 2009, p. 380, ISBN: 978-1-60558-894-0. DOI: 10.1145/1653662.1653708. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1653662.1653708 (visited on 11/02/2022).

[40] Z. Weinberg, S. Cho, N. Christin, V. Sekar, and P. Gill, "How to catch when proxies lie: Verifying the physical locations of network proxies with active geolocation," Oct. 31, 2018, ISBN: 978-1-4503-5619-0. DOI: 10.1145/3278532.3278551.

[41] "Mullvad – servers." (Apr. 11, 2024), [Online]. Available: https://mullvad.net/en/servers (visited on 07/08/2024).

[42] J. Qiu and L. Gao, "CAM04-4: AS path inference by exploiting known AS paths," in *IEEE Globecom 2006*, ISSN: 1930-529X, Nov. 2006, pp. 1–5. DOI: 10.1109/GLOCOM.2006.27. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4150657 (visited on 08/12/2024).

[43] "Tor Rendezvous Specification – Version 3," The Tor Project, Tech. Rep., Jul. 8, 2024. [Online]. Available: https://github.com/torproject/torspec/blob/main/rend-spec-v3.txt (visited on 07/08/2024).

[44] "Removing the support for forwarded ports - Blog," Mullvad VPN. (May 29, 2023), [Online]. Available: https://mullvad.net/en/blog/2023/5/29/removing-the-support-for-forwarded-ports/ (visited on 07/08/2024).

[45] "Gradual removal of port forwarding from the IVPN service." (Jun. 29, 2023), [Online]. Available: https://www.ivpn.net/blog/gradual-removal-of-port-forwarding/ (visited on 07/08/2024).

[46] "Ip leak affecting vpn providers with port forwarding." (Feb. 8, 2024), [Online]. Available: https://www.perfect-privacy.com/en/blog/ip-leak-vulnerability-affecting-vpn-providers-with-port-forwarding (visited on 07/08/2024).

[47] C. H. E. A. Keranen and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal," IETF, Request for Comments RFC 8445, Jul. 2018, 100 pp. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8445.

[48] H. Rosenberg Weinberger and Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," IETF, Request for Comments RFC 3485, Mar. 2003, 47 pp. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3485.

[49] Tor Project. "Tor Metrics!" (2024), [Online]. Available: https://metrics.torproject.org/ (visited on 05/16/2024).

[50] Rob Jansen and Justin Tracey and Ian Goldberg. "tornettools." (Feb. 22, 2022), [Online]. Available: https://github.com/shadow/tornettools/blob/2ea14da/tornettools/generate_tgen.py (visited on 07/08/2024).

[51] verizon. "Monthly IP Latency Data | Verizon Enterprise Solutions." (2024), [Online]. Available: https://enterprise.verizon.com/terms/latency (visited on 06/15/2024).

[52] M. Perry, "RTT-based Congestion Control for Tor," The Tor Project, Jul. 2, 2020. [Online]. Available: https://gitlab.torproject.org/tpo/core/torspec/-/blob/main/proposals/324-rtt-congestion-control.txt (visited on 07/08/2024).

[53] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira, "Measuring and Mitigating AS-level Adversaries Against Tor," presented at the 23rd Network and Distributed System Security Symposium (NDSS), 2016. DOI: 10.14722/ndss.2016.23322.

[54] "Extra security with double vpn | nordvpn." (Aug. 29, 2024), [Online]. Available: https://nordvpn.com/features/double-vpn (visited on 08/29/2024).

[55] "What is secure core?" (Aug. 29, 2024), [Online]. Available: https://protonvpn.com/support/secure-core-vpn (visited on 08/29/2024).

[56] "Multihop with wireguard." (Apr. 17, 2024), [Online]. Available: https://mullvad.net/en/help/multihop-wireguard (visited on 08/29/2024).

[57] "Onion over vpn: Layers of ultimate online security | nordvpn." (Aug. 29, 2024), [Online]. Available: https://nordvpn.com/features/onion-over-vpn (visited on 08/29/2024).

[58] "Onion over a VPN." (Nov. 9, 2023), [Online]. Available: https://surfshark.com/blog/tor-over-vpn (visited on 08/29/2024).

[59] "What is Onion Over VPN." (Jan. 20, 2024), [Online]. Available: https://www.cyberghostvpn.com/privacyhub/onion-over-vpn (visited on 08/29/2024).

[60] "Nymvpn free beta testing has begun." (Aug. 21, 2024), [Online]. Available: https://nymvpn.com/en/blog/nymvpn-free-beta-testing (visited on 08/29/2024).

[61] H.-C. Hsiao *et al.*, "LAP: Lightweight anonymity and privacy," in *2012 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA: IEEE, May 2012, pp. 506–520, ISBN: 978-1-4673-1244-8. DOI: 10.1109/SP.2012.37. [Online]. Available: http://ieeexplore.ieee.org/document/6234433/ (visited on 08/28/2024).

[62] J. Sankey and M. Wright, "Dovetail: Stronger anonymity in next-generation internet routing," in *Privacy Enhancing Technologies*, E. De Cristofaro and S. J. Murdoch, Eds., Series Title: Lecture Notes in Computer Science, vol. 8555, Cham: Springer International Publishing, 2014, pp. 283–303, ISBN: 978-3-319-08505-0. DOI: 10.1007/978-3-319-08506-7_15. [Online]. Available: http://link.springer.com/10.1007/978-3-319-08506-7_15 (visited on 08/30/2024).

[63] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "HORNET: High-speed onion routing at the network layer," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, New York, NY, USA: Association for Computing Machinery, Oct. 12, 2015, pp. 1441–1454, ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813628. [Online]. Available: https://dl.acm.org/doi/10.1145/2810103.2813628 (visited on 08/30/2024).

[64] C. Diaz, H. Halpin, and A. Kiayias, "The nym network: The next generation of privacy infrastructure," *White Paper, version*, vol. 1, 2021.

[65] M. Akhoondi, C. Yu, and H. V. Madhyastha, "LASTor: A low-latency AS-aware tor client," in *2012 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA: IEEE, May 2012, pp. 476–490. DOI: 10.1109/SP.2012.35. (visited on 07/08/2024).

[66] A. Barton and M. Wright, "DeNASA: Destination-naive AS-awareness in anonymous communications," 2016. [Online]. Available: https://petsymposium.org/popets/2016/popets-2016-0044.php (visited on 04/17/2024).

[67] F. Rochet, R. Wails, A. Johnson, P. Mittal, and O. Pereira, "CLAPS: Client-Location-Aware Path Selection in Tor," presented at the 27th ACM Conference on Computer and Communications Security (CCS), Nov. 2, 2020, pp. 17–34. DOI: 10.1145/3372297.3417279.

[68] Y. Sun, A. Edmundson, N. Feamster, M. Chiang, and P. Mittal, "Counter-RAPTOR: Safeguarding tor against active routing attacks," presented at the 38th IEEE Symposium on Security and Privacy (SP), May 2017, pp. 977–992. DOI: 10.1109/SP.2017.34.

[69] Y. Sun *et al.*, "RAPTOR: Routing Attacks on Privacy in Tor," presented at the 24th USENIX Security Symposium, 2015, pp. 271–286, ISBN: 978-1-939133-11-3.

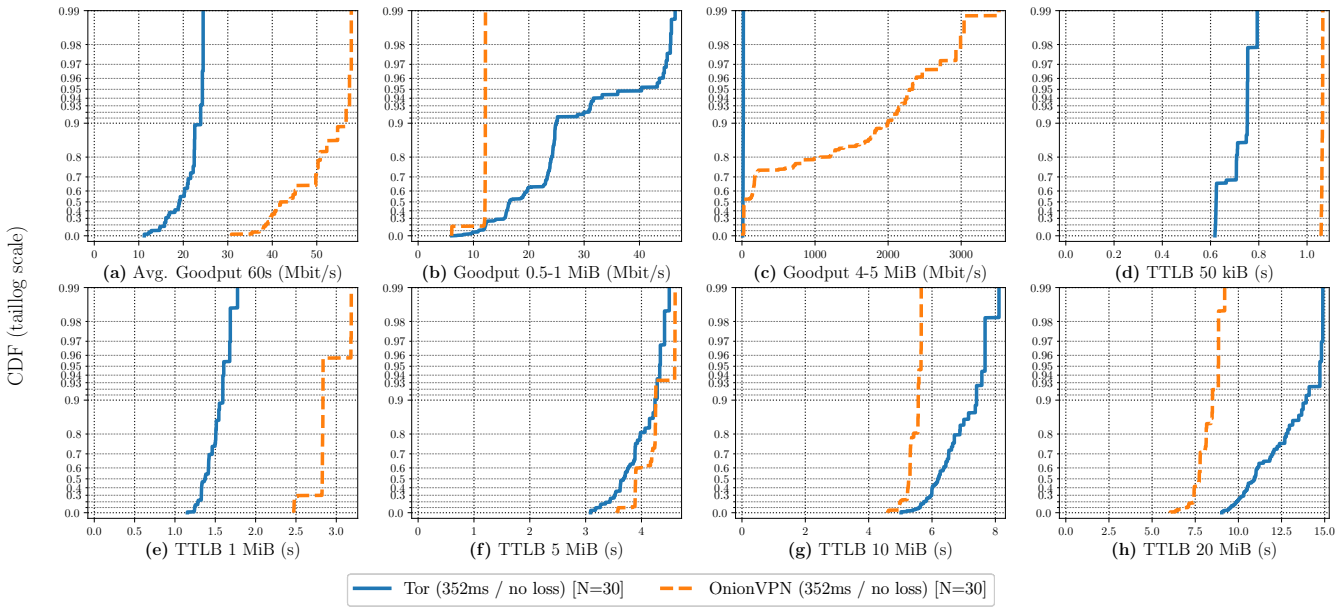


Figure 8: Performance measurements compare Tor against OnionVPN in a local setup (see Section 5.2.2 for details about the experimental setup). This setup has a circuit latency of 352 ms but no packet loss for all experiments.

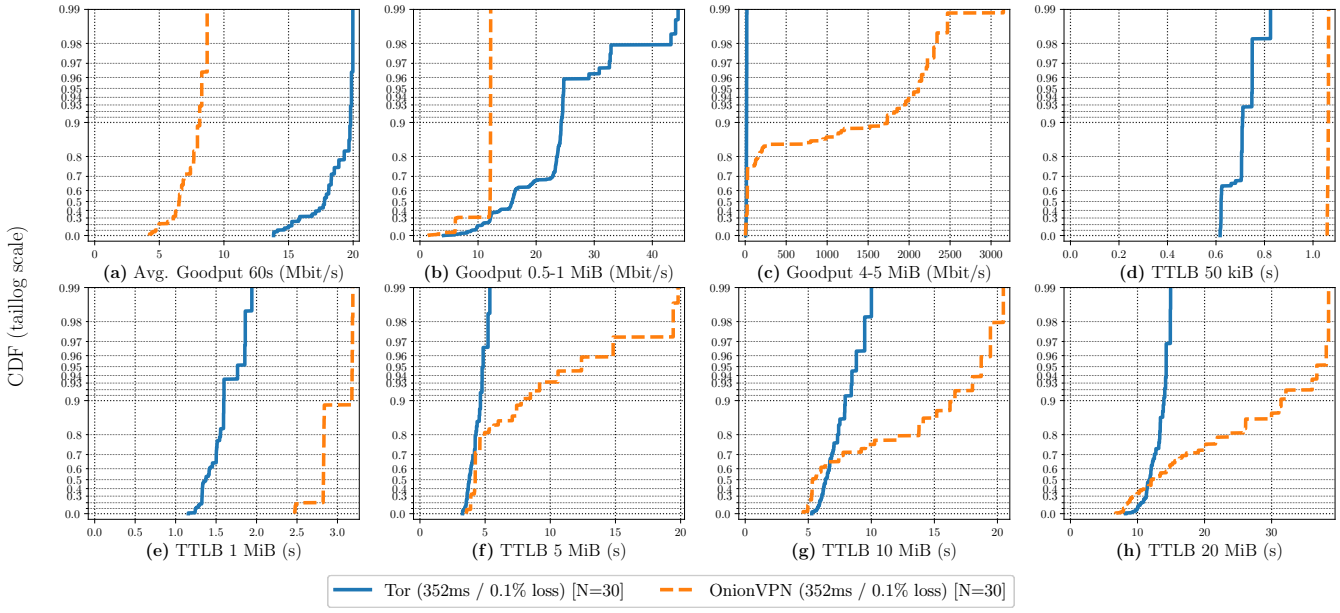


Figure 9: Performance measurements compare Tor against OnionVPN in a local setup (see Section 5.2.2 for details about the experimental setup). This setup has a circuit latency of 352 ms and packet loss of 0.1 % for all experiments.

A FULL PROVIDER LIST

You can find a list of all VPN providers in Table 2.

Table 2: List of all VPN providers

Provider	Suitable	Provider	Suitable	Provider	Suitable
Mullvad	●	Bitdefender Premium VPN	○	Total VPN	✗
OVPN	●	Surfshark	○	TorVPN	✗
iVPN	●	Hotspot Shield	○	tigerVPN	✗
Windscribe	◐	Liberty Shield	○	SwissVPN	✗
AirVPN	◐	Panda VPN	○	SpyOff	✗
VPN.ac	◐	ExpressVPN	○	Speedify Review	✗
ProtonVPN	◐	IPVanish	○	Banana VPN	✗
Anonine	◐	Avira Phantom VPN	○	Betternet	✗
Hide.me	◐	HMA VPN	○	ZPN	✗
Easy Hide IP	○	VPN Unlimited	○	SiteLock VPN	✗
GhostPath	○	SurfEasy	○	TorGuard	✗
ZenVPN	○	Adtelly	○	EarthVPN	✗
CactusVPN	○	NordLayer	✗	UltraVPN	✗
Shellfire VPN	○	Norton Secure VPN	✗	Opera VPN	✗
PrivadoVPN	○	BartVPN	✗	Mozilla	✗
SwitchVPN	○	OverPlay	✗		
NoodleVPN	○	Perimeter 81	✗		
WorldVPN	○	PrimeVPN	✗		
FrootVPN	○	Private WiFi	✗		
VPNTunnel	○	proXPN	✗		
Hide My IP	○	MyVPN	✗		
Le VPN	○	Kepard	✗		
ZoogVPN	○	LiquidVPN	✗		
boxpn	○	Leafy VPN	✗		
VPN.ht	○	ProxyServer.com	✗		
VyprVPN	○	IronSocket	✗		
VPNArea	○	Internetz.me	✗		
SlickVPN	○	ibVPN	✗		
VPNSecure	○	NJALLA VPN	✗		
VPN.asia	○	HashtagVPN	✗		
FastestVPN	○	Encrypt.me	✗		
AVG Secure VPN	○	BullGuard VPN	✗		
Perfect Privacy	○	Buffered VPN	✗		
PrivateVPN	○	BolehVPN	✗		
Avast SecureLine VPN	○	Proxy.sh	✗		
StrongVPN	○	SecureTunnel	✗		
HideIPVPN	○	RUSVPN	✗		
Atlas VPN	○	VeePN	✗		
Anonymous VPN	○	ZenMate	✗		
Bright VPN	○	WeVPN	✗		
BTGuard	○	Webroot WiFi Security	✗		
12VPN	○	VPNLand	✗		
Private Internet Access	○	VPNhub	✗		
ClearVPN	○	VPNGhost	✗		
CyberGhost VPN	○	VPN4ALL	✗		
F-Secure Freedome	○	VPN Traffic	✗		
GOOSE VPN	○	VersaVPN	✗		
GoTrusted	○	USAIP.eu	✗		
PureVPN	○	SaferVPN	✗		
Ivacy	○	TunnelBear	✗		
NordVPN	○	Trust.Zone	✗		