# 1 Introduction

# 2 Related work

In this chapter, we will examine prior works relevant to our problem of connecting multiple parties over The Internet for a joint MPyC computation. To identify the challenges, we will begin with an overview of The Internet Protocol Suite, which is central to the modern Internet. Following that, we will review several existing network overlay solutions that we can later use as building blocks for an overall solution to our problem. To maintain a consistent mental picture of the solutions, we will map them to the layers of the model. While many protocols implement aspects of several layers and do not strictly fit inside the OSI model, it is still a useful visualization tool.

## 2.1 The Internet

The modern Internet is a global multi-tiered network of devices that communicate using the protocols of the Internet Protocol Suite (TCP/IP). Typically, an manages the physical infrastructure that connects an end-user’s devices with the rest of the internet. Figure describes the responsibilities of the 7 layers of the OSI model and how they fit with the protocols of the Internet Protocol Suite, which are often referred to as the TCP/IP model. The newer TCP/IP model only recognizes 4 layers as it merges the OSI Session (L5) and Presentation (L6) layers into the Application layer (L7), as well as the Physical layer (L1) into the Data link layer (L2). While the TCP/IP model is a more accurate representation of The Internet, the 7-layer numbers of the OSI model are still widely used in the literature.

![OSI model mapping of the Internet Protocol Suite](data:application/pdf;base64,)

OSI model mapping of the Internet Protocol Suite

The is a Network (L3) protocol of the Internet Protocol Suite that is responsible for delivering datagrams between devices across the boundaries of their by possibly routing traffic via multiple intermediate devices (routers). A datagram is a connectionless packet that is delivered on a best-effort basis. IP datagrams have a header that contains fields such as the **IP addresses** of its source and destination, and a payload that encapsulates the data from the protocols of the layers above. Routers are devices that are part of multiple networks and relay datagrams between them based on a routing table that maps IP address ranges () to networks.

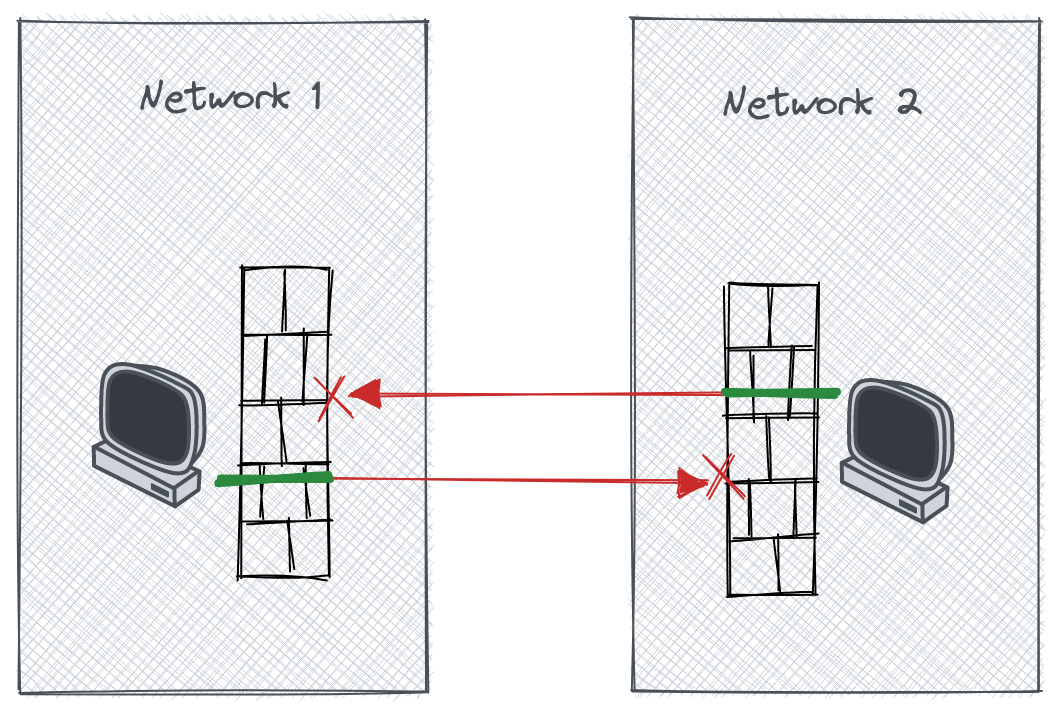
and are Transport (L4) protocols that add the concept of ports to allow having multiple communication channels simultaneously between two devices. UDP provides best-effort delivery, while TCP is a reliable transport with delivery guarantees. TCP maintains stateful connections and handles acknowledgments and retransmissions in case packets are lost in transit.

is a protocol that adds encryption on top of a reliable transport protocol such as TCP. It is usually placed in the Presentation Layer (L6), but it does not strictly fit in any single OSI layer. It is rather complex because it needs to support many possible use cases across the internet. The **Noise Protocol Framework** (Perrin 2018) is a more recent effort that applies the ideas of TLS in a simplified way by serving as a blueprint for designing use-case specific protocols for establishing secure communication channels based on handshake patterns. It powers the end-to-end encryption in messaging applications such as WhatsApp and Signal, and software such as WireGuard and Nebula.

The version of the Internet Protocol that was originally deployed globally (IPv4) uses 32-bit numbers as IP addresses, allowing for around 4 billion unique addresses. Due to the popularity of the internet, there are many more devices than available IPv4 addresses, which has caused challenges. IPv6 is a newer version of the protocol that uses a larger 128-bit address space which is sufficient for assigning 100 addresses for each atom on Earth. Its adoption has been slow, as according to Google as of 2023 only around 37% of the requests to their services use IPv6. Additionally, despite that IPv6 allows for all devices to be addressable on the Internet, for security reasons, most of them would use firewalls to block incoming remote traffic that is not associated with outgoing connections.

A widespread solution to the addressing problem is . It allows many devices without globally unique IP addresses to initiate connections to publicly addressable devices on the Internet via a limited number of gateways that must have globally unique IP addresses. A NAT gateway replaces the local source IP address of each outgoing IP datagram with its own public IP address before passing it on to the next link on the way to the destination while maintaining a mapping between the source and destination IPs in a translation table. The destination host can then address its responses back to the NAT gateway’s public IP address, which in turn replaces its own IP from the incoming datagrams with the IP of the local device and relays them to it. If the IP datagrams encapsulate TCP/UDP packets, the gateway additionally rewrites the source and destination ports, which means that NAT techniques can be placed somewhere between Layers 3 and 4 of the OSI model.

NATs have implications on connectivity that are similar to IPv6 firewalls as they allow devices on a local network to initiate bidirectional communication to remote devices with public IP addresses, but connections cannot be natively initiated by the remote devices. As Figure shows, it follows that when two devices are behind separate NATs, neither can contact the other first. **Client/Server** communications are less affected by this limitation because many local Clients can contact a Server deployed to a public IP address by its administrators. **Peer-to-Peer** communications, however, are more challenging because the peers are often devices in separate residential networks behind different NATs. Several **NAT traversal** techniques try to solve this with different performance tradeoffs and success that varies depending on the NAT (**natBehaviorRFC?**) and its behaviors when mapping ports and IP addresses.



Two parties behind separate NATs

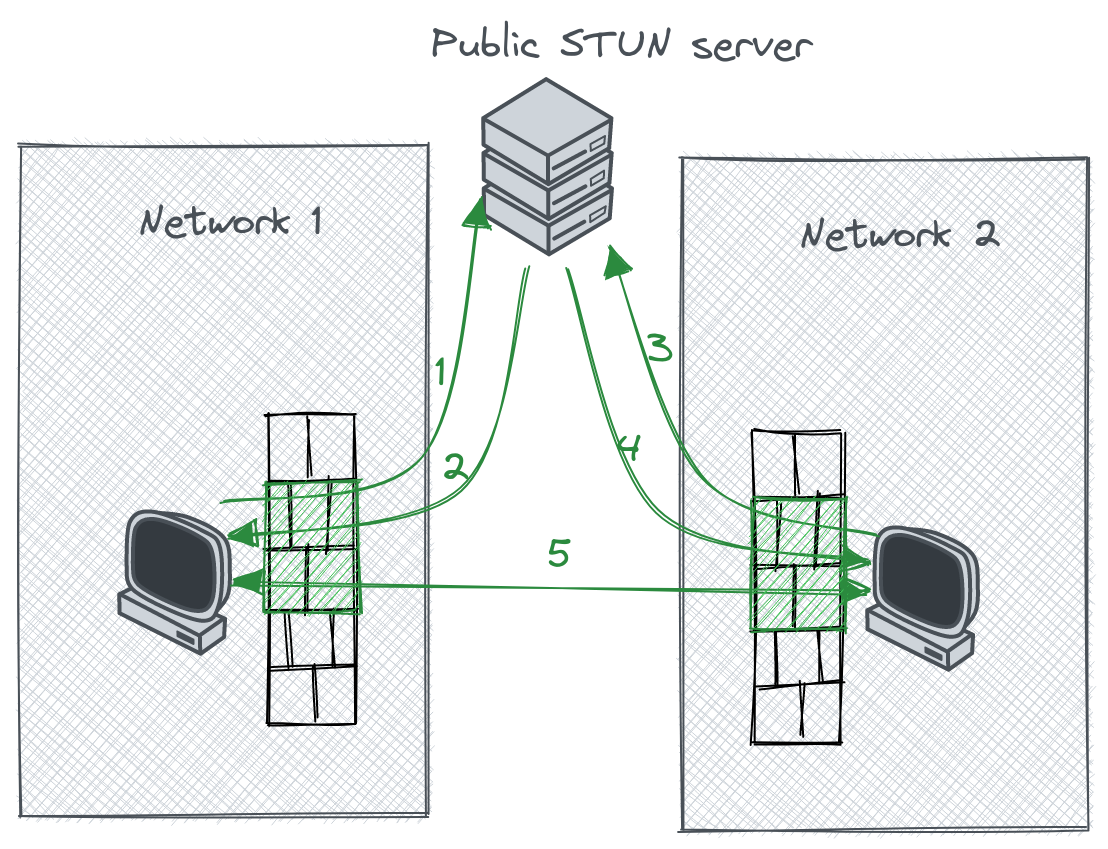
One approach that fits the Client/Server model is for the devices behind NATs to initiate bidirectional connections to a publicly addressable **relay** server that forwards the Peer-to-Peer traffic to the appropriate recipient. Compared to direct communication, relaying results in a higher network latency due to the longer path that each packet must travel. Maintaining a relay server requires some technical expertise and may be costly depending on the expected throughput. Despite the drawbacks, relaying works under most networking scenarios and is therefore often used as a fallback in case all other approaches fail to find a direct path. Protocols such as (**turnRFC?**) and (**derpDocs?**) can be used to securely implement relaying.

The NAT gateway in many residential networks is a Router device under the customer’s control that has a statically or dynamically assigned public IP address. Most routers can be manually configured through their admin page to forward all traffic that arrives at a given port to a specific device on the local network. Remote applications that know the IP address of the router and the forwarded port can then contact it to initiate a connection to the local device. The manual configuration, however, can be inconvenient and many users may be unaware of that setting because it is not necessary for the more familiar Client/Server communications. Some routers also support programmatic configuration of port forwarding via a Layer 7 protocol like or its successors and . Those protocols, however, are not always supported and are often disabled by the local network administrators due to security concerns related to bugs in their implementation, vulnerable IOT devices or malicious programs being able to expose local devices to the internet.

An efficient NAT traversal approach that works with some types of NATs is to use (**stunRFC?**) in combination with UDP hole punching. STUN is a Client/Server protocol operating at Layer 7 that allows applications to detect the presence of NAT gateways on the network path to a public endpoint, to identify their types and discover the public IP address that they map to. To achieve this, the device uses UDP to contact a public STUN server which responds with the source IP address and port of the incoming datagrams. If a NAT gateway rewrote the IP address and port, then the device will know this by comparing them to its local IP address and port. Additional STUN servers may be contacted to check if the NAT gateway maps consistent IP addresses and ports. UDP hole punching is a related technique that allows two devices

to contact detect its own NAT type and the public address of the NAT gateway via, which relies on the predictable port mapping algorithms that many routers use and a public third party host that can be contacted by the local devices and later serve as an introduction point for them (Figure ).

hole punching based on concepts from . The machines of each party can contact a public server , which will note what addresses the connections come from and inform the parties. Since the parties initiated the connection to the STUN server, their routers will keep a mapping between their local IP addresses and the port that was allocated for the connection to be able to forward the incoming traffic. Those “holes” in the NATs were originally intended for the STUN server, but mesh VPNs use the stateless “fire and forget” UDP protocol for their internal communication, which does not require nor provides a mechanism for the NATs to verify who sent a UDP packet. With most NATs, this is enough to be able to (ab)use the “punched holes” for the purpose of traffic from other parties. Mesh VPNs implement the stateful and protocols on top of UDP and expose a regular network interface to the other programs, keeping them shielded from the underlying complexities. Other NAT implementations such as Symmetric NAT and can be more difficult to “punch through” due to their more complex port mapping strategies. In those cases, establishing P2P connections might involve guesswork or even fail and require falling back to routing the (encrypted) traffic via another party or service.



NAT traversal via STUN

In most mobile networks (4G, 5G) the utilizes a as part of their infrastructure, while all devices under the user’s control, including the router, only have local IP addresses. STUN techniques would fail to discover a direct path between two parties behind separate CGNATs or other unpredictable NAT algorithms. The only remaining possibility is for a like , where a publicly reachable third-party host is used not only for introducing the peers but also for relaying all (possibly encrypted) traffic between them.

* Hairpinning
  + Layer 3 protocol suite part of the Internet Protocol Suite
  + used inside VPN software
  + has implementations in both user and kernel space as well as hardware implementations
  + rewrites and encrypts the IP headers and payloads
  + virtual routing table
  + Initially was built into IPv6, separate from IPv4

## 2.2 Network overlays

Most Network Overlay solutions use a combination of the NAT traversal techniques mentioned previously. They can be placed in Layers 2, 3 or 7. Layer 2 overlays act as a virtual network switch, while Layer 3 overlays act as a virtual network router. Layer 7 overlays are implemented in user-space as libraries or applications that run on top of the network stack of the host operating system. Layer 2 and 3 overlays can either be implemented as kernel modules or as user-space applications that use a **TUN/TAP** driver to interface with the kernel.

Figure shows an approximate OSI model mapping of several protocols and network overlay solutions from the point of view of the systems that use them and the arrows show dependency relations between them.

![OSI model mapping of various protocols ](data:application/pdf;base64,)

OSI model mapping of various protocols

### 2.2.1 TUN/TAP driver

* Layer 2 vs Layer 3 Networks
  + Layer 2 overlays bridge networks
    - virtual network switch
    - remote machines are on the same virtual LAN and can share the same IP address range
    - allows broadcast/multicast
    - TAP driver
  + Layer 3 overlays route traffic between separate local networks
    - virtual network router
    - remote machines are on separate LANs
    - simpler to configure
    - TUN driver

### 2.2.2 Traditional VPNs

are implemented as Layer 2 or 3 network overlays. They are commonly used for securely connecting machines from different . They provide software emulation of a network interface controller via a TUN/TAP driver on the operating system level and allow other software to transparently use the functionality of the suite without requiring extra changes. Traditional such as IPSec (**ipSecDocs?**) and OpenVPN (OpenVPN 2022) use a centralized service that all (encrypted) client communications must pass through. This introduces a single point of failure and a potential bottleneck that might negatively impact the performance of the multi-party computations due to their nature.

### 2.2.3 WireGuard

WireGuard (Donenfeld 2017) is a more recent protocol with a design informed by lessons learned from IPSec and OpenVPN and a key management approach inspired by SSH. It is a lower-level protocol that focuses on configuration simplicity while network topology, peer discovery and key distribution are left as a responsibility of higher-level systems that use it as a building block. Wireguard is implemented as a Layer 3 overlay over UDP tunnels. WireGuard has both user-space implementations that use a TUN driver and also has direct support built into the Linux Kernel since version 5.6 (May 2020). The kernel implementation allows for better performance because it does not need to copy packets between the kernel and user-space memory.

The snippets below show a minimal set of configuration options that need to be provided for two peers to be able to form secure tunnels with each other.

# peer1.conf  
[Interface]  
Address = 101.0.0.1/32  
ListenPort = 53063  
PrivateKey = ePTiXXhHjvAHdWUr8Bimk30n0gh3m241RAzsN0JZDW0=  
  
[Peer]  
PublicKey = BSn0ejd1Y3bKuD+Xpg0ZZeOf+Ies/oql0NZxw+SOmkc=  
AllowedIPs = 101.0.0.2/32  
Endpoint = peer1.example.com:38133

# peer2.conf  
[Interface]  
Address = 101.0.0.2/32  
ListenPort = 38133  
PrivateKey = sN/d6XUPEVPGSziVgCCOnOivDK+qAoYC3nxnssQ5Rls=  
  
[Peer]  
PublicKey = e/TxvPmrgcc1G4cSH2bHv5J0PRHXKjYxTFoU8r+G93E=  
AllowedIPs = 101.0.0.1/32

Each peer has a public/private key pair that is used for authentication and encryption based on the Noise Protocol Framework (Perrin 2018). The Address field specifies the virtual IP address that the local network interface will use, while the AllowedIPs field specifies what virtual IP addresses are associated with a peer’s public key. A peer’s Endpoint field specifies the URL at which it can be reached. Only one of the peers must be configured with a reachable endpoint for the other one. In the above example once peer1 initiates communication with peer2, peer2 will learn the current endpoint of peer1 and will be able to communicate back with it.

### 2.2.4 Mesh VPNs

* Tinc
* N2N
* Tailscale
* Nebula
* ZeroTier

Mesh such as Tinc (Sliepen 2022), Tailscale (Tailscale n.d.) and Nebula (Defined 2022) utilize NAT Traversal techniques to create direct links between the clients for the data traffic. Authentication, authorization and traffic encryption are performed using certificates based on public key cryptography.

All three are open-source, except Tailscale’s coordination service which handles peer discovery and identity management. Headscale (Font [2020] 2022) is a community-driven open-source alternative for that component. Tinc is the oldest of the three but has a relatively small community. It is mainly developed by a single author and appears to be more academic than industry motivated. Nebula and Tailscale are both business driven. Tailscale was started by some high-profile ex-googlers and is the most end-user-focused of the three, providing a service that allows people to sign up using identity providers such as Google, Microsoft, GitHub and others. They also provide an Admin console that allows a user to easily add their personal devices to a network or share them with others. It also has support for automation tools like Terraform for creating authorization keys and managing an based firewall. Nebula was originally developed at the instant messaging company Slack to create overlay networks for their cross-region cloud infrastructure, but the authors later started a new company and are currently developing a user-centric platform similar to Tailscale’s. Nebula is more customizable than Tailscale and since it is completely open-source it can be adapted to different use cases, but it is also more involved to set up. A certificate authority needs to be configured for issuing the identities of the participating hosts. Furthermore, publicly accessible coordination servers need to be deployed to facilitate the host discovery. Tailscale employs a distributed relay network of servers, while Nebula can be configured to route via one of the other peers in the VPN.

### 2.2.5 Layer 7 overlays

#### 2.2.5.1 WebRTC is

* WebRTC
  + Uses STUN/TURN
  + d
* OpenZiti
  + uses relays
* ngrok
* TOR
* BitTorrent
* IPFS
* Ethereum
* Teleport
* Freenet

# 3 Testing methodology

In the following chapters, we will design and implement several solutions for ad hoc MPC sessions based on a subset of the previously discussed related works:

* Internet protocol
* Wireguard
* Tailscale
* Headscale
* ? Headscale with DID identity?
* ? WebRTC?
* A custom solution that automates the WireGuard configuration by visiting a web page

Additionally, we will analyze and compare them in terms of performance, security and usability

## 3.1 Measuring performance

During the preparation phase of the project, we developed the framework which simplifies and automates the process of deploying machines in different geographical regions, connecting them via an overlay network and executing multiparty computations between them, where each machine represents a different party.

To summarize, is a set of scripts that use several automation tools:

* Terraform - declarative provisioning
* NixOS - declarative Linux distribution
* Colmena - declarative deployment for NixOS
* PSSH - parallel execution of remote scripts over ssh
* DigitalOcean - a cloud provider

It allows us to quickly provision cloud virtual machines in multiple regions and reproducibly deploy all necessary software for running a multiparty computation over a chosen network overlay solution. The source code of can be found on [GitHub](https://github.com/e-nikolov/mpyc)

Each solution will be deployed using the framework and the performance will be quantitatively measured in terms of the time it takes to execute several MPyC demos. The selected demos have different complexities in terms of communication rounds and message sizes which will allow us to observe their impact on the overall performance.

1. Secret Santa - high round complexity with small messages
2. Convolutional Neural Network (CNN) MNIST classifier - low round complexity with large messages

The demos will be configured at three different input size levels

* Low,
* Medium
* High

Furthermore, the demos will be executed in several networking scenarios:

1. 1-10 parties in the same geographic region
2. 1-10 parties evenly distributed across two nearby regions
3. 1-10 parties evenly distributed across two distant regions
4. 1-10 parties distributed across multiple distant regions

## 3.2 Security

We will analyze aspects such as

* key distribution
* trust model - are there any trusted third parties and what would be the consequences if they are corrupted or breached
* traffic encryption
* identity strength

## 3.3 Usability

For each solution, we will describe the steps that the parties need to perform to execute a joint multiparty computation. Those steps will be analyzed in terms of:

* Complexity - how much technical expertise is expected from the parties to be able to execute the steps
* Initial effort - how much effort is each party expected to put in preparing for their first joint computation
* Repeated effort - after the initial setup, how much effort is required to perform another computation
  + with the same set of parties
  + with another set of parties
* Finalization effort - how much effort is required to finalize the MPC session once it is complete and clean up any left-over artifacts or resources so that the machine of each party is in its original state

# 4 Internet Protocol based solution

This solution focuses on directly using the internet protocol without involving an overlay network. Our goal is to analyze the implications of using only the functionalities that MPyC directly supports to serve as the reference for our later experiments.

## 4.1 Implementation details

We will manually set up the multiparty computations via the public IP addresses of the machines and DNS.

## 4.2 Performance analysis

## 4.3 Security analysis

## 4.4 Usability analysis

# 5 WireGuard based solution

This solution creates an overlay network by manually configuring WireGuard on each machine.

## 5.1 Implementation details

## 5.2 Performance analysis

## 5.3 Security analysis

## 5.4 Usability analysis

# 6 Tailscale based solution

Tailscale is a VPN solution that configures a mesh of direct Wireguard tunnels between the peers.

## 6.1 Implementation details

## 6.2 Performance analysis

## 6.3 Security analysis

### 6.3.1 Trust model

There is a centralized service that deals with the key distribution, which needs to be trusted to provide the correct public keys for the correct parties

### 6.3.2 Identity

Identity is based on third party identity providers such as Microsoft and GitHub

* Magic DNS

## Usability analysis

With tailscale each party needs to

* register a Tailscale account
* Download and install tailscale on the machine they want to run a multiparty computation
* Run tailscale on their machine and logs into their account in order to link it to their own Tailnet
* Share their Tailscale machine with the Tailnets of each of the other parties
* Download the demo they want to run
* Form the flags for running the chosen demo
  + add -P $HOST:$PORT for each party using their Tailscale hostname/virtual IP
* Run the demo

# 7 Headscale based solution

This solution is similar to the Tailscale one, but it uses Headscale - a self-hosted open-source alternative to the closed-source Tailscale coordination service.

## 7.1 Implementation details

## 7.2 Performance analysis

## 7.3 Security analysis

### 7.3.1 Trust model

There still is a centralized service like in the Tailscale solution, but here it is self-deployed.

### 7.3.2 Identity

## 7.4 Usability analysis

Defined. 2022. “Nebula: Open Source Overlay Networking | Nebula Docs.” 2022. <https://docs.defined.net/docs/>.

Donenfeld, Jason A. 2017. “WireGuard: Next Generation Kernel Network Tunnel.” In *Proceedings 2017 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society. <https://doi.org/10.14722/ndss.2017.23160>.

Font, Juan. (2020) 2022. “Juanfont/Headscale.” <https://github.com/juanfont/headscale>.

OpenVPN. 2022. “Community Resources.” OpenVPN. 2022. <https://openvpn.net/community-resources/>.

Perrin, Trevor. 2018. “The Noise Protocol Framework,” July.

Sliepen, Guus. 2022. “Tinc Docs.” November 30, 2022. <https://www.tinc-vpn.org/docs/>.

Tailscale. n.d. “Tailscale.” Tailscale. Accessed November 30, 2022. <https://tailscale.com/kb/>.