On the Design and Implementation of Structured P2PVPNs

David Isaac Wolinsky[*], *Linton Abraham*[•], *Kyungyong Lee*[*], *Yonggang Liu*[*],
Jiangyan Xu[*], *P. Oscar Boykin*[*], *Renato Figueiredo*[*]
[*]*University of Florida,* [•]*Clemson University*

Abstract

Centralized Virtual Private Networks (VPNs) when used in distributed systems have scalability concerns as all traffic must traverse through a central server. In recent years, there has been a paradigm shift towards the use of P2P for VPNs that alleviates the pressure placed upon a central server by allowing participants to communicate directly, relegating the server to handling session management and acting as a relay when NAT traversal fails. Approaches to remove all centralization have turned towards unstructured P2P systems, such as a mesh. These approaches, currently lack the depth in security options provided by traditional VPNs. Additionally, they provide relays when NAT traversal is unsuccessful, but the scalability in current approaches has not been validated.

In order to deal with these issues, we propose and implement a novel VPN architecture using a structured P2P system for peer discovery, session management, NAT traversal, and autonomic relay selection. We relegate the use of a central server for certificate management. Additionally, we provide a design and implementation of the first P2P VPN to support full-tunneling, whereby all non-P2P based Internet traffic routes through a trusted third party.

In the paper, we describe the components of our model as well as present evaluate our reference implementation quantitatively to compare system and networking overheads of the different VPN technologies focusing on latency, bandwidth, and memory. We also discuss our usage of the system over the past 3 years.

1 Introduction

A Virtual Private Network (VPN) provides the illusion of a Local Area Network (LAN) spanning over a public wide area network infrastructure, guaranteeing secure and authenticated communication amongst participants. Common uses of VPNs include secure access to enterprise network resources from remote/unsecure locations, connecting distributed resources from multiple sites, and establishing virtual LANs for multi-player video games over the Internet. In the context of this paper, we focus on VPNs that provide connectivity amongst individual resources, namely, all resources requiring symmetric connectivity will need to be configured with VPN software. Our work is significantly different in scope from approaches which define VPNs " as the 'emulation of a private Wide Area Network (WAN) facility using IP facilities' (including the public Internet or private IP backbones)." [25]. These tend to be clustered: endpoints are connected to the VPN through one of more routers connecting large sets of machines to the virtual network.

A centralized VPN server can be a single point of failure, cause create performance bottlenecks, and can compromise end-to-end traffic. Several alternative approaches that address these issues have been conceived: 1) support for multiple VPN servers for a single VPN [41], 2) the use of P2P connections enabling direct communication amongst peers bypass the server while relying on the server for authentication and session management [28, 32], and 3) the use of unstructured P2P networks to form decentralized VPNs [24, 17, 35, 19].

Existing centralized VPN approaches suffer from one or more of the following issues: reliance on a single session or relay server, non-negligible cost in deploying and maintaining additional session and relay servers, performance bottlenecks due to relay servers in central systems or in P2P systems in lieu of NAT traversal. While P2P VPN approaches handle these they each introduce one or more of the following issues: discovering a peer requires broadcasting over the entire overlay, limited options for security and authorization, lack of information regarding scalability with environments that include traversable and non-traversable NATs, lack of support for full-tunnel VPN mode.

The problems we seek to address with our P2PVPN model include:

- reducing the role of centralization for user authentication in a VPN
- managing members in the VPN
- supporting full-tunneling of Internet traffic in a P2P system
- handling relay selection in lieu of unsuccessful NAT traversal

A brief overview of our solutions to the above problems follows and will be covered in depth in the rest of this paper. To provide fully decentralized run-time connectivity and public-key based management of VPN endpoint membership, we use an automated certificate authority based upon the use of user groups. In the case of full-tunneling, P2PVPNs introduce significantly more complexity since a simple routing table swap as done in central VPNs no longer work. Thus, we investigate three different mechanisms for tunneling all non-P2P-based Internet traffic to our full-tunnel gateway(s). When nodes cannot directly communicate, they seek to connect to peers that are physically close to each other and use them to relay communication. For handling trust issues in P2P, we use the bootstrapping a private P2P system for the VPN whose participants are only the active members of the VPN.

Overall, the main contribution made in this paper is the design and implementation of a novel structured P2PVPN overlay architecture. Many of the individual components of our solution are not novel in themselves, but rather the integration of these components and their interaction results in a system that is unique in supporting:

- automated group-based certificate authority
- support for full-tunneling through multiple gateways
- decentralized NAT traversal with autonomous selection of relays for non-traversable nodes
- use of a private P2P VPN system bootstrapped as a subset of the P2P system

The rest of this paper is organized as follows. Section II gives an overview of current VPN technologies and the efforts to decentralized. Section III introduces P2P structures and our previous work IPOP (IP over P2P). Section IV describes the contributions of this paper, namely a feature-full P2PVPN. In Section V, we discuss our implementation and present evaluation comparing centralized, P2P, and our VPN. Finally, we give some concluding remarks in Section VI.

2 Virtual Private Networks

There exist many different flavors of virtual networking, this paper focuses on those that are used to create or extend a virtual layer 3 network. A few examples of such technologies include Cisco's Systems VPN and AnyConnect VPN Client [15] as well as OpenVPN [41]. In this section, we begin by going in depth on client configuration of VPNs followed by sections describing different VPN server configurations as highlighted in Table 2. Finally we conclude the section by presenting a Table 2.4, which presents actual VPN products stating the VPN server type and presenting some qualitative overview.

Type	Description
Centralized	Clients communicate through one or more servers which are statically configured
Centralized Servers / P2P Clients	Servers provide authentication, session management, and optionally relay support while peers attempt to communicate directly with each other, i.e., P2P links
Unstructured P2P	No distinction between clients and servers, members either know the entire network or use broadcast to discover routes between each other
Structured P2P	No distinction between clients and servers, members are usually within $O(\log N)$ hops of each other via a greedy routing algorithm

Table 1: VPN Classifications

2.1 Client VPN Configuration

In Figure 1, we abstract the common features of all VPNs with focus on the client. The key components of the client are 1) client software that communicates with the VPN overlay directly and 2) a virtual network device. During initialization VPN software begins by authenticating with some overlay agent, optionally it then queries the agent for information about the network such as the network address space, and then starts the virtual network device.

There are many different mechanisms for communicating with the overlay agent. For quick setup, a system may provide a shared secret password or key that is common for the entire network. A more user-friendly and manageable approach re-uses the shared secret mechanism and then adds user accounts and passwords, thus blocking unauthorized users from the VPN, while still making it somewhat difficult for brute force attacks to work, so long as the key remains private. For the strongest level of security, each client can be configured to have a signed-certificate that makes brute force attacks

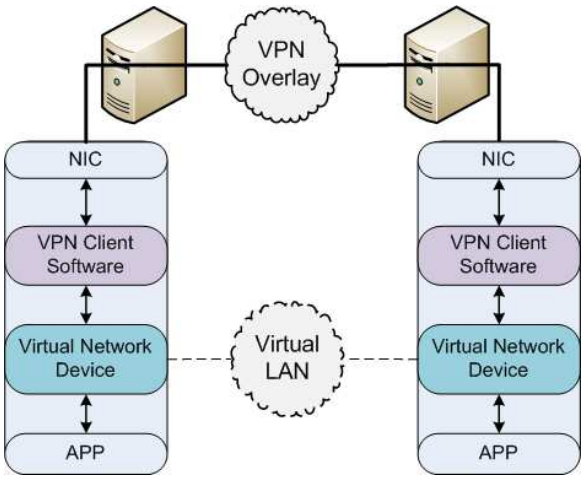

Figure 1: A typical VPN client. The VPN uses a virtual network device to make interaction with the virtual network transparent. Packets that are destined for virtual network destinations are sent via routing rules to the virtual network device, which acts as like a file descriptor read and written to by the VPN client. The VPN client in turn sends and receives packets over the hosts physical (real) network device.

all but impossible. The tradeoffs come in terms of usability. While the use of uniquely sign-certificates may be the most secure, it can be quite difficult for novice computer users. A good balance found in many environments is the mixture of a shared secret and user account, where the shared secret is included with the installation of the VPN application where the application is distributed from a secured site.

Once the user has connected with the overlay, the virtual networking device needs to be configured enabling the user's machine to communicate with other participants in the VPN. This configuration varies by VPN, commonly though, this information contains the network address space, an allocation of an address for the user's machine, and potentially a remote peer for full-tunneling.

In order to communicate over the VPN transparently, there must exist a network device driver that allows common network APIs such as Berkeley Sockets and hence existing application to work without modification. There are many different types of virtual networking devices, though due to our focus on an open platform, we focus on TAP [26]. TAP allows the creation of one or more Virtual Ethernet and / or IP devices and is available for almost all modern operating systems including Windows, Linux, Mac OS/X, BSD, and Solaris. A TAP device exists as a file descriptor providing read and write operations. Incoming packets from the virtual network are written to the TAP device and the networking stack in the OS delivers the packet to the appropriate socket. Packets that are read from a TAP device are those that are sent by sockets to the virtual network.

The virtual network device is either configured using static addressing or dynamically through dynamic host configuration process (DHCP) [18, 10]. This causes a new routing rule that causes all packets sent to the virtual network address space to be sent to the virtual network device. When a packet is read from the TAP device, it can then be sent to the overlay via the client application. The overlay will deliver the packet to another end point, which can be a client or a server enabled with virtual networking stack. When receiving a packet, it will be written to the TAP device. In most cases, the IP layer header will remain unchanged while configuration will determine how the Ethernet header will be handled.

The described configuration so far, creates what is known as a split tunnel, or virtual network connection that only has passes traffic directly related to the virtual network and not Internet traffic. Another form of tunneling exists called full tunneling. Full tunneling allows a VPN client to securely forward all their Internet traffic through a VPN router. This enables a user to ensure all their Internet communication originates from a secure and trusted location and provides some level of security when a user is an insecure and potentially hostile environment, such as an open wireless network at coffee shop.

Most centralized VPNs implement full-tunneling by doing a routing rule swap, where the default gateway becomes an end-point in the VPNs subnet and traffic for the VPN server is routed to the local network gateway. For example, on a typical home network, all traffic for the VPN server is sent via to the networks router and then via the Internet to the VPN server. All other traffic is sent to the virtual network device, then sent securely to the VPN server. In a P2P system, there becomes two new considerations 1) P2P traffic must not be routed to the VPN gateway and 2) there may be more than one VPN gateway. Allowing more than one VPN gateway per VPN allows distribution of the cost of maintaining a full-tunnel who will have 3 additional messages for each incoming packet. We discuss and provide solutions to this problem in Section 4.2.

2.2 Centralized VPN Servers

OpenVPN presents, as its name implies, an open and clear way of implementing a centralized VPN system. While there may be minor differences amongst the different centralized VPN implementations, it is our opinion that OpenVPN provides a reasonable representation of features found in centralized VPN. The key aspects of a centralized VPN server are:

- accepting connections from clients
- routing packets between clients
- providing a NAT to the servers local resources and Internet
- inter-server communication

How do I login? Central VPNs server operate at well-known end-points as in a universal resource identifier (URI) consisting of a hostname or IP address and a port. Clients will randomly attempt to connect with one of the servers until successful, implementing a meek load balance. Once connected, clients obtain an address that is routable in the virtual network address space. Depending on configuration this will allow a client to communicate with other clients, resources on the same network as the server, or Internet hosts via the VPN. There are many, many different ways a client and server can authenticate with each other. For a server to authenicate with a client, the safest way is for the client to have some secure knowledge, such as the server's certificate, retrieved from a secure source, this can then be used to verify the server's identity. The three most common mechanisms for a client to authenticate with a server are via shared secrets, password, or a CA-signed certificate.

Not all OpenVPN servers allow inter-client communication, though the configuration to enable is quite simple. When two clients communicate, the process involves:

1. sender's application sends a packet which writes an outgoing packet to a socket descriptor
2. the packet is written to the Virtual Ethernet device by the OS
3. the VPN stack reads the packet from the Virtual Ethernet device
4. the VPN stack encrypts and signs the packet and sends it to the VPN Server
5. the VPN server decrypts and verifies the packet, encrypts and signs the clear-text message, and sends it to the receiver
6. the receiver's VPN stack receives the packet and decrypts and verifies the packet
7. assuming all is good, the packet is written to the Virtual Ethernet device
8. the OS writes the packet to the packet socket descriptor and the application receives a packet

All packets flow through the central server. As can be seen from the above example, OpenVPN does not by default prevent a server from eavesdropping on client-to-client communication. While it is possible, through a shared secret key, that requires out-of-band communication and is less secure than relying on a CA-signed certificate.

To support full-tunneling or allowing the client to access the server's resources, he too must behave somewhat like a client by enabling a VPN end-point via a VN device. The VN device is configured to support NAT from the VN to the local network and/or the Internet. In Linux, this is achieved with minimal configuration via IPtablesc̆iteiptables, a layer 3 network stack manipulator.

OpenVPN allows a distribution of servers, so as to provide fault tolerance and to a lesser degree load balancing. Servers must be configured to know about each other in advance and need routing rules established to forward packets. Load balancing exists only in the process of the client randomly connecting to different servers and potentially with a server refusing connection due to load. There is no distributed load balancing.

2.3 Centralized P2P VPN Systems

With the advent of Hamachi [28] began the advant of centralized VPNs that went with the ambiguous monicker "P2P VPN". In reality, these systems would be best classified as centralized VPNs servers with P2P clients. Specifically, the nature of P2P in these [32, 27] types of systems provides direct connectivity between clients once authenticated by a central server. While direct connection is desirable, it does not always happen due to firewalls or inpenetrable NATs, when this happens the central server either acts as a relay or the two machines are unable to communicate. One security consideration is that each of these implementations use their own security protocols that involves using the server to verify the authenticity of each other. Most of these projects are closed source, meaning that a user must trust that the server will not act as a man in the middle and eavesdrop. Finally, there has been no work on full-tunneling nor accessing resources on the same network as the other client in any of these cases.

2.4 P2PVPN Client / Server Roles

Unlike centralized systems, pure (or decentralized) P2P systems have no concept of dedicated servers, though it is entirely possible to add reliability to the system by starting an instance of the P2P VPN software purely for enhancing or enabling connectivity. In these systems, all participants are members of a collective known as an overlay. Current generation P2P, decentralized VPNs use a P2P unstructured network, where there are no guarantees about distance and routability between peers. As a result participants tend to be connected to a random distribution of peers in the overlay. Finding a peer requires either global knowledge of the pool or at worst case broadcasting a lookup message to the entire overlay. While unstructured P2P systems have some scalability concerns, P2P systems in general allow for server-less systems. In the realm of VPNs, all client VPNs are also

servers with varying different responsibilities depending on the VPN application, as we present in Table 2.4.

Typically, decentralized, P2P VPNs begin by attempting to connect to well-known end-points running the P2P overlay software, a list of such end points can be easily maintained by occassionally querying the overlay for active participants on public IP addresses and distributed with the application or some other out-of-band mechanism. In the case of P2PVPN, this involves communication with one or more BitTorrent trackers to find other members of the P2PVPN group. N2N requires knowledge of any existing peer in the system. It uses this endpoint to bootstrap more connections to other peers in the system, allowing the application to be an active participant in the overlay and potentially be a bootstrap connection for other peers attempting to connect.

3 Structured Peer-to-Peer Systems

Structured P2P systems provide distributed look up services with guaranteed search time in $O(\log N)$ to $O(\log_2 N)$ time unlike unstructured systems that most either know all the state in the system or make random walks [12]. Some examples of structured systems can be found in [34, 36, 29, 30, 33]. In general structured systems, are able to make these guarantees by self-organization, whereby a node entering the system follows some form of these abstracted steps:

1. generates or obtains a unique identification number (node id) on the order of 128-bits to 160-bits
2. connects to random addresses on a pre-shared well-known end-points list
3. become connected to at least one peer in the list (leaf connection)
4. look up the peers closest in number to its node id connecting to the one immediately smaller and larger than itself (neighbors)
5. connect other nodes in the ring that are further in away in the address space (shortcuts)

The node id must be unique to each peer, otherwise there will be an address collision and the two peers will fight for the same neighbors. Furthermore, having the node ids well distributed will assist in providing better scalability as many algorithms for selection shortcuts depends on having node ids uniformly distributed across the entire node id space. A simple mechanism to ensure this is to have each node use a good, cryptographically strong random number generator. Applying the birthday problem in this context would require between 2^64 to 2^80 peers in a system before there is a 50% chance of there being a collision. Another mechanism involves having a third party generate the node ids and cryptographically signing them [31].

Similarly to the case of unstructured P2P systems, the incoming node must know of at least one participant in the system in order to connect to the system. To summarize what was stated in Section 2.4, a list of nodes that are running on public addresses should be maintained and distributed with the application or be available through some out-of-band mechanism. Other proposals suggest using multicast to find pools [34], this works well except multicast range can be quite limited.

Depending on the protocol, a node must be connected to either his closest neighbor smaller, larger, or both. Optimizations for fault tolerance suggest that it should be upwards of 4 on both sides. If a peer does not know of an address who is his immediate predecessor or successor and a message is routed to him destined for them, depending on the message type, he will assume it was meant for him or throw the packet away. Thus having multiple peers on both sides assist stability when there exists churn in the system where peers freely come and go with out warning.

There are many implementations and proposals for determining shortcuts, each has differing costs associated. A few of these include: maintaining large tables without using connections and only verifying usability when routing messages [34, 30], maintaining a connection with a peer every set distance from you in the P2P address space [36], or using a handful mathematically chosen locations in the node id space [29]. Shortcuts make quickly traversing a P2P structure possible.

4 Components of a P2PVPN

Prior to the discussion of this papers contributions, we will present our previous works as they are the baseline for our P2P VPN. Our previous research involves implementing a P2P system similar to Symphony [29], named Brunet [11]. The specific components of the system that make it interesting for use in a P2P VPN system include:

- system stability when two nodes next to each other in address space cannot directly connect [22]
- selection of shortcuts using information based upon proximity [22]
- a distributed data store based upon a distributed hash table [23]

Furthermore, in previous works, we have discussed and implemented a virtual networking technique with the following features:

- self-configuring, low overhead use [40, 21]
- discovery of members through the use of a virtual DHCP server using DHT [40, 23]
- scalability in the count of hundreds of peers in a single VN [40]

	VPN Type	Authentication Method	Peer Discovery	NAT Traversal	Availability
OpenVPN	Centralized	Certificates or passwords with a central server	Stored at central server(s)	Relay through server(s)	Open Source
Hamachi	Centralized P2P	Password at central server	Stored at central server	NAT traversal and centralized relay	limited free-use, limited Non-Windows clients, no private relays
GBridge	Centralized P2P	Password at central server	Stored locally	NAT traversal, centralized relay	Windows only, freeware, no private relays
Wippien	Centralized P2P	Password at central server	Stored locally	NAT traversal, no relay support	Mixed Open / Closed source
N2N	Unstructured P2P	Shared secret	Broadcast look up	NAT traversal, decentralized relay	Open Source
P2PVPN	Unstructured P2P	Shared secret	Everyone has global knowledge	No NAT traversal, decentralized relay	Open Source
tinc	Unstructured P2P	CA Certificates / Private key	Everyone has global knowledge	No NAT traversal, support of relay through overlay	Open Source
CloudVPN	Unstructured P2P	CA Certificates	Broadcast	Relay through server(s)	Open Source
IPOP	Structured P2P	CA Certificates or pre-exchanged keys	DHT lookup	NAT traversal and relay through physically close peers	Open Source

Table 2: VPN Comparison

- portability to any system that supports Tap and Mono[1] [16]
- ability to behave as a VN interface or router [40]
- used in grid computing for over 3 years [20, 38, 37, 39]

Though this framework provides the basis for our design and implementation, we will attempt to abstract the components as much as possible.

4.1 Security through Groups

A group infrastructruce with one or more group administrators, who verify the incoming members and ensuring proper behavior of current methods removing those that misbehave, can easily be adapted into a certificate authority model. We take this model and use it to support intuitive management of VPN members. The process for follows:

1. in a groups environment, a new group is created
2. when a user requests to join, they give relevant information for the group and agree to some terms of service
3. this triggers an e-mail to be sent to the administrators for the group, which at a minimum will be the groups creator
4. the administrator can either deny or accept the users access
5. assuming access is provided, the user can then go and download configuration information
6. this configuration contains the users personal information and a secret key
7. the user provides the configuration information to the VPN and starts the service
8. on first boot, the VPN connects with the group web server providing a certificate request containing his node id and his secret key, the server verifies the authenticity of the user through the secret and signs the certificate
9. the VPN client connects to the overlay and can now communicate with other members in the group

The two key ambitions to using groups stem from providing a decentralized authentication mechanism and to reduce entry barriers into using VPNs. Through the use

[1] Mono is an open source implementation of .Net also known as the Common Language Runtime (CLR), which provides managed run-time environment similar to Java.

of certificate authority signed certificates, users can verify that they are members of the group by verifying the signature on the exchanged certificate. This removes the need for users to authenticate through a centralized server and removes the security weakness created by using only user name and passwords. Providing the user a secret key in a binary blob used to obtain a signed certificate significantly reduces the entry barrier into using a VPN. The key to this system is ensuring that the binary blob is only exchanged over secure mediums, such as HTTPS, which can be done with no user intervention.

The other key aspect of such an environment is efficiently dealing with misbehaving users. Centralized and Centralized P2P VPNs can do this by terminating the link between two users. Such an approach is impossible in a decentralized system where the group administrator as well as the group site would never be anything more than another member in the P2P overlay. To deal with this problem, we use the following mechanisms:

1. use a certificate revocation list (CRL) hosted on the web group website
2. sign a revocation and place it into the dht, peers can verify they are communicating with trusted peers as often as they like
3. broadcast to the entire P2P system the revocation of the peer

The strength of the CRL is that it can be done out of band and thus users of the overlay may not necessarily be able to prevent a user from updating to the most recent CRL. Furthermore, if a client is unable to retrieve the latest CRL, it will be clear that there is a connectivity issue with the CRL server and it can act appropriately, depending on the security requirements. The downside is that the CRL is centralized and it can be prohibitively expensive for peers to verify certificates on a regular, short periods.

The use of a DHT provides a mechanism that allows peers to regularly verify peers prior to and during connections. DHTs can also be used to implement event notification, so that the CA can retrieve a list of peers who would like to be notified if a peer has had his ceritficate revoked. The problem with a DHT is that they can be easily compromised if they have not been implemented with significant measures to protect against maliciousness.

Finally, the most rudimentary mechanism is broadcasting the certificate revocation over the entire P2P overlay. In small networks, the cost of such a broadcast may be negligible, but as a network grows, such a broadcast may become prohibitively expensive. A broadcast can be harder to block through malicious behavior as a malicious node would need to have significant collusion in order to completely interrupt the broadcast. Furthermore, the broadcast acts similarly to the DHT event notification, as such, peers find out on a push like mechanism.

4.2 Full-Tunneling over P2P

As discussed in Section 2.1, VPN tunneling is usually divided into two categories: split tunneling and full tunneling. With split tunneling, the clients only route VPN destined messages over the VPN; whereas in full tunneling, all traffic both VPN and Internet traffic with the exception of overlay messages route over the VPN. Using full-tunneling ensures that a peer cannot easily eavesdrop into what would otherwise be public communication by forwarding all traffic securely to a third party who resides in a more trusted environment.

There are two key components to this scheme, a server or traffic relayer and a client or a traffic forwarder. In the following sections, we present a simple scheme for providing servers and a few solutions to configuring clients in a structured P2P overlay.

4.2.1 The Server

Prior to becoming a server, a machine must properly be configured, this can easily be done with a Linux machine using IPTables [1] component called masquerade, which will a machine to receive Internet bound packets on one network interface and forward them on another taking care of the responsibilities of changing source and destination Ethernet and IP addresses. The command we use is:

iptables -t nat -A POSTROUTING -s 5.0.0.0/8 -o eth0 -j MASQUERADE

Where 5.0.0.0 is the VPN's lowest address with an 8-bit netmask, thus all addresses between 5.0.0.0 and 5.255.255.255 inclusive would be NATed to the network interface eth0. Additionally, by default, Linux will not forward packets this is enabled by tweaking procfs state via:

echo 1 > /proc/sys/net/ipv4.ip_forward

There are many ways to implement a NAT box, our goal was to get a easily reproducible model, which was well served through this setup.

Once the user has setup NAT capabilities, the VPN server software must be configured to let VPN clients that desire full tunneling where there is a provider. For that purpose, we added an enable flag into the VPN configuration that specifies if the machine is a VPN full tunnel server. If it is, it appends a value to the list of known full tunnel servers for a VPN group. At which point, the

system acts identical to any other VPN system, no additional configuration is necessary.

4.2.2 The Client

VPN Clients wishing to use full tunnel must redirect their default traffic to their virtual network device. Using the above example, the routing rule would look like:

0.0.0.0 5.0.0.1 0.0.0.0 UG 0 0 0 tap0

There does not necessarily have to be an active node online at 5.0.0.1, this can be a "virtual" virtual address, where the result can than be forwarded to any machine in the network for relaying to the Internet. This works because the structure of a message sent to the Internet via a gateway as shown in Figure **??** contains only the Ethernet address of the relayer. In our system, as described in [40], each site uses a single Ethernet address to represent all remote hosts which differ by IP address, this is called a transparent subnet gateway [2]. Furthermore, the smallest address in the subnet is used as a special address for running virtual network services, such as DHCP and DNS. If a node uses that address in the overlay, it will not be routable. With this in mind, the client software need only pay heed to the destination IP address, which points to the Internet. The process of handling virtual network traffic from the point of view of an Internet packet becomes:

1. am I packet destined for some area outside of the virtual network address space
2. if I am, verify that I have full-tunnel VPN enabled
3. if I do, discover a remote peer for full-tunneling
4. if I find one, send the message to the remote peer

Once we've sent the packet to the server, we simply wait for the packet to return. Once the packet returns, we will notice that the source IP address doesn't match the nodes virtual IP address and so we should confirm that we trust this entity to be a full-tunnel relayer. If the packet is sent from an authenticated source, we can write the packet to the virtual network device completing the sending and receiving of an Internet request.

The two issues in the client configuration are selecting the server to use as our relaying partner and how to route P2P packets directly so that they are not relayed through the server. The problem of selecting the server is not well evaluated in our model, our model was to query the DHT for a list of potential servers and selecting one randomly from that list. Furthermore, we check the liveness of the server through a ping like mechanism every 15 seconds, though that could be configured to provide more fault tolerance when servers are actively churning. We apply a pessimistic approach to handling loss of a server, where we will only update the selected server upon the next outgoing Internet packet.

The second issue, handling P2P routed packets, is the focus of our work. In the centralized VPN case, namely when a client only ever communicates to a single point in the VPN overlay, this can be handled by having a single rule in the routing table that routes all packets to this end point via the networks default gateway, like so:

128.227.56.121 192.168.1.1 255.255.255.255 UG 304 0 0 eth0

In this line, the VPN server is located at IP address 128.227.56.121. Our local gateway is located at IP address 192.168.1.1. The use of the netmask 255.255.255.255 ensures that only packets destined for 128.227.56.121 are sent to the machine located at address 192.168.1.1. In a P2P system, ensuring that all P2P overlay messages are routed directly becomes nontrivial because we will have many such routes most of which will not be known ahead of time. If we maintained the model used by the centralized format, we would end up routing both Internet and P2P traffic over the relay machine and if he were disconnected, we would have to reinitialize all state and more importantly we would lose the advantage provided by P2P, namely scalability. To solve this we look at many different possibilities but were only able to come up with one successful model for easily setting up a client. We present all our work, including our failed attempts, such that it may be the ground work for further research.

4.2.3 The Client – Approach 1 – Adding Routes

The first approach taken was to follow a model similar to the centralized version, that is, for each P2P machine we communicated, we had an explicit rule in the routing table. In order to ensure this, we added a feature to the socket handling code in our system that would indirectly add the remote nodes public address to our routing table. This action would occur prior to the first outgoing communication attempt and would remain in effect until the VPN closed down. In order to ensure there were not excessive rules, we check the currently active connections and remove those that are no longer active.

This model works well but has two major flaws. Common to all VPNs that employ the standard route switch technique, all communication, not just VPN, is routed directly to the server insecurely. So while the VPN traffic is most likely encrypted, if the server also hosts a website, that traffic will be in its natural state, potentially unencrypted, and visible to any eavesdroppers. Secondly, a malicious user could send spoof packets to have the VPN add extra routes to the routing table, resulting in a simi-

lar situation as the first problem. The next two solutions attempted to solve these problems.

4.2.4 The Client – Approach 2 – Writing Ethernet Packets

This approach recognizes that when using UDP, we know the source port from which all traffic originates, also even in the case of TCP, we can easily determine our source ports for outgoing connections. While the routing rule directing all traffic to the VPN remains, there are no additional routing rules. Thus all packets are directly sent to the virtual network device and the VPN client software must handle routing the packets appropriately.

To distinguish P2P packets from non-P2P packets, the VPN client needs to look at the source port for the communication, if it matches the VPN client's source port, it must route the packet through the local gateway. In order to do this, we need a component that can write to the hosts physical network device an Ethernet packet to encapsulate the P2P IP packet. Furthermore, we need to modify the IP packet to change the source address to match the physical ethernet devices IP address. We focused on two approaches that would be cross-platform capable: 1) using a bridged tap device whose sole purpose is to send and receive Internet packets to and from the gateway and 2) using PCap [3] to send packets. Additionally, Linux also supports a packet type called packet [4], which allows writing complete Ethernet packets, a feature not enabled in Windows. For this model, we only focused on approaches that were reasonably portable to other OSes.

The problem with this approach is that when an incoming packet arrived at the clients networking stack, the client would not recognize the packet because the destination IP address was the physical ethernet's and not the virtual ethernet's, the one where it originated. This would trigger the OS sending a TCP reset message. Thus the only solution would be to implement some form of NAT, a task we have not started yet.

4.2.5 The Client – Approach 3 – Using Linux Tools

The problem with having the VPN client redirect packets directly is that it can be potentially costly, so we also investigated the potential of using existing Linux tools to solve this problem. IPTables [1] provides the ability to manipulate IP packets to implement firewalls and NATs, we had hoped that we could do redirect our packets to the gateway, somehow.

Using only IPTables, this approach is not possible as we cannot manipulate the Ethernet addresses directly using IPTables. The only way to manipulate the Ethernet addresses is by changing the destination or source IP address. There was no change that could be made that would accomplish what we were looking for.

IPTables had another feature, which was to mark a packet so that it could use a special routing table. The routing table can be managed using iproute2 [5]. We then can make all packets that arrive at the special routing table take a different default route, one that routes it to the local gateway. The problem with this approach is that it does not deal with the source address of the packet. In other words, the packet will have a source IP address of the virtual network device and not of the physical Ethernet device.

4.3 Autonomic Relays

In a handful of P2PVPNs [28, 27], there is support for relaying of communication when direct communication is not possible though in a majority of them this is not the case. All centralized VPNs do not have to worry about direct connectivity as all traffic always routed through a centralized relaying system. Because centralized relaying systems have scalability concerns, we propose a distributed, autonomic relaying system based upon previous work [22]. In this work, we described a mechanism using triangular routing that allowed peers next to each other in the node id space communicate inspite of them being able to communicate directly with each other, whether the cause be firewall, NAT, or Internet disconnectivity issues. To support this behavior, two nodes would discovery each other by indirect communication via the overlay. This would trigger a best effort to exchange peer lists via the current set of near neighbors. In most cases, the peers would have at least one overlapping neighbor and the messages would be exchanged. From thereafter, the peers would use the overlapping neighbors to communicate with each other "directly".

This model worked well in the case that the nodes were neighbors, but when two peers are far away from each other the algorithm will fail. Most high bandwidth and low latency applications can be improved via direct connectivity and in [21], we make a case for its use in VNs. The problem becomes how to make intelligent relaying possible for distant nodes. Our solution, as represented in Figure **??** is to have nodes connect to one one or more of the remote node's neighbors thus creating an overlap and our ability to reuse previous work. There is one difference though, in the previous work, there was no consideration for the viability of the intermediate, the goal was simply to have connectivity regardless of performance. Thus in addition to exchanging neighbor lists, we made it possible to exchange abitrary information to assist in making decisions. So far we have focused on the use of the stability as measured by the age of the connection between the far node and his neighbor, the latency

between the far node and his neighbor, and the optimal overlap based upon network coordinates. We present our test environment, experiment, and results in 5.1.

4.4 Bootstrapping Private Overlays

There have been many papers that discuss handling of distributed security via purely decentralized mechanism. While the work is quite useful, we have also heard many users state they do not trust sending their data over an insecure, untrusted overlay system. Decentralized security provides only probablistic security and has no hard guarantees that only trusted individuals will be involved and aware of your communication behaviors. For that purpose, we suggest bootstrapping a private overlay off of a public overlay. The private overlay will be completely encrypted and authenticated with only members of the VPN allowed access. The use of a public overlay to bootstrap application specific overlay has been discussed in [13].

Beyond security, the attractive features of a private ring for VN include:

- simplified multicast, broadcast routing
- another mechanism for selecting relays
- smaller overlays for faster overlay routing
- a sybil-resistant P2P system

A key feature, which is unexplored in this paper, is that multicast and broadcast routing becomes much easier as all peers in the system would want to receive such packets. Where as in a general P2P overlay, mechanisms such as scribe [14] are necessary to provide multiicast and broadcast capabilities.

Members of the VPN are the only members of the overlay, providing a powerful feature that the entire P2P overlay can be secured through groups. This prevents malicious users outside of the VPN from attacking it and more easily enabled the removal of misbehaving peers, primarily rooted in the fact that the use of a broadcast to signal a certificate revocation is now important to the entire overlay.

4.5 Discovering Faults

In literature, the argument typically directed against structured P2P systems is the ability to handle fault tolerance. In this section, we share our techniques for finding faults in the overlay.

Before we even deploy our code, we heavily use unit testing. If unit testing is successful, we have a virtual time simulator that allows us to test our software in large scale on a single machine rather quickly, where we can simulate well over 1,000 peers for days of virtual time in a matter of hours. The model allows for easy integration testing, so that as new features are incorporated they can easily be tested prior to doing wide-area tests.

The next step is to run a week long test on planetlab that ensures reliability, stability, and consistency, or that a node is congruent with both its first and second left and right neighbors. We call this a crawl. We store this information in a database, and then query the database to find cases where a node may have been inconsistent for consecutive times. If a node is able to fix inconsistencies in future crawls, the inconsistency was probably due to churn in the system. Otherwise, it will probably still be in an inconsistent state and we are able to query the node and other nodes nearby for state information. This would at least help us determine if there exists a problem. To assist in finding bugs, we have added liveness messages to threads to assist in finding deadlocks. Additionally, Mono can print out trace logs by sending a "USR2" signal to the running process.

Other information covered under stability include peer count as well as memory and CPU usage. Node count can be quite difficult to keep track of in Planet-Lab as machines at a rate of about 5 to 20 a day are restarted and software is not automatically restarted on these machines. Thus we check for extreme cases, where node count may have dropped by a significant amount and does not follow a linear model. Planet-Lab also places challenges on memory, as the systems can often be I/O starved causing what appears to be memory leaks as Brunet's internal queue can grow without bound. After solving this via a disconnect on overload, that occurs when the queues moving average exceeds a threshold of 4,096, the memory usage on planetlab has helped us discover that memory leaks exist. The advantage of Planet-Lab as a test ground is that it presents so many unique situations that can bevery difficult to reproduce in a lab controlled test system. It is our belief that any system that uses large scale Planet-Lab deployments as a testing ground will be quite reliable.

We are still actively seeking better ways to verify the state of our system. For example, the cost of doing a crawl can be on the order of $O(N \log(N))$, since we have to communicate with every single node with an average routing time of $O(\log(N))$. Furthermore, many of mechanisms employed in our system though originally based in mathematical principles have since been heavily influenced by experience that is difficult to formally justify.

4.6 P2PVPN in Other Structured Overlays

The purpose in this work is to develop a model of a P2PVPN that can easily be applied to other structured P2P systems. In this section, we focus on the viability of our platform running on top of other sturctured P2P systems, namely Pastry and Chord by analyzing FreeP-

astry [6] and NChord [7] respectively. FreePastry can easily reuse our C# implemented library through the use of IKVM.NET [8], which allows the porting of Java code into the CLR. NChord is a Chord implementation written in C#. In Table 4.6, we compare the features of the structured P2P systems as they apply to the use as a VPN.

To configure our model to run on NChord, we would first need to find the owner of the DHT key containing the mapping of virtual IP address to node address through "find_successor". Then we can query the owner of the key for the value, which would be an address of the node owning the virtual IP. We then need to execute "find_successor" on that address so that we determine the physical IP address who owns the virtual IP address. Then we would need to "connect" to the remote node using either a UDP or TCP socket. Virtual IP messages would then be sent and received through this "connection".

With FreePastry, we, first lookup the mapping of virtual IP to node id from PAST. Afterward, we can use the "route" call to send packets and "deliver" to handle incoming packets. In fact, the model is very similar to Brunet. Furthermore, FreePastry has knowledge of proximity in shortcuts, so it may be very easy to apply high-performance autonomic relays to pastry.

5 Evaluation of VPN Models

For the purpose of quantitatively evaluation, we have added the features of the proposed design parameters described in Section 4 to IPOP [40] and Brunet [11]. We begin by examing the effects of different relay selection mechanisms. Afterwards, we evaluate the system overheads of OpenVPN, Hamachi, and our P2P VPN determine the OS resource costs and the cost of each in a distributed environment.

5.1 Comparing Relay Selection

5.2 Comparing System Overheads

6 Conclusions

References

[1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

[9]

[10] S. Alexander and R. Droms. *RFC 2132 DHCP Options and BOOTP Vendor Extensions*.

[11] P. O. Boykin and et al. A symphony conducted by brunet. `http://www.citebase.org/abstract?id=oai:arXiv.org:0709.4048`, 2007.

[12] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 85–98, Berkeley, CA, USA, 2005. USENIX Association.

[13] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. One ring to rule them all: Service discover and binding in structured peer-to-peer overlay networks. In *SIGOPS European Workshop*, Sept. 2002.

[14] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. volume 20, October 2002.

[15] Cisco. Cisco vpn. `http://www.cisco.com/en/US/products/sw/secursw/ps2308/index.html`, March 2007.

[16] M. de Icaza and et al. The mono project. `http://www.mono-project.com`, September 2009.

[17] L. Deri and R. Andrews. N2N: A layer two peer-to-peer vpn. In *AIMS '08: Proceedings of the 2nd international conference on Autonomous Infrastructure, Management and Security*, pages 53–64, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] R. Droms. *RFC 2131 Dynamic Host Configuration Protocol*, March 1997.

[19] E. Exa. CloudVPN. `http://e-x-a.org/?view=cloudvpn`, September 2009.

[20] R. Figueiredo and et al. Archer: A community distributed computing infrastructure for computer architecture research and education. In *CollaborateCom*, November 2008.

[21] A. Ganguly, A. Agrawal, O. P. Boykin, and R. Figueiredo. IP over P2P: Enabling self-configuring virtual IP networks for grid computing. In *International Parallel and Distributed Processing Symposium*, Apr 2006.

[22] A. Ganguly and et al. Improving peer connectivity in wide-area overlays of virtual workstations. In *HPDC*, 6 2008.

[23] A. Ganguly, D. Wolinsky, P. Boykin, and R. Figueiredo. Decentralized dynamic host configuration in wide-area overlays of virtual workstations. In *International Parallel and Distributed Processing Symposium*, pages 1–8, March 2007.

[24] W. Ginolas. P2PVPN. `http://p2pvpn.org`, August 2009.

[25] B. Gleeson, A. Lin, J. Heinanen, T. Finland, G. Armitage, and A.Malis. RFC 2764 a framework for IP based virtual private networks. February 2000.

System	Overlay messaging	NAT Traversal	DHT	Secure PtP	Secure EtE
Brunet	Yes (AHSender)	UDP and overlay relaying	Yes with reliability	Yes	Yes
FreePastry	Yes (route)	Only with port-forwarding enabled	Yes, PAST [9], reliable	No	No
NChord	No, only lookup	No	Simple, non-fault tolerant DHT	No	No

Table 3: A comparison of structured P2P systems. PtP stands for point-to-pont communication, such as communication between physical connections in a P2P overlay. EtE stands for end-to-end communication, such as messages routed over the overlay between two peers.

[26] M. Krasnyansky. Universal tun/tap device driver. `http://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt`, March 2007.

[27] G. LLC. Gbridge. `http://www.gbridge.com`, September 2009.

[28] LogMeIn. Hamachi. `https://secure.logmein.com/products/hamachi/vpn.asp`, July 2008.

[29] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: distributed hashing in a small world. In *USITS*, 2003.

[30] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '02*, pages 53–65, 2002.

[31] A. G. A. R. Miguel Castro, Peter Druschel and D. S. Wallach. Security for structured peer-to-peer overlay networks. In *5th Symposium on Operating Systems Design and Implementaion (OSDI'02)*, December 2002.

[32] K. Petric. Wippien. `http://wippien.com/`, August 2009.

[33] S. Ratnasamy, P. Francis, S. Shenker, and M. Handley. A scalable content-addressable network. In *In Proceedings of ACM SIGCOMM*, pages 161–172, 2001.

[34] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[35] G. Sliepen. tinc. `http://www.tinc-vpn.org/`, September 2009.

[36] I. Stoica and et al. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *SIGCOMM*, 2001.

[37] D. Wolinsky and R. Figueiredo. Simplifying resource sharing in voluntary grid computing with the grid appliance. In *2nd Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid 2008) with IPDPS*, 2008.

[38] D. I. Wolinsky and et al. On the design of virtual machine sandboxes for distributed computing in wide area overlays of virtual workstations. In *VTDC*, 2006.

[39] D. I. Wolinsky and R. Figueiredo. Grid appliance user interface. `http://www.grid-appliance.org`, September 2009.

[40] D. I. Wolinsky, Y. Liu, P. S. Juste, G. Venkatasubramanian, and R. Figueiredo. On the design of scalable, self-configuring virtual networks.

[41] J. Yonan. OpenVPN. `http://openvpn.net/`, March 2007.