On the Design and Implementation of Structured P2PVPNs

David Isaac Wolinsky*, *Linton Abraham*$^{\bullet}$, *Kyungyong Lee**, *Yonggang Liu**, *Jiangyan Xu**, *P. Oscar Boykin**, *Renato Figueiredo**
**University of Florida*, $^{\bullet}$*Clemson University*

Abstract

In recent years, there have been a multitude of P2P VPNs primarily falling into two categories: 1) P2P communication but centralized session management and 2) use of unstructured P2P systems. These tend to reduce the overhead in creating a VPN and allow users to directly connect bypassing the overhead of communicating through a third party proxy. This significantly improves upon classical, centralized VPNs, though it adds a new complexity as the approaches have varying scalability constraints. To address this, we propose a run-time decentralized P2P model using a structured P2P system. In this paper, we will describe the components of this model as well as present and evaluate our reference implementation. Our approach provides an intuitive and simplistic setup, reduces the requirements for connectivity, offers better proxy selection in lieu of NAT traversal, and provides an opportunity for more intuitive trust solutions. For evaluation, we will compare system and networking overheads of the different VPN technology focusing on latency, bandwidth, and memory.

1 Introduction

A Virtual Private Network (VPN) provides the illusion of a Local Area Network (LAN), namely direct communication, over a wide area network such as the Internet while guaranteeing secure and authenticated communication amongst participants. Common uses of VPNs include accessing company or academic network resources while traveling abroad, playing LAN based video games over the Internet, connecting distributed resources from multiple sites, and securing your Internet traffic while in unsecure locations. In the context of this paper, we focus on VPNs that provide connectivity between individual resources and so all resources that need symmetric connectivity will need to be configured with VPN software. Our work is significantly different than others which define VPNs “ as the ’emulation of a private Widea Area Network (WAN) facility using IP facilities’ (including the public Internet or private IP backbones).” [16]. Those VPNs tend to be clustered where a single site may have a few virtual routers and many nodes on a subnet unique to the rest of the VN, these virtual routers are connected to other environments similarly configured systems.

Centralized VPNs enable distributed connectivity but at the cost of maintaining a central server, which becomes the conduit for all traffic, becoming a performance bottleneck and can make end-to-end security complicated. To alleviate this, there have been three directions 1) support for multiple VPN servers for a single VPN [32, 10], 2) the use of P2P connections for bypassing central communication that rely on run-time central authentication [19, 23], and 3) the use of unstructured P2P networks to form VPNs based upon shared secrets without user authentication and limitations on network size [15, 8, 26]. In this paper, we present a novel approach to forming secure, scalable, efficient, and self-configuring VPNs through the the use of Structured P2P systems that has no reliance on centralized systems after initialization. Structured P2P technology enables users to communicate directly with all users without knowing anything beyond their virtual IP bypassing the need for centralization while providing all-to-all communication without maintaining all-to-all connectivity with participants. Interesting applications of P2P include efficient wide area multicast, data distribution, storage, chat applications, and even IP connectivity.

Current generation P2PVPNs do not scale well, provide features such as full-tunneling of network traffic, such as forwarding Internet traffic, nor do they have intuitive ability for scalable multicast or broadcast. P2PVPNs rely on direct connectivity and in general will not work if NAT (Network Address Translation) traversal between peers is unsuccessful. Unstructured P2P based VPNs have similar issues, though have the ability to reuse the unstructured overlay to relay packets though with scalability constraints as this requires complete knowledge of the system, all-to-all connectivity, or use of broadcast messages to find a destination. Furthermore, current unstructured P2P systems lack the ability to police participants in the system.

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

- reducing the role of centralization for user authentication in a VPN
- managing participants in a live system
- supporting full-tunneling of Internet traffic in a P2P system
- handling relay selection in lieu of unsuccessful NAT traversal
- supporting multicast and broadcast communication
- debugging a P2P overlay

A rudimentary 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 policing, 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, as such we investigate three different mechanisms for tunneling all Internet traffic to our full-tunnel endpoint(s) besides our P2P traffic. When nodes cannot directly communicate, they seek to connect to peers that are mutually physically close to each other and use them to relay communication. For efficient multicast and broadcast communication, we rely on the use of bootstrapping a private P2P system whose members are only participants of the VPN.

Explicitly, our contributions made in this paper are:

- automated group-based certificate authority
- three different approaches to configuring full-tunneling
- intelligent selection of relays
- use of a private P2P VPN system bootstrapped of a general P2P system
- discussion of our techniques used for discovering bugs in our system

We would like to note that many of our contributions are not novel by themselves, but rather, it is the culmination of these components and their interaction, namely a model of a structured P2P VPN, that is our novel contribution for this paper.

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 [6] as well as OpenVPN [32]. 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
Decentralized Servers	Clients communicate through one ore more servers which are dynamically 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.

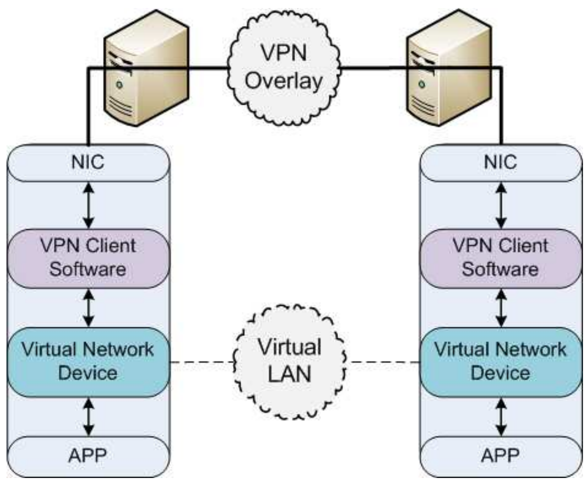

Figure 1: A typical VPN client. The VPN uses a virtual network device to make interaction with the virtual network transparent. Packets that are destined for virtual network destinations are sent via routing rules to the virtual network device, which acts as like a file descriptor read and written to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

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 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 [17]. 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) [9, 1]. 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 4.3.

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 IPtables c̃iteiptables, 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 [19] 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 [23, 18] 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 [3]. Some examples of structured systems can be found in [25, 27, 20, 21, 24]. 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 [22].

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 [25], 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.

4 Components of a P2PVPN

4.1 Background

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 [25, 21], maintaining a connection with a peer every set distance from you in the P2P address space [27], or using a handful mathematically chosen locations in the node id space [20]. Shortcuts make quickly traversing a P2P structure possible.

	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
CloudVPN	Decentralized	CA-signed Certificates	Broadcast	Relay through server(s)	Open Source
Hamachi	Centralized P2P	Password at central server	Stored at central server	NAT traversal and relay through central server	personal use only, no private servers
GBridge	Centralized P2P	Password at central server	Stored locally	NAT traversal and relay through central server	free to use, close source, no private relays, Windows only
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 and support of relay through overlay	Open Source
P2PVPN	Unstructured P2P	Shared secret	Everyone knows about everyone else	No NAT traversal, support of relay through overlay	Open Source
tinc	Unstructured P2P	CA Certificates / Private key	Everyone knows about everybody	No NAT traversal, support of relay through overlay	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

Our previous research involves implementing a P2P system similar to Symphony [20]. The specific components of the system that make it interesting for use in a P2P VPN system include:

- system stability in lieu of two nodes next to each other in address space cannot directly connect [13]
- selection of shortcuts using information based upon proximity [13]
- a distributed data store based upon a distributed hash table [14]

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

- self-configuring, low overhead use [31, 12]
- scalability in the count of hundreds of peers in a single VN [31]
- portability to any system that supports Tap and Mono [7]
- ability to behave as a VN interface or router [31]
- used in grid computing for over 3 years [11, 29, 28, 30]

Though throughout the following sections we will attempt to be as abstract as possible, this framework provides the basis for our design and implementation of our P2P VPN.

4.2 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 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.

4.3 Full-Tunneling over P2P

As discussed in 2.1, generally, tunneling technology can be divided into two forms: split tunneling and full tunneling. With split tunneling, the VPN clients can only have the VPN connections with other entities that also reside in the same network. But with full tunneling, the clients' Internet traffic is also forwarded through the VPN network.

The full tunneling scheme is based on the IPOP infrastructure, which is introduced in section III. An entity such as a work group, organization, or institute who wants to construct a VPN network must construct the group ID and certificate, which are used by the gateway servers and clients to identify and authenticate each other. At least one VPN gateway server is needed for one group. All packets between the client-running applications and the Internet web servers will be forwarded securely through the VPN network, and routed by the VPN gateway server. The packet forwarding in VPN network is based on Brunet, which means the packets may be forwarded via the peers not belonging to the group. But since the packets are encrypted, malicious peers are not able to eavesdrop them or modify them while the receiver not knowing it.

On the gateway client, all Internet packets coming from the general applications (that means, except the VPN process) are directed to a local TAP interface. The IP of the TAP interface is a virtual one that is unique within the VPN network. The VPN client captures all packets sending through the TAP interface, encapsulates them into the VPN packets and encrypts them with the group certificate. Finally these packets are sent through the physical interface. When the VPN client receives a VPN packet, the packet is decrypted, and if a Internet packet is encapsulated within it, the Internet packet will be extracted and injected directly to the TAP device for the applications to receive.

In deployment, we need to enable a TAP device for the applications to communicate with their Internet servers, while still having a physical network interface running for the VPN communication. So one problem the gateway client has to solve is how to configure the Routing Table to direct all the general application packets to the TAP device while directing all VPN packets to the physical device (and routed to the local gateway). We have three different approaches to implement the gateway client routing.

In this paper, our evaluation is based on the client implementation according to approach 1.

4.3.1 The Server

If multiple servers exist, the client will select them randomly or depending on the load balance. The gateway server is very different from a generic one in two aspects: first, the packets between the clients and servers are in the VPN fashion (encapsulated and encrypted); second, the servers do a NAT on the packets sending though them. One Internet packet from the gateway clients is encapsulated into the VPN packet which is encrypted by the group certificate. When the gateway server receives that packet, after decryption and extraction, it is able to get the original Internet packet. The gateway server does a NAT on the packet's source IP:Port to map the original (virtual) IP:Port to the server's IP:Port. Then the gateway server sends it out, and the reply will be sent back to the gateway server. The gateway server does a deNAT on the packet's destination IP:Port, after that, the packet is encapsulated into the VPN packet, encrypted, and sent back to the previous gateway client.

For the deployment on Linux, the gateway server adopts the functionality of ”iptables”. When the gateway server eventually gets one original Internet packet from the client, whose source IP is still the original (virtual) one, it just sends it out. Then ”iptables” module on the server is to do a MASQUERADE NAT on POSTROUTING stage, to modify the source IP:Port to the physical one available on the server. When the reply comes back, the ”iptables” will deNAT the packet automatically, so the gateway server can easily see the virtual IP of the client that should receive this packet. The gateway server encapsulates the reply packet into a VPN packet, encrypts it, and finally sends it back to the client.

4.3.2 The Client – Approach 1

Simply rewrite the routing table will direct the traffic well in general: 1.Make the default gateway in routing table fall into the virtual network IP space, i.g., The virtual network IP space is 5.0.0.0/8, we set the default gateway to be 5.0.0.1. 2.Add routing rules to route all IP's that the VPN process connected to the the local gateway. The general applications will send their packets to the TAP device since the TAP device is within the same (virtual) subnet as the default gateway. And all the VPN traffic will be directed to the physical device because of the added entries in the routing table. This is a lightweight approach to do all the work. But there are two flaws in this solution: First, the routing table will be very big if the VPN connection number gets large. And second, the routing is only based on the packets' destination IP, and sometimes it is very imprecise. Imagine a case that one client Alice has VPN connection with Bob, Bob also runs some web server at TCP port 80. If Alice uses web browser to access Bob's web server, the traffic will not go through the VPN channel, instead, it will be directed to the local gateway because the routing table says ”Route all the packets to Bob via the local gateway.”

4.3.3 The Client – Approach 2

To solve the problem in approach 1, we have a more sophisticated approach, which is routing based on the port number. ”iptables” provides the functionality of marking packets based on the source port number, and ”iproute2” supports multiple customized routing tables and is able to direct packets to different routing tables based on the firewall marks. So the following configuration will show how ”iptables” and ”iproute2” work together. 1.Make the default gateway in routing table fall into the virtual network IP space. 2.Configure iptables to set the packets coming from VPN process port with firewall mark X. 3.In iproute2, build routing table vpn_table with the local gateway as default gateway. 4.Configure iproute2 to direct the packets with firewall mark X to routing table vpn_table. This approach will ensure that only the VPN packets are routed to the local physical gateway. But still, approach 2 and approach 1 may suffer from some efficiency drawbacks, and they only work on Unix / Linux operating systems.

4.3.4 The Client – Approach 3

There is an approach not even bothering the routing table any more. In this approach, we write raw packets (frames) and inject them directly into the physical Internet interface with the destination MAC address (gateway) already configured. The ”pcap” library, which works for both Linux and Window is adopted to do the injection. So in this approach, when the VPN process wants to send a packet, it generates a link layer frame for it, and then injects the frame directly into the physical device by the ”pcap” library. But when the reply comes back, problems emerges. The reply packet is dropped by the Operating System. This is not strange because as ”pcap” bypasses the routing table, it also bypasses the OS network stack, so the OS network stack doesn't have such thread responsible for receiving the reply. Fortunately (somehow), ”pcap” is able to capture these reply packets, and then these packets can be processed by the VPN process. This approach should work in theory if we do enough work, but we did not bothering implementing it because this type of communication makes the network very strange, and brings a lot of management problems (such as the flood of ”tcp reset” packets). Though this approach has the potential of working for both Linux and Windows, as well as good efficiency.

4.4 Autonomic Relays

In a handful of P2PVPNs [19, 18], 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 [13]. 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 [12], 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.

4.5 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 [4].

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 [5] 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.

4.6 Discovering Faults

In structured P2P systems, improper formation of the overlay structure would be regarded as a fault. In a structured P2P ring system[2, 25], incorrect neighbor or routing table information may result in routing delay. To keep the neighbor or routing table information vivid, Pastry [25] uses soft state to keep the routing table containing proper routing information. Thus, if a faulty node is not accessed while routing, the stale information remains in the routing table until the failure is discovered. To detect incorrect neighbor information in [2], a temporary master node contacts every node in the overlay network, and asks each nodes neighbor information. After gathering all neighbor information, temporary master node can calculate neighbor information correctness using following steps :

1. Check node As left-most neighbor information(e.g. As left1) 2. Check As left1s right-most neighbor information(e.g. As left1s right1) 3. If As left1s right1 is same as A, two nodes are regarded as having correct neighbor information.

Same step can be applied to As second left neighbor information. After gathering whole nodes neighbor information, temporary master node can instruct a node which has incorrect view of neighbor information to adjust its neighbor information correctly to prevent future incorrect routing. We can check neighbor information further than 2 steps away at the cost of processing time.

We simulated node neighbor information correction check using other methods. For example, we followed above step by comparing node As left2 neighbors right1 address and node As left1 address. This method misses some special cases of incorrect view of ring. Thus, we regard comparing left2 and left1 neighbors right2 and right1 neighbor information respectively is a good metric to measure ring neighbor information correctness.

We found out that calculating ring neighbor information correctness value does not depend on the direction of measuring method. With given neighbor information, neighbor relation arrow is already shaped before measuring the correctness value. Thus, measuring correctness value in clockwise direction or counter clockwise direction does not affect the correctness value.

However, above method needs O(n) cost, when there are n nodes in the overlay network. Because this method does not seem to be scalable, we need to improve scanning method, which would be one of our future works.

4.7 P2PVPN in Other Structured Overlays

Besides Brunet, the P2PVPN and its virtual networking substrate can also be implemented on top of other popular P2P overlays such as Chord and Pastry with the same idea. Both Chord and Pastry provide features of structured P2P overlay described in Section IV with some differences in route selection, etc. Both of them have Mono runtime compatible libraries available. NChord is a C# implementation of Chord and FreePastry is a Java implementation of Pastry, which can be ported to Mono using IKVM.NET. FreePastry comes with a PAST DHT storage implementation while NChord only has a sample version of DHT with no fault tolerance.

To implement virtual networking, we need to deliver packets to the destination identified by its virtual IP. As there is no recursive routing for application traffic in NChord, we need to first find out its real (physical) IP address. For that, we need to do two lookups.

- First, when we capture a packet from the TAP device, we look at the destination IP and then look up the P2P ID of the destination from the DHT;
- Then we look up the real (physical) IP of the given P2P ID from the overlay using the find_successor operation provided by Chord.

Then we send packets to the real destination IP with the destination port unchanged.

With Pastry, after we look up the P2P ID from the DHT, we can directly route a packet to this ID by calling its route(message, id) API. The packet is then delivered to the node with the ID (or the node with a numerically closest ID). Brunet, like Pastry, also support this recursive overlay application packet delivery.

Pastry takes locality of nodes into consideration in route selection. When a node has multiple candidates for the next hop selected using the prefix-matching mechanism, it routes the packet to the one that is closest according a scalar proximity metric. However, Pastry expects the application on top of it to provide such a function to get the metric. In FreePastry, there is a simple implementation of proximity provider upon roundtrip time between nodes. Brunet, on the contrary, has Network Coordinates built-in which can be used to determine the distance more efficiently.

Also, neither Pastry nor Chord deals with NAT traversal. They both assume that nodes operate on public IPs. NAT traversal is a feature missed in most existing P2P overlays but addressed in Brunet.

Overall, we think it is possible to implement the virtual networking substrate above other existing P2P overlays but Brunets has an edge on them.

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 [31] and Brunet [2]. 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] S. Alexander and R. Droms. *RFC 2132 DHCP Options and BOOTP Vendor Extensions*.

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

[3] 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.

[4] 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.

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

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

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

[8] 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.

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

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

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

[12] 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.

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

[14] 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.

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

[16] 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.

[17] 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.

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

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

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

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

[22] 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.

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

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

[25] 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.

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

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

[28] 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.

[29] 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.

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

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

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