

On the Design and Implementation of Structured P2PVPNs

David Isaac Wolinsky*, Linton Abraham#, Kyungyong Lee*, Yonggang Liu*,
P. Oscar Boykin*, Renato Figueiredo*
*University of Florida, #Clemson University

ABSTRACT

In recent years, P2P VPNs have become quite popular by allowing users to connect directly with each other bypassing the overhead of communicating through a third party proxy. These P2P VPNs require connecting to a central server for authentication, NAT traversal, and proxying in the off chance NAT traversal fails. This significantly improves upon classical, centralized VPNs, though it adds a new complexity either maintenance of all-to-all connections during run-time or the involvement of a centralized authentication entity for each live connection attempt. For this solution, we propose a completely run-time decentralized P2P model based upon a structured P2P system. In this paper, we will describe the components of this model as well as present and evaluate our reference implementation. A decentralized P2PVPN has 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, CPU, 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SC09 November 14-20, 2009, Portland, Oregon, USA
(c) 2009 ACM 978-1-60558-744-8/09/11 \$10.00.

While traditional VPNs enable such distributed connectivity they do so at the cost of maintaining a central server, which becomes the conduit for all traffic, becoming a performance bottleneck and potentially removing end-to-end security. To alleviate this, there have been three directions 1) support for multiple VPN servers for a single VPN [27, 13], 2) the use of P2P connections for bypassing central communication that rely on run-time central authentication [20, 22], and 3) the use of unstructured P2P networks to form VPNs based upon shared secrets without user authentication and limitations on network size [18, 11, 23]. In this paper, we present a novel approach to forming secure, scalable, efficient, and self-configuring VPNs through 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 typically relying on all-to-all connectivity in the system. Furthermore, unstructured P2P systems currently 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

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 [10] as well as OpenVPN [27]. In this section, we begin by going in depth on client configuration of VPNs, overview concepts of server roles in centralized VPNs, and then roles of participants in P2P VPNs. Finally we conclude the section by presenting a table ?? which compares qualitatively the features of VPNs that fit in these categories.

2.1 Client VPN Configuration

In figure ??, 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 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 [19]. 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) [12, 8]. 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 in 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 endpoint 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 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 represents the typical features of a centralized VPN and then some. 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, other changes may be instituted by the VPN client software depending on the services provided. There are many, many different ways a client and server can authenticate with each other. For a server to authenticate 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`/`ipchains`, 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 [20] began the advent of centralized VPNs that went with the ambiguous monicker "P2P VPN". The P2P provided by these [22, 1] types of systems is direct connectivity between peers bypassing a central server. While direct connection is desirable, it does not always happen due to firewalls or impenetrable 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-tunnel P2P VPNs in any of the P2P VPN fields.

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 ??.

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 occasionally 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 [2]. Some examples of structured systems can be found in [3, 24, 21, 4, 5]. 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 $2d$ to $2\sqrt{d}$ 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 [9]

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 [3], 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

	VPN Type	Authentication Method	Peer Discovery	NAT Traversal	Availability
OpenVPN	Centralized with multiple servers	Certificates or passwords with a central server	Stored at central server(s)	Relay through server(s)	Open Source
CloudVPN	Centralized with multiple servers	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 1: VPN Comparison

only verifying usability when routing messages [3, 4], maintaining a connection with a peer every set distance from you in the P2P address space [24], or using a handful mathematically chosen locations in the node id space [21]. Shortcuts make quickly traversing a P2P structure possible.

Our previous research involves implementing a P2P system similar to Symphony [21]. 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 [16]
- selection of shortcuts using information based upon proximity [16]
- a distributed data store based upon a distributed hash table [17]

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

- self-configuring, low overhead use [?, 15]
- scalability in the count of hundreds of peers in a single VN [?]
- portability to any system that supports Tap and Mono [6]
- ability to behave as a VN interface or router [?]
- used in grid computing for over 3 years [14, 26, 25, 7]

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

4.3 Full-Tunneling over P2P

4.4 Autonomic Relays

In a handful of P2PVPNs [20, 1], 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 [16]. In this work, we described a mechanism using triangular routing that allowed peers next to each other in the node id space communicate in spite 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 discover 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 [15], 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 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 arbitrary 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 probabilistic 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 [?].

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 free P2P system

A sybil-resistant system is made available through the use of the P2P group architecture, whereby malicious peers may be able to misbehave for a short period of time. They will be quickly detected, and through the use of certificate revocation, removed from the overlay.

4.6 Discovering Faults

5. EVALUATING VPN MODELS

5.1 Comparing Relay Selection

5.2 Comparing System Overheads

6. CONCLUSIONS

7. REFERENCES

- [1]
- [2]
- [3]
- [4]
- [5]
- [6]
- [7]
- [8] S. Alexander and R. Droms. *RFC 2132 DHCP Options and BOOTP Vendor Extensions*.
- [9] M. Castro and et al. Secure routing for structured peer-to-peer overlay networks.
- [10] Cisco. Cisco vpn. <http://www.cisco.com/en/US/products/sw/secursw/ps2308/index.html>, March 2007.
- [11] 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.
- [12] R. Droms. *RFC 2131 Dynamic Host Configuration Protocol*, March 1997.
- [13] E. Exa. CloudVPN. <http://e-x-a.org/?view=cloudvpn>, September 2009.
- [14] R. Figueiredo and et al. Archer: A community distributed computing infrastructure for computer architecture research and education. In *CollaborateCom*, November 2008.
- [15] 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.
- [16] A. Ganguly and et al. Improving peer connectivity in wide-area overlays of virtual workstations. In *HPDC*, 6 2008.
- [17] 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.
- [18] W. Ginolas. P2PVPN. <http://p2pvpn.org>, August 2009.
- [19] 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.
- [20] LogMeIn. Hamachi. <https://secure.logmein.com/products/hamachi/vpn.asp>, July 2008.
- [21] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: distributed hashing in a small world. In *USITS*, 2003.
- [22] K. Petric. Wippen. <http://wippen.com/>, August 2009.
- [23] G. Sliepen. tinc. <http://www.tinc-vpn.org/>, September 2009.
- [24] I. Stoica and et al. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *SIGCOMM*, 2001.
- [25] 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.
- [26] 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.
- [27] J. Yonan. OpenVPN. <http://openvpn.net/>, March 2007.