On the Design and Implementation of Structured P2P VPNs

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 performance constraints 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. These approaches, currently lack the depth in security options provided by other VPN solutions. Additionally, the unstructured systems provide relays when NAT traversal is unsuccessful, but the scalability in current approaches has not been validated.

In order to deal with these issues in a method that would even be intuitive to non-expert users , 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 via a partially automated web interface. 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 this paper, we describe the components of our model and evaluate our reference implementation quantitatively to compare system and networking overheads of the different VPN technologies focusing on latency, bandwidth, and memory usage. We also discuss some of our experiences with developing, maintaining, and deploying such a system.

1 Introduction

A Virtual Private Network (VPN) provides the illusion of a Local Area Network (LAN) spanning over a public wide area network (WAN) infrastructure, guaranteeing secure and authenticated communication amongst participants. Common uses of VPNs include secure access to enterprise network resources from remote/insecure locations, connecting distributed resources from multiple sites, and establishing virtual LANs for multiplayer 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 that define VPNs "as the 'emulation of a private Wide Area Network (WAN) facility using IP facilities' (including the public Internet or private IP backbones)." [18]. In these VPNs, large sets of machines are connected to a WAN-like VPN through one or more virtual routers.

A centralized VPN server can be a single point of failure, can create a performance bottlenecks, and can compromise end-to-end security. Several alternative approaches that address these issues have been conceived: 1) support for multiple VPN servers for a single VPN [35], 2) decentralized, managed VPNs that do not differentiate between client and server, 3) the use of P2P connections enabling direct communication amongst peers bypassing the server, relying on the server only for authentication and session management [21, 25], and 4) the use of unstructured P2P networks to form decentralized VPNs [17, 10, 29].

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 regard-

ing scalability with environments that include traversable and non-traversable NATs, lack of support for full tunnel VPN mode. While decentralized VPN experience a mix of the issues presented by centralized and P2P VPNs.

The problems we seek to address with our P2P VPN model include:

- reducing the role of centralization for user authentication in a VPN
- providing intuitive membership management in a 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 these 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 using an intuitive web based interface using user managed groups. For handling general trust concerns in P2P systems, we use the bootstrapping of a private P2P system dedicated only for VPN use. P2P VPNs introduce significantly more complexity when attempting to do full tunneling since a simple routing table swap as done in central VPNs will 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.

Overall, the main contribution made in this paper is the design and implementation of a novel structured P2P VPN 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
- easy to use interfaces for managing VPN members and securing VPN tunnels
- support for full-tunneling through multiple gateways
- decentralized NAT traversal with autonomous selection of relays for non-traversable nodes
- bootstrapping a private P2P VPN system off an existing public P2P system

The rest of this paper is organized as follows. Section II gives an overview of current VPN technologies with emphasis on decentralization and scalability. 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 P2P VPN. In Section V, we discuss our implementation and present evaluation comparing a VPN models. 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 [8] as well as OpenVPN [35]. In this section, we begin by going in depth on client configuration of VPNs, then we describe different VPN server configurations as highlighted in Table 1, finally concluding the section by presenting Table 2, which presents a qualitative overview of production VPNs.

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
Decentralized Servers and Clients	No distinction between client and servers, each member in the system authenticates directly with each other, links between members must be explicitly defined
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, use distributed data store for discovery

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 (VN) device. During initialization VPN software starts by au-

thenticating with an overlay or VPN agent, then, optionally, it queries the agent for information about the network such as the network address space, and finally the VN device is started.

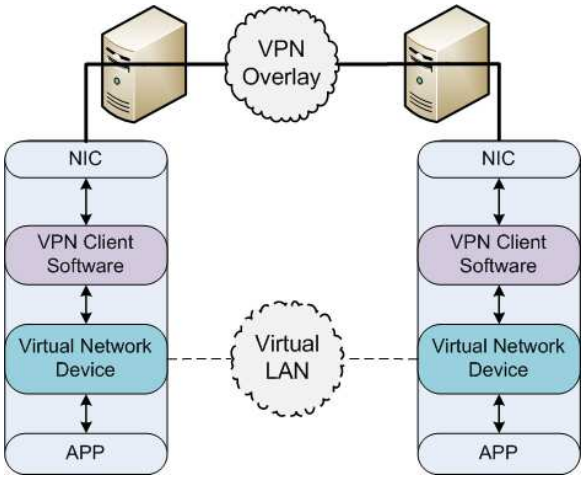

Figure 1: A typical VPN client. The VPN uses a VN device to make interaction over the VPN transparent. Packets that are destined for VPN destinations are sent via routing rules to the VN device, which acts as like a file descriptor read and written to by the VPN client. The VPN client in turn sends and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

There are many different mechanisms for communicating with an overlay agent. For quick setup, a system may require no authentication or use a shared secret such as a key or password. Individual authentication can be provided through the use of accounts with passwords in addition to a shared secret. This allows blocking out users if the shared secret becomes compromised or users act maliciously. For the strongest level of security, each client can be configured to have a signed-certificate that makes brute force attacks very difficult. The trade-offs come in terms of usability. While the use of uniquely signed-certificates provides better security than shared secrets, it can be more difficult to set up and use, in particular for a target user base of non-expert. A good balance found in many environments is the mixture of a shared secret and individual user accounts, where the shared secret is included with the installation of the VPN application where the application is distributed from a secured site.

Once the VPN has connected with the overlay, the VN 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 and an allocation of an address for the user's machine.

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 VN 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 block device providing read and write operations. Incoming packets from the VN are written to the TAP device and the networking stack in the OS delivers the packet to the appropriate socket. Outgoing packets from local sockets are read from the TAP device.

The VN device can be configured manually through static addressing or dynamically through dynamic host configuration process (DHCP) [11, 1]. Once the address to the device is set, a new routing will be added to the routing table causing all packets sent to the VPN address space to be sent to the VN device. After a packet is read from the TAP device, it is encrypted and sent to the overlay via the VPN client. The overlay delivers the packet to another client, which either is a client or a server enabled with virtual networking stack. The received packet is decrypted, verified for authenticity, and then written to the TAP device. In most cases, the IP layer header remains unchanged while VPN configuration determines how the Ethernet header is handled.

The described configuration so far, creates what is known as a split tunnel, or a VPN connection that only has passes *VPN related traffic only 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. Both are visually expressed in Figure 2.

Most centralized VPNs implement full tunneling through a routing rule swap, which makes the default gateway an endpoint in the VPNs subnet and traffic for the VPN server is routed to the LAN gateway. As an example, in a typical home network, an Internet bound packet will be retrieved at the VN device, encrypted, and sent to the VPN gateway via the LAN's gateway. At the VPN gateway, the packet is decrypted and delivered to the Internet. All other traffic is sent to the VN 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. We further

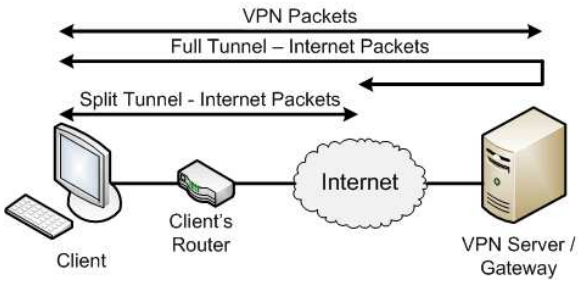

Figure 2: A VPN setup expressing both full and split tunnel modes. In both modes, packets for the server are sent directly to the server. In split tunnel mode, Internet packets are routed directly to the Internet. In full tunnel mode, Internet packets are first routed to the server / gateway and then to their Internet destination.

discuss this issue 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 most centralized VPN. The key aspects of a centralized VPN server are:

- authentication of clients
- routing packets between clients
- providing a NAT to the servers local resources and Internet (full tunnel)
- inter-server communication

Central VPNs server operate at well-known endpoints as in a universal resource identifier (URI) consisting of a hostname or IP address and a port. To log in, clients will randomly attempt to connect to one of the servers until successful, implementing a simple load balance. Once connected, clients obtain an address in the VPN 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 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.

When two clients communicate over an OpenVPN server, the process involves the following steps:

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. after positive authentication, the packet is written to the VN 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 allow the client to access the server's resources, the server too must behave somewhat like a client by enabling a VPN endpoint 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, 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.

Two examples of systems that assist in distributing load in VPN systems are tinc [29] and CloudVPN [12]. Unlike the decentralized P2P systems, these decentralized systems lack the ability to automatically form and maintain VPN systems. In other words, they lack self-organization. This means that like OpenVPN, these systems can suffer VPN outages when nodes go offline. The difference being that OpenVPN makes it explicit who is a server and who is not, whereas in tinc and CloudVPN anyone can be a server or a client. In the typical Tinc and CloudVPN setup, individual users share endpoints with each other out of band and then place them in the VPN configuration file. Due to the lack of self-configuration, members in the system will not replace links as members go offline. Thus the mesh, ring, and other structures

mentioned in tinc and CloudVPN documentation require direct human interaction for setup and management.

2.3 Centralized P2P VPN Systems

Hamachi [21] began the advent of centralized VPNs that went with the ambiguous moniker "P2P VPN". In reality, these systems would be best classified as centralized VPNs servers with P2P clients. Specifically, the nature of P2P in these [25, 20] 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 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 involve using a server to verify the authenticity and setup secure connections between clients. 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. Furthermore, these systems do not support full tunneling nor accessing resources on the same network as the other client in any of these cases.

2.4 P2P VPN 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 dedicated instances of the P2P VPN. 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. Two popular examples of unstructured P2P VPNs are N2N [10] and P2PVPN[1] [17]. 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 look up 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.

Typically, decentralized, P2P VPNs begin by attempting to connect to well-known endpoints 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 [10] requires knowledge of an 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.

[1] Due to the similarities between the name P2PVPN and focus of this paper, P2P VPNs, all occurrences of the string "P2PVPN" refer only to [17] and all occurrences of P2P VPN refer explicitly to our research.

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 must either know all the state in the system or make random walks [4]. Some examples of structured systems can be found in [27, 30, 22, 23, 26]. 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 endpoints 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 attempt to connect with the same set of peers. The peers will only connect to one of them creating disconnectivity. Furthermore, having the node IDs well distributed will assist in providing better scalability as many algorithms for selection of shortcuts depend 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 for there to be a 50% chance of address collision. Another mechanism for distributing node IDs involves the use of a trusted third party to generate node IDs and cryptographically sign them [5].

Similar 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

	VPN Type	Authentication Method	Peer Discovery	NAT Traversal	Availability
OpenVPN [35]	Centralized	Certificates or passwords with a central server	Stored at central server(s)	Relay through server(s)	Open Source
tinc [29]	Decentralized	CA Certificates	Global knowledge	Relay through mesh	Open Source
CloudVPN [12]	Centralized	CA Certificates	Global knowledge	Relay through mesh	Open Source
Hamachi [21]	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 [20]	Centralized P2P	Password at central server	Stored locally	NAT traversal, centralized relay	Windows only, freeware, no private relays
Wippien [25]	Centralized P2P	Password at central server	Stored locally	NAT traversal, no relay support	Mixed Open / Closed source
N2N [10]	Unstructured P2P	Shared secret	Broadcast look up	NAT traversal, decentralized relay	Open Source
P2PVPN [17]	Unstructured P2P	Shared secret	Global knowledge	No NAT traversal, decentralized relay	Open Source
IPOP	Structured P2P	CA Certificates or pre-exchanged keys	DHT look up	NAT traversal and relay through physically close peers	Open Source

Table 2: VPN Comparison

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 [27]. This can work well in certain environments but multicast range can be quite limited.

Depending on the protocol, a node must be connected to either the closest neighbor smaller, larger, or both. Optimizations for fault tolerance suggest that it should be between 2 to $\log(N)$ on both sides. If a peer does not know of the address of its immediate predecessor or successor and a message is routed to it destined for them, depending on the message type, it will assume the packet was meant for itm or throw the packet away. Thus having multiple peers on both sides assist stability when the system is experiencing churn, that is when peers leave without warning.

Shortcuts make quickly traversing a P2P structure possible. 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 [27, 23], maintaining a connection with a peer every set distance from you in the P2P address space [30], or using a handful of well calculated locations in the node ID space [22].

4 Components of a P2P VPN

Before presenting our contributions made in this paper, we first review our current work as it provides the basis for our P2P VPN. At the heart of our system lies a P2P system similar to Symphony [22], named Brunet [2]. The specific components of the system that make it interesting for use in a P2P VPN system include:

- NAT traversal [2]
- system stability when two nodes next to each other in address space cannot directly connect [15]
- selection of shortcuts using information based upon proximity [15]
- a distributed data store based upon a distributed hash table [16]

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

- self-configuring, low overhead use [34, 14]
- discovery of members through the use of a virtual DHCP server using DHT [34, 16]
- scalability in the count of hundreds of peers in a single VN [34]

- portability to any system that supports Tap and Mono[2] [9]
- ability to behave as a VN interface or router [34]
- used in grid computing for over 3 years [13, 32, 31, 33]

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

Setting up, deploying, and then maintaining security can easily become a non-negligible task. Most VPN systems that support the use of certificates require the use of command-line utilities, setting up your own methods of securely deploying certificates and policing users. In this section, we present partially automated certificate management through groups use of a redistributable web interface. The key to the group infrastructure is that users should have a mechanism to verify the authenticity of the server, properly authenticate themselves with the server, and thereafter acquire signed certificates as needed. From an administrators point of view, user information should be easily available for verification and the ability to ban and remove users from the pool should be as easy as click a mouse button.

Client usage in our model uses the following process:

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 the user's node ID and secret key, the server verifies the authenticity of the user through the secret and ...
   (a) automatically signs the certificate
   (b) or waits for an administrator to verify the certificate request
9. upon receiving a signed certificate, 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. A group acts as a trusted set of peers where the administrator acts as the certificate authority signing certificates as he authenticates members into the system. Signed certificates can be used in decentralized systems as they do not require communication with the certificate authority, 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. An insecure solution for configuration generation would be to have the server generate a private key and certificate for a peer. The use of a binary blob generated by the server allows the user to keep the user's private key private, but he can also use the blob to generate multiple VPN clients without direct user interaction from the website. Securing this system becomes very important as the blob can easily become a weak spot in the system. While it may be cryptographically stronger than a key and password, it must still be exchanged over secure mediums, such as HTTPS, which can be done with no user intervention.

Administrators need mechanisms for dealing with malicious users. In systems that only use shared secrets, the systems creator has no ability without collusion to remove malicious participants from the VPN. Our group model, like other certificate based schemes, provides the ability to administrate a system and police such users through the following methods:

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

A CRL offers an out of band mechanism for distributing user revocations, unlike information exchanged in a P2P overlay, primarily because malicious users can use common P2P attacks to prevent notification of certificate revocations transmitted via the overlay, but it is significantly more difficult to prevent the retrieval of a 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 expen-

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

sive for peers to verify certificates on regular intervals with short periods.

The DHT approach allows peers to regularly verify the remote peer prior to and during connections. Additionally, 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's certificate has been 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, though this is feasible in our model if groups use automated certificate signing. 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 divided into two categories: split tunneling and full tunneling. In split tunnel mode, the the VPN client receives and sends messages related only to the VPN, i.e., both end points are in the VPN; whereas in full tunnel mode, all traffic both VPN and Internet traffic with the exception of VPN "control" messages route over the VPN, as shown in Figure 2. In a centralized VPN these "control" messages would be the communications directly with the VPN server and full tunnel gateway, which most likely is the same machine destination IP address from the perspective of the client. In a P2P VPN, "control" messages would be communications directly between P2P peers. Using full-tunneling ensures that a malicious user cannot easily eavesdrop into what would otherwise be public communication by forwarding all non-VPN related traffic securely to a third party who resides in a more trusted environment. There are two key components to this scheme, a gateway / server or traffic relayer and a client or a traffic forwarder. In the following sections, we present a simple scheme for providing gateways and a few solutions to configuring clients in a structured P2P overlay.

4.2.1 The Gateway

Configuring a machine as a gateway can easily be done in Linux using an iptables component called masquerade, which automatically handles forwarding packets received on one interface and forwarding them to the next hop automatically taking care of NAT as well as receiving packets from the Internet and forwarding them back to the appropriate client. The specific features nice about the NAT handling is that it handles IP and Ethernet manipulation in addition to some packets that contain IP information, such as FTP and SIP. 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

Please note, that there are many ways to implement NAT in Linux, our goal was to present an easily reproducible model, which was well served through this setup.

With a gateway intact, the VPN software can now announce that it provides full tunneling. For that purpose, we added an enable flag into the VPN configuration to specify that a machine is a full tunnel gateway. When the VPN software starts, it will automatically append itself to the list of known gateways for the VPN group in the DHT.

The only remaining difference in VPN gateway is the state machine used in processing packets coming from the VPN. Previously, we rejected packets if the destination was not part of our network, now we check first to see if we have enabled gateway mode. If it is enabled, all packets are written to the TAP device with the destination Ethernet address being that of the TAP device. The remaining configuration is identical to other members of the system.

4.2.2 The Client

VPN Clients wishing to use full tunnel must redirect their default traffic to their VN device. Using the above example, the routing rule would be default traffic (0.0.0.0/0 – IP/netmask) is directed to 5.0.0.1 (the gateway). There does not necessarily have to be an active node online at 5.0.0.1, as shown in Figure 3 the only sign of that this packet is destined for the gateway appears at the Ethernet header as the destination address. This is relevant to two of our previous works as described in [34]: 1) each VPN software stack uses a single Ethernet address to represent all remote hosts which differ only by their IP address, this approach is called a transparent subnet gateway [3] and 2) the lowest address in a network is reserved for VN

services such as DHCP and DNS. Thus 5.0.0.1 becomes a "virtual" virtual address, and packets sent for gateway purposes to this address can be forwarded to any machine in the VPN for relaying to the Internet without any IP or higher layer changes. With this in mind, the client software need only pay heed to the destination IP address, which points to the Internet. The VPNs state machine has to be slightly modified to handle outgoing packets not destined for a remote VPN end point:

1. is the packet destined for some area outside of the VPN address space
2. if so, verify that full tunnel VPN is enabled
3. if so, discover a remote peer to act as a full tunnel gateway
4. after finding one, send the message to the remote peer

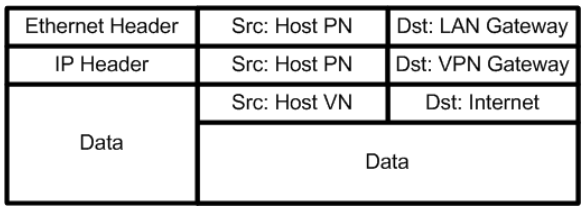

Figure 3: The contents of a full tunnel Ethernet packet. PN and VN are defined as physical and virtual network, respectively. Interesting components of this packet are 1) the destination Ethernet address is the LAN gateway, 2) the destination IP address is the VPN gateway, and 3) the IP payload contains another IP packet whose source IP is the VN device and destination is the Internet.

The pathway for packets coming back from the Internet will also require some tweaking, that is, the source IP address will not match the VPN gateways IP. Thus the VPN client must confirm that the source is a VPN gateway or throw away the packet. If the source is authenticated, it can be written to the VN device and the user application will receive a packet from the Internet.

The two issues in the client configuration are selecting the machine to use as a gateway and how to ensure P2P packets are sent directly to the remote peer and not through the gateway. Our model uses a simple mechanism for determining which gateway to choose: query the DHT for a list of potential gateways, select a random one from the list, and verify liveness via periodic (15 second interval) ping messages. For handling faults, we take a pessimistic approach, that is, if a server is lost, we take note of it and only query the DHT again upon the next outgoing Internet packet.

For this paper, the second issue, handling P2P routed packets, was the focus of our work. In the centralized VPN case, a client only needs to communicate directly with a single point in the VPN. This can be handled by having a single rule added to the routing table that would route all packets to this end point via the LAN's default gateway. The rule would look similar to 128.227.56.121/32 directed to 192.168.1.1, where 128.227.56.121 is the remote machine and 192.168.1.1 is the local gateway.

In a P2P system, ensuring that all P2P overlay messages are routed directly becomes non-trivial 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 gateway 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 present many different possibilities but in our efforts were only successful with one 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 uses a model similar to the centralized version, that is, for each P2P machine we communicate with, we add an explicit rule in the routing table so that packets destined for them are routed directly to them via the LAN's default gateway. 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 prior to the first outgoing communication attempt and would remain in effect until the VPN closed down or the link was no longer active. This model does not need to worry about incoming packets, as they will be unaffected by the routing table and will be delivered as they were in the normal system.

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. An issue, unique to our P2P solution, allows a malicious user to send spoofed packets to have the VPN add extra routes to the routing table, resulting in a similar situation as the first, unencrypted communication visible to eavesdroppers. 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 using a technique similar to the one used in the first solution. 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 VN 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 of outgoing IP packets, 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, a packet capture and injection library, to send packets. Additionally, Linux also supports a packet type called packet, 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 VN Ethernet's, the one where it originated. This would trigger the OS either ignoring the packet in the case of UDP or sending a TCP reset message. Thus the only solution would be to implement some form of NAT, that would receive incoming packets and rewrite them prior to the network stacking thus avoiding a TCP reset message, 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, as all VPN traffic will traverse the TAP device twice, once as VPN traffic and the second as P2P traffic, so we also investigated the potential of using existing Linux tools to solve this problem. Our focus was on the use of iptables as it 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.

Iptables alone was not able to do the job as it 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. 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 VN device and not of the physical Ethernet device.

4.3 Autonomic Relays

A handful of P2P VPNs [21, 20] support relaying when direct communication is not possible though a majority of them do not. Centralized and decentralized VPNs do not suffer from this problem as all traffic passes through the central server or managed links. Because of the management and overhead concerns presented by these systems, we propose the use of distributed, autonomic relaying system based upon previous work known as "tunnels" [15]. In this work, we describe a mechanism using triangle routing that allows peers next to each other in the node ID space communicate despite them being unable 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 through the overlay. This would trigger a best effort to exchange peer lists through the current set of near neighbors. In most cases, the peers would have at least one overlapping neighbor and the messages would be exchanged. Thereafter, peers would use the overlapping neighbors to communicate with each other "directly".

Direct connectivity in most cases provides significant performance gains over routing through the overlay. When direct connections is not available, which is more efficient relaying or overlay routing? In practical experience, we have seen overlay transit time in the order of seconds and further establish the use of relays in our evaluation in Section 5.1. When peers are not near each other in address space, there is a high chance that they will not have any overlapping peers, thus our current technique would not work. Our solution, as represented in Figure 4, 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.

The original approach had one more issue, the lack of consideration on the viability of relays, the goal was to have connectivity with no reflection on stability or performance. To improve this, when exchanging neighbor

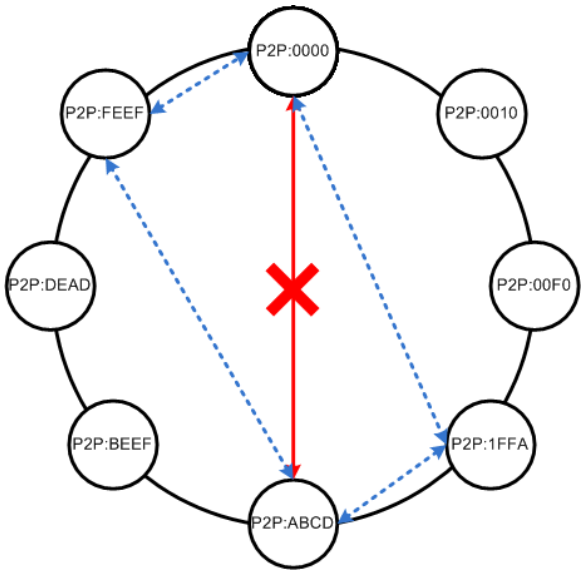

Figure 4: Creating relays from across the node address space, when direct connectivity is not possible. Two members, 0000 and ABCD, desire a direct connection but are unable to directly connect, perhaps due to NATs or firewalls. They exchange neighbor information through the overlay and connect to one of each other's neighbors, creating overlap. The overlap then becomes a a dedicated relay path (represented by dashed lines), improving performance over relaying across the entire overlay.

lists, we added the ability to exchange arbitrary information to assist in decision making. So far we have focused on exchanging connection stability as measured by the age of the connection between the far node and his neighbor and the latency between the far node and its neighbor. Additionally, when overlap changes, we make it optional to select to use only a subset of the overlap, thus only the fastest or most stable overlap be used with many more in reserve. We present our test environment, experiment, and results. in 5.1.

4.4 Bootstrapping Private Overlays

In P2P systems, distributed security cannot provide the same level of security as centralized or managed security. Hopefully securing an overlay using well-known security concepts such as certificates and SSL will encourage wider adoption of P2P systems. One problem with this approach though is that users who want a private overlay may not have the resources, i.e. public addresses, to host there own overlays. To address this, we suggest bootstrapping a private overlay from an existing public overlay. The private overlay will be completely encrypted and authenticated with only members of the VPN allowed access. This work is similar to previous in suggesting the use of a public overlay to bootstrap application specific overlay has been discussed in [6].

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 become 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 [7] are necessary to provide multicast 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 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 [24] 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. At a minimum, this would be help us determine if there exists a problem. To assist in finding bugs, we have added liveness messages to threads to assist in find-

ing deadlocks. Additionally, Mono can print out trace logs by sending a "USR2" signal to the running process.

Other information covered we watch include peer count, memory, and CPU usage. Node count can be quite difficult to keep track of in Planet-Lab as machines at a rate of 5 to 20 per day are restarted and our software is not automatically restarted on these machines, thus the case to watch for is non-linear loss of nodes. 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 Planet-Lab 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 be very 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 P2P VPN in Other Structured Overlays

The purpose of this work is to develop a P2P VPN model that can easily be applied to other structured P2P systems. In this section, we focus on the portability of our platform to other structured P2P systems, namely Pastry and Chord by analyzing FreePastry and NChord respectively. FreePastry can easily reuse our C# implemented library through the use of IKVM.NET, 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.

The bootstrapping of a connection in NChord, begins by finding the owner of the DHT key containing the mapping of virtual IP address to node address through "find_successor". After establishing a connection, the owner of the key needs to be queried for the key's value, which would be the node ID of the node owning the virtual IP. After executing "find_successor" and retrieving the destination nodes, the owner of the virtual IPs, physical IP address, the VPN can "connect" to the remote node using either a UDP or TCP socket. Virtual IP messages would then be sent and received through this "connection". Unlike other systems, NChord does not support sending messages through the overlay, thus a separate "connection" for application purposes will need to be created.

The steps involved in FreePastry begin by looking up the mapping of virtual IP to node ID in PAST. The result will be a node ID, which can be called using "route" 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. FreePastry does not have the ability to form shortcuts based upon demands, so if two peers were actively communicating, they may always have to route traffic over the overlay, which will probably be significantly slower than if they were directly connected. Though one could argue, that since FreePastry does not support NAT traversal, all nodes will already be public and thus an application, as in the NChord example, could form a direct connection bypassing the overlay.

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 [34] and Brunet [2]. We begin by examining the effects of different relay selection mechanisms. Afterwords, 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] S. Carl-Mitchell and J. S. Quarterman. RFC 1027 - using arp to implement transparent subnet gateways, October 1987.

[4] 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.

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 [28], reliable	No	No
NChord	No, only look up	No	Simple, non-fault tolerant DHT	No	No

Table 3: A comparison of structured P2P systems. PtP stands for point-to-point 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.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, 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.

[6] 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.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE Journal on Selected Areas in Communication (JSAC)*, volume 20, October 2002.

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

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

[10] 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.

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

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

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

[14] 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.

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

[16] 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.

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

[18] 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.

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

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

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

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

[24] L. Peterson and D. Culler. Planet-lab. `http://www.planet-lab.org`, March 2007.

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

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

[27] 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.

[28] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles (SOSP'01)*, pages 188–201, Oct. 2001.

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

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

[31] 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.

[32] 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.

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

[34] D. I. Wolinsky, Y. Liu, P. S. Juste, G. Venkatasubramanian, and R. Figueiredo. On the design of scalable, self-configuring virtual networks. In *IEEE/ACM Supercomputing 2009*.

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