On the Design and Implementation of Autonomic, Decentralized GroupVPNs

***Abstract*—Virtual private networks (VPNs) enable existing network applications to run unmodified in insecure and constrained environments by creating an isolated and secure virtual environment providing all-to-all connectivity for VPN members. Traditionally VPNs have been used in dedicated systems to connect many users to a single site or a few sites together through manual configuration. Recently, decentralized VPNs have been developed to support those that require systems with no dedicated resources and reduced configuration overhead. We extend this concept to provide a truly novel approach to VPNs, a decentralized, self-configuring VPN. To put our approach in prospective, we survey existing VPN approaches and quantitatively compare a reference implementation of our approach to existing VPN systems.**

I. Introduction

A Virtual Private Network (VPN) provides the illusion of a local area network (LAN) spanning a wide area network (WAN) infrastructure by creating encrypted and authenticated, secure[1] communication links 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. VPNs, in the context of this paper, differ from others that provide "'emulation of a private Wide Area Network (WAN) facility using IP facilities' (including the public Internet or private IP backbones). " [1]. These style of VPNs connect large sets of machines through virtual routers to a virtual WAN environment.

The architecture described in this paper addresses a usage scenario such as collaborative academic environments linking individuals spanning multiple institutions belong to the same virtual organization, where coordinated configuration of network infrastructure across the different sites is often impractical. Another example is the small/medium business (SMB) environment, where it is often desirable to interconnect desktops and servers across distributed sites and secure traffic to enterprise networked resources without incurring the complexity or management costs of traditional VPNs. Alternatively, consider an academic at a conference who wants their Internet traffic encrypted to prevent eavesdropping from their peers or desires to browse websites only available to those on the university network.

In this paper, we discuss an architecture that fits the following requirements: 1) direct communication between remote members, 2) NAT and firewall traversal or handling, 3) security based on a public key infrastructure (PKI), 4) decentralized discovery of members in the system, 5) ability to route Internet traffic through a secure location (full tunnel VPN), 6) self-configuring links amongst peers, and 7) user interfaces for managing the VPN.

Centralized approaches (e.g. OpenVPN [2]) by their very nature require dedicated infrastructures and do not allow direct communication between peers though are the only VPN approach to full tunnel operation and guarantee all-to-all communication regardless NAT and firewall conditions. P2P-based approaches (e.g. Hamachi [3], Wippien [4], Gbridge [5], PVC [6]) are vulnerable to man-in-the-middle attacks if session management is handled by an external provider, rely on a central resource for the creation of VPN links, and require centralized relays if NAT and firewall traversal fails. While other approachs require manual configuration of links between members of the virtual network (e.g., ViNe [7], Violin [8], VNET [9], tinc [10]). Decentralized approaches (N2N [11] and P2PVPN [12]) lack the ability to scale to large networks and only support pre-shared key security and thus cannot remove malicious peers from the VPN.

To address these issues and meet our requirements, we propose the use of structured peer-to-peer (P2P) overlay as the fundamental building block to create a decentralized, dynamic VPN. We have built a virtual, though not private, network in previous work [13], [14]. In this paper, we consider the security implications of using a structured overlay and present methods that create a VPN on top of a structured overlay. The foundation of our VPN is an extension of our previous work in [15], where we implemented PKI secured overlays configured through a Web 2.0 Group infrastructure, to support the configuration of GroupVPNs. The other components missing from previous and related research and work is the lack of decentralized, autonomic creation of relays and support for full tunnel VPN mode in a decentralized system, our novel contributions for this paper. This paper stands alone as a canonical resource to provide necessary documentation to create a structured P2P VPN supporting all the requirements listed earlier.

The rest of this paper is organized as follows. Section II discusses the different VPN approaches including ours which is based upon structured P2P overlays. We present our novel contributions of decentralized relays, full tunnel VPN modes for P2P VPNs, and user-friendly configuration of VPNs in Section III, Section IV, and Section V, respectively. To validate the P2P approach, we compare it against centralized VPNs in Section VI. Finally, we conclude our paper in Section VII with a discussion on real systems using our approach.

[1]For the remainder of this proposal, unless explicitly stated otherwise, security implies encryption and authentication.

II. Virtual Private Networks

There exist many VPN architectures, in Table I, we present some of the more common abstractions. In this section, we begin by reviewing the fundamental features found in all VPN clients and then we describe the components that are attributed to the different VPN approaches, such as VPN client address allocation, peer discovery and authentication, and the available security models using real VPN systems.

TABLE I
VPN Classifications

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 traffic; peers may communicate directly with each other via P2P links if NAT traversal succeeds
Decentralized Servers and Clients	No distinction between clients 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

A. The Basic Client VPN Configuration

In Figure 1, we abstract the common features of all VPNs clients, a service that communicates with the VPN system and a virtual network (VN) device for host integration. During initialization, the VPN services authenticates with the overlay [2], optionally, querying for information about the network, such as network address space, address allocations, and domain name service (DNS) servers. At which point, the VPN enables secure communication amongst participants.

Clients can authenticate with the overlay using a variety of methods. A system can be setup quickly by using no authentication or a shared secret such as a key or a password. Using accounts and passwords with or without a shared secret provides individualized authentication, allowing an administrator to block all users if the shared secret is compromised or individual users who act maliciously. In the most secure approaches, each client has a unique signed certificate making brute force attacks very difficults. The trade-offs in the approaches come in terms of security, usability, and management. While the use of signed certificates provides better security than shared secrets, certificates require more configuration and maintenance. In a system comprising of non-experts, the usual setup uses a shared secret and individual user accounts. The secret is packaged with the VPN application, which is distributed through secure channels such as authenticated HTTPS.

A VN device allows applications to communicate transparently over the VPN. The VN device provides mechanisms for injecting incoming packets into and retrieving outgoing packets from the networking stack, enabling the use of common network APIs such as Berkeley Sockets, thereby allowing existing application to work over the VPN without modification. While there are many different types of VN devices, we focus on TAP [16] due to its open source and pervasive nature. 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 presents itself as a character device providing read and write operations. Incoming packets from the VPN 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.

VN devices can be configured manually though command-line tools or OS' APIs or dynamically by the universally supported dynamic host configuration process (DHCP) [17]. Upon the VN device obtaining an IP address, the system adds a new rule to the routing table that directs all packets sent to the VPN address space to be directed to the VN device. Packets read from the the TAP device are encrypted and sent to the overlay via the VPN client. The overlay delivers the packet to another client or a server with a VN stack enabled. Received packets are decrypted, verified for authenticity, and then written to the VN device. In most cases, the IP layer header remains unchanged, while VPN configuration determines how the Ethernet header is handled.

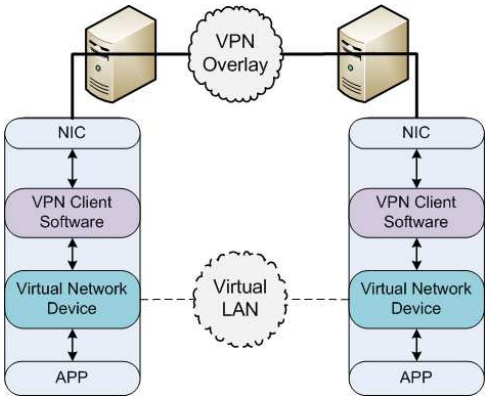

Fig. 1. A typical VPN client. A VN device makes application interaction with the VPN transparent. Packets going to the VPN destination are sent by routing rules to the VN device interfaced by the VPN client. The VPN client sends and receives packets from other VPN participants via the hosts physical network device.

B. Centralized VPN Systems

OpenVPN is an open and well-documented platform for deploying centralized VPNs. We use it as the basis for understanding centralized VPNs as it represents features common to most centralized VPNs. In centralized VPN systems, clients forward all VPN related packets to the server. Clients responsibilities are limited to configuring the VN device and authenticating with the VPN server; whereas the servers are responsible for authentication and routing between clients and

[2]An overlay in this context refers any portion of the VPN system including a central server, another VPN client, or a relay.

providing access to the servers' local resources and the Internet (full tunnel).

While central VPNs do support multiple servers, the client must know of all existing servers randomly selects a server, implementing a simple load balance. Once connected, servers hand to the client an IP 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. Additionally, servers must be configured to connect with all other servers otherwise clients behind remote servers will not be able to communicate with each other.

All inter-client communication flows through the central server. By default, a client encrypts a packet and sends it to the server. Upon receiving the packet, the server decrypts it, determines where to relay it, encrypts it, and then sends the packet to its destination. This model allows a server to eavesdrop on communication. While a second layer of encryption is possible through a shared secret, it requires out-of-band communication and increases the computing overhead on communication.

C. Centralized P2P VPN Systems

Hamachi [3] is the first well known centralized VPN that used the ambiguous moniker "P2P VPN". In reality, these systems would be best classified as centralized VPN servers with P2P clients. Specifically, the nature of P2P in this, Wippien [4], and Gbridge [5] is direct connectivity between clients once authenticated by a central server. In the case where direct connectivity is unavailable due to NATs or firewalls, the central server can act as a relay and if not peers will be unable to communicate. Each approach uses their own security protocols that involve using a server to verify the authenticity and setup secure connections between clients. Furthermore, most of these projects are closed preventing users from hosting their own authentication servers and relays and verifying the code for security purposes forcing users to trust the third-party server to not eavesdrop or perform other man-in-the-middle attacks.

D. Decentralized VPN Systems

Some examples of systems that assist in distributing load in VPN systems are tinc [10], CloudVPN [18], ViNe [7], VNET [9], and Violin [8]. These systems are not autonomic and require explicit specification of links between resources. This means that, like OpenVPN, these systems can suffer VPN outages when nodes go offline, thus administrators must maintain the VPN connection table. Unlike OpenVPN, these approaches typically do not require all-to-all direct connectivity for all-to-all communication. Users can either setup out-of-band NAT traversal or route through relays. Users must manually create links with those whom they desire maximum network performance.

E. Unstructured P2P VPN Systems

Unlike centralized and decentralized systems, P2P environments are self-configuring requiring the only task for the user is to connect to the P2P overlay. The most simplest form of overlays are unstructured, where peers form random connections with each other and use broadcast and stochastic techniques to find information and other peers, though due to its unstructured nature, the system cannot guarantee distance and routability between peers. Two examples of unstructured P2P VPNs are N2N [11] and P2PVPN[3] [12]. P2PVPN is somewhat of a hybrid using a centralized BitTorrent tracker to assist in peers finding each other, they eventually will also be able to search for each other using the overlay. In N2N, peers first connect to super nodes and then to find another peer they broadcast discovery messages to the entire overlay. In the case that peers cannot form direct connection, peers can route to each other over the N2N overlay. While unstructured P2P systems have some scalability concerns, they provide a server-less approach. In the realm of VPNs, all client VPNs are also servers performing authentication though neither approach deals with decentralized address allocation.

F. Structured P2P VPN Systems

To address the scalability concerns in unstructured systems, our approach uses structured P2P overlays. Structured P2P overlays provide distributed look up services with guaranteed search time with a lower bound of $O(\log N)$, in contrast to unstructured systems, which rely on global knowledge/broadcasts, or stochastic techniques such as random walks [19]. Some examples of structured systems can be found are Pastry [20], Chord [21], Symphony [22], Kademlia [23], CAN [24], and Brunet [25]. In general, structured systems are able to make these guarantees by self-organizing a structured topology, such as a 2D ring or a hypercube, deterministically by randomly generated node identifiers.

Most structured P2P overlays support decentralized storage/retrieval of information by mapping keys to specific node IDs in an overlay. At a minimum, the data is stored at the node ID either smaller or larger to the data's node ID and for fault tolerance the data can be stored at other nodes. This sort of mapping and data storage is called a distributed hash table (DHT).

In [13], we describe the construction of a virtual, though not private, network using structured P2P overlays named IPOP. Because this is the basis of our work in this paper, we first describe the construction of this structured P2P VN and then we describe our first novel contribution, privacy for the structured P2P VN. Each VN has a unique identifier or namespace so that the VPNs can share a common overlay. To perform address allocation, peers perform atomic writes into the structured overlays DHT, where the key is a hash of the VPN's namespace and the desired address and the value is the peer's node ID. A successful write implies that the user is the first to have written a value to that key and the user successfully has reserved an IP address. When a peer's VPN wants to communicate with an unknown VPN

[3]Due to the similarities between the name P2PVPN and focus of this paper, we use "P2PVPN" to refer only to [12] and "P2P VPN" to refer explicitly to the use of P2P in VPNs.

IP address, it queries the DHT using the hash of the VPN's namespace and the remote peer's IP address. This will be used to create a cached mapping, so that future packets can be directly routed to the remote peer. When two peers have a reasonable stream of communication, this triggers the creation of a direct connection between them.

The minimum requirement for communicating in a VPN is using secure links between communicating peers. In free-to-join structured overlays, this will not be enough as the DHT can easily become polluted and the overlay may become unreliable due to malicious peers [26]. While there are many works that describe tamper resistant overlays, they cannot make the same security guarantees made by centralized systems. In that regard, we refer to security systems that protect the centralized system like PKI and shared secrets. If they were protected by those types of security systems, they would no longer be public, free-to-join systems. In [15], we presented a method by which a public overlay could be used to bootstrap a private overlay secured by PKI. Peers first join the public overlay, then using the public overlays DHT find other peers in the private overlay and connect with them. Connection messages are routed securely over the public overlay enabling the creation of an overlay constructed only of peers behind oppressive NATs and firewalls. In the private overlay, both IP and overlay control messages are secured using the PKI encrypted and authenticated using DTLS [27], thus if a peer misbehaves in the overlay, he can easily be revoked using both the centralized and decentralized approaches described in the prior work.

I3 [28] or Internet Indirection Infrastructure presents another approach to structured overlay based virtual networking. Peers use the DHT similarly to IPOP, though there is no discussion on address exclusivity. Also the purpose of I3 is not to provide direct communication between peers, but rather, indirect communication so that peers never discover each others real identities. The purpose of I3 does not align with the requirements we set forth in the introduction.

III. Dealing with Oppressive NATs and Firewalls

As of 2010, the majority of the Internet is connected via Internet Protocol (IP) version 4. A limitation in this protocol is that there are only 2^{32} addresses total though or approximately 4 billion addresses. With the Earth already having a population of over 8 billion and each individual having multiple devices that have Internet connectivity the IPv4 limitation is becoming more and more apparent. IPv6 provides an alternative, supporting 2^{128} addresses, it is not pervasive with sparse amounts of the Internet reachable via IPv6. In the meantime during the transition from IPv4 to IPv6, network address translation (NAT) enables many machines and devices to share a single IP address. The cost of this operation means that such machines lose direct addressability on the Internet.

Approaches for handling NAT traversal come from two main branches: traversal and relay. When performing NAT traversal, peers simulataneously attempt to create "holes" in their NAT devices allowing direct TCP or UDP communication with a peer behind another NAT. NATs work by making IP:Port mappings and so if two peers do this simulataneously, they may be able to have both of their NATs create these holes allowing direct communication. The issue with TCP NAT traversal is that it works poorly on systems that use stateful firewalls. Relaying always works as peers behind NATs communicate with a third peer who is not behind a NAT, though it has the obvious side effect of additional latency and the cost for the relay of supporting the bidirectional communication.

A. Relays

Centralized and decentralized VPNs do not suffer from this problem as all traffic passes through the central server or managed links. In the unstructured examples, P2PVPN does not support any form of relaying though N2N allows peers to communicate through the overlay when traversal fails. Simiarly a structured P2P VPN can use the overlay, though as an overlay grows in size, so does the hop count between peers increasing latency and potentially creating a bandwidth bottle neck. To address this issue, we propose the use of distributed, autonomic relaying system based upon previous work [29], wherein, peers next to each other in the overlays node ID space communicate "'directly" through overlapping connections due to firewall, NAT, or Internet fragmentation issues.

When two nodes discover each other via the overlay and attempt to become connected but fail during NAT and firewall traversal, they exchange peer lists overlap through the overlay. Upon receiving this list, the two peers identify the overlap in the neighbor sets to form a two-hop connection. In this work, we further extend this model to support cases when nodes do not have an overlap set. This involves having the peers connect to each other's neighbor set proactively creating overlap, as represented in Figure 2.

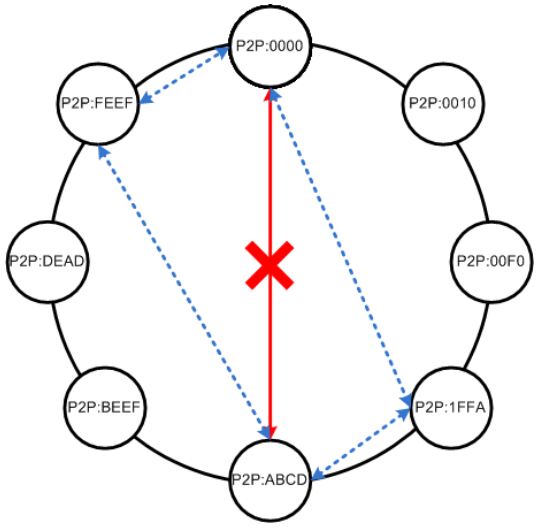

Fig. 2. Creating relays across the node address space, when direct connectivity is not possible. Two members, 0000 and ABCD, desire a direct connection but are unable to directly connect, perhaps due to NATs or firewalls. They exchange neighbor information through the overlay and connect to one of each other's neighbors, creating an overlap. The overlap then becomes a relay path (represented by dashed lines), improving performance over routing across the entire overlay.

To assist in the selection of overlap connection, we exchange arbitrary information along with the neighbor lists. So far we have implemented systems that share information about node stability (measured by the age of a connection) and proximity (based upon ping latency to neighbors). When overlap changes, we make it optional to select to use only a subset of the overlap, thus only the fastest or most stable peers are used with extras in reserve.

B. Usefulness of Relays

To verify the usefulness of two-hop over overlay routing, we have evaluted the latency approach using an event-driven simulator that reuses the code base of IPOP to faithfully implement its functionality using event-driven simulated times to emulate WAN latencies. For this experiment, we chose latencies based upon the MIT King Data Set [30], which consists of all-to-all latencies between 1,740 well-distributed Internet hosts. We then evaluated overlays using various network sizes up to 1,740. After starting the overlay, we wait for it to reach steady state, when the overlay is completely formed and no new connections are created, then we calculate the average all-to-all latency for all messages that would have taken two overlay hops or more, the average of our low latency relay model, and the average of single hop communication. In the low latency relay model, each destination node form a connection to the source node's physically closest peer as determined via latency (in a live system by application level ping). Then this pathway is used as a two-hop relay between source and node. We only look at two overlay hops and more, as a single hop would only benefit under triangular inequalities that are not a consideration in our work. The simulations were performed on a distributed grid platform, Archer [31], that uses IPOP for its virtual networking component.

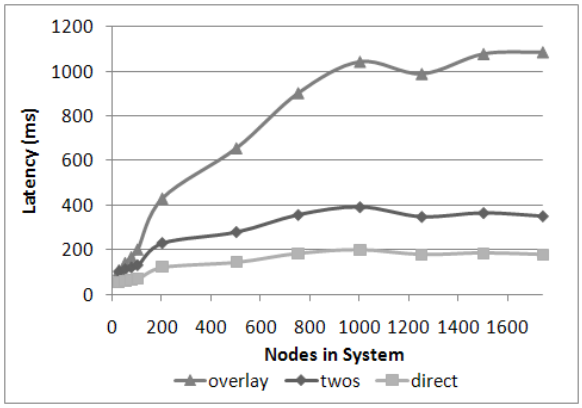

Fig. 3. A comparison of the average all-to-all overlay routing, two-hop relay, and direct connection latency in a Structured P2P environment, Brunet, using the King data set.

Our results are presented in Figure 3. We performed the tests for varying network sizes. We began our tests at 25, because network sizes around 20 and under tend to be fully connected due to the connectivity requirements of the system. It is not until the network size expands past 100 and towards 200 nodes that relays become significantly beneficial. At 100 nodes, there is approximately a 54% performance increase, whereas at 200 there is an 87% increase and it appears to grow proportionately to the size of the pool. The key take away is that latency-bound applications using a reasonably sized overlay would significantly benefit from the use of two-hop relays.

Additionally, we verified their use in a reference implementation of our VPN using a real system. The environment consists of using PlanetLab as the public overlay and our Archer environment as the VPN environment. PlanetLab [32] provides a set of over 500 distributed computing nodes all with public IP addresses. Archer provides grid computing to computer architecture researchers and consists of over 500 nodes located at various academic sites. To ensure that peers formed two-hop connections, we instituted a firewall preventing direct communication between the two. In tests, we found that the peers were always able to find a good relays, the bandwidth and latency averages were 2245 Kbit/s $\pm$ 1080 and 58.1 ms $\pm$ 35.5, respectively.

IV. Full Tunnel VPN Operations

The configuration detailed so far describes a split tunnel: a VPN connection that handles *internal VPN traffic only and not Internet traffic*. Prior to this work, only centralized VPNs currently support full tunnel: a VPN that provides the features of a split tunnel in addition to securely forwarding *all their Internet traffic* through a VPN gateway. A full tunnel provides network-layer privacy when a user is in a remote, insecure location such as an open wireless network at a coffee shop by securly relaying all Internet traffic through a trusted third party, the VPN gateway. Both models are illustrated in Figure 4.

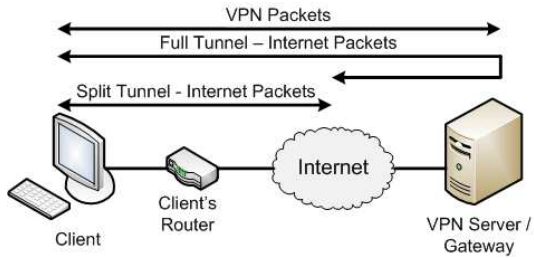

Fig. 4. An example of both full and split tunnel VPN modes. In both, packets for the server are sent directly to the server. In split tunnel mode, Internet packets bypass the VPN and are routed directly to the Internet. In full tunnel mode, Internet packets are first routed to the VPN gateway, and then to their Internet destination.

Central VPN clients use full tunneling through a routing rule swap, setting the default gateway to be an endpoint in the VPN subnet and traffic for thE VPN server is routed explicitly to the LAN gateway. This rule swap causes all Internet packets to be routed to the VN device and the VPN software can then send them to the remote VPN gateway. At the VPN gateway, the packet is decrypted and delivered to the Internet. A P2P system encounters two challenges in supporting full tunnels: 1) P2P traffic must not be routed to the VPN gateway and 2) there may be more than one VPN gateway. We address these issues and provide a solution to this problem in Section IV.

The challenges we face in our VPN are providing decentralized discovery of a VPN gateway and supporting full tunnel mode in a P2P environment such that all P2P traffic is sent to the intended receiver directly instead of through the gateway. The remainder of this section covers our gateway and client solutions to address these challenges.

A. The Gateway

A gateway can be configured through NAT software, like masquerading in IPtables or Internet Connection Sharing with Windows. This automatically handles the forwarding of packets received on the NAT interface to another interface bringing the packet closer to its destination. Similarly, incoming packets on the outgoing interface must be parsed in order to determine the destination NAT client.

Following from our previous work on VPNs [13], if a VPN is a gateway, the VPN state machine no longer rejects packets, when the destination is not in the VPN subnet, though when the VPN gateway mode is disabled these packets are still rejected. When enabled, all Internet and non-VPN based traffic is written to the TAP device setting the destination Ethernet address to the TAP device. The remaining configuration is identical to other members of the system as packets from the Internet will automatically have the clients IP as the destination as a product of the NAT. To provide for dynamic, self-configuring systems, VPN gateways announce their availability via an entry in the DHT. As future work, this approach can be explored to provide intelligent selection and load balancing of gateways.

B. The Client

VPN Clients wishing to use full tunnel must redirect their default traffic to their VN device. In our VPN model, we use a virtual IP address for the purpose of providing distributed VN services DHCP and DNS. This same address is used as the the default gateway's IP. Because this IP address never appears in a Internet bound packet, only its Ethernet addres does, as shown in Figure 5, this approach enables the use of any and multiple remote gateways.

To support full tunnel mode, the VPN's state machine has to be slightly modified to handle outgoing packets destined for IP addresses outside of the VPN, only rejecting them when full tunnel client mode is disabled. When enabled, the VPN software sends packets to the remote peer acting as a full tunnel gateway. Likewise, incoming packets that have a source address outside the subnet should not be rejected but instead the overlay address should be a certified VPN gateway prior to forwarding the packet.

To select a remote gateway, peers query the DHT. As there may be multiple gateways in the system, the peer randomly selects one, forwarding packets to that node. To ensure reliability, when the client has not heard from the gateway recently, the client sends a liveness query to the gateway. If the gateway is down, we take a pessimistic approach and find a new gateway when the next Internet packet arrives.

The real challenge in applying full tunnel VPN mode to P2P VPNs is the nature of the P2P system, namely dynamic connections. Peers do not know ahead of time what remote peer connections will be thus a simple rule switch does not work. Our original thought was to watch incoming connection requests and adding additional routing rules on demand, though this is only reasonably feasible with UDP as a TCP handshake message would need to be intercepted and potentially replayed by the local host in order to enable the rule and allow proper routing. The real drawback of the approach though is that UDP messages can easily be spoofed by remote peers enabling unsecured Internet packets to be leaked in the public environment. Even if the connections are secured, it could take some time for the peers to recognize a false connection attempt and delete the rule.

<table>
<tr><td>Ethernet Header</td><td>Src: Host PN</td><td>Dst: LAN Gateway</td></tr>
<tr><td>IP Header</td><td>Src: Host PN</td><td>Dst: VPN Gateway</td></tr>
<tr><td rowspan="2">Data</td><td>Src: Host VN</td><td>Dst: Internet</td></tr>
<tr><td colspan="2">Data</td></tr>
</table>

Fig. 5. The contents of a full tunnel Ethernet packet. PN and VN are defined as physical and virtual network, respectively.

To solve the security problem, we propose an alternative, have all traffic directly routed to the VN device with no additional routing rules. The VN is then responsible for filtering P2P traffic and forwarding it to the LAN's gateway via Ethernet packets. In the VPN application, outgoing IP packets' source ports are compared to VPN application's source ports. Upon a match, the VPN application directs the packet to the LAN's gateway. The three steps involved in this process are 1) translating the source IP address to match the physical Ethernet's IP address, 2) encapsulating the IP packet in an Ethernet packet with a randomly source address [13] and the destination the LAN's gateway, and 3) sending the packet via the physical Ethernet device. Sending an Ethernet packet is not trivial as Windows lacks support for this operation and most Unix systems require administrator privilege. Our platform independent solution uses a second TAP device bridged to the physical Ethernet device, allowing Ethernet packets to be sent indirectly through the Ethernet device via the TAP device. Because our solution results in incoming packets to arrive at a different IP address than the actual original source IP address TCP does not work in this solution. This method has been verified to work on both Linux and Windows using OS dependent TAP devices and bridge utilities.

C. Full Tunnel Overhead

While our full tunnel client method effectively resolves the lingering problem of ensuring that all packets in a full tunnel will be secure, it raises an issue: could the effect of having all packets traverse the VPN application be prohibitively expensive. To analyze this, we have implemented our approach and compared it to one that uses the routing rule switch. In Figure IV-C, we present the ping time from a residential location to one of Google's IP addresses using a gateway

located at the University of Florida when the VPN is in split tunnel mode, full tunnel using the routing rule switch, and full tunnel using Ethernet forwarding. The results express that there is negligible difference between the full tunnel approaches. One interesting result is the latency to gateways public address in the routing test, which most likely is a result of the ping being sent insecurely avoiding the VPN stack completely.

	Google	GW Pri	GW Pub
Ethernet	70.6	12.9	13.9
Routing	71.4	13.2	11.0
None	66.1	N/A	10.9

Fig. 6. Latency results comparing full tunnel approaches measured in ms. Legend: GW Pri - gateway's VPN address, GW Pub - gateway's VPN address, Ethernet - full tunnel Ethernet packet method, Routing - full tunnel routing rule switch, None - split tunnel or no VPN.

V. Enabling User-Friendly VPN Operation Through Groups

A Group based Web 2.0 environment enables collaborative environments in a user-friendly approach. The roles in a group environment are users who can join and create groups and administrators who can accept or deny join requests, remove users, and promote other users to administrators. To apply this to a VPN[4], we consider the group environment as a wrapper around a public key infrastructure (PKI), where the administrators of the group act as the certificate authority (CA) and the members have the ability to obtain signed certificates. Elaborating further, when a user joins a group, the administrator can enable automatic signing of certificates or require prior review; and when peers have overstayed their welcome, they can have their certificates revoked by removing them from the group. As described in [15], revocation can be stored as a CRL on the group site or distributed through broadcast and DHT on the overlay. Though we argue for the use of a user revocation list URL as opposed to a CRL, as a URL can easily handle cases where a malicious user has automatically signed large amounts of certificates.

During the creation of the group, the administrator configures the VPNs address range, namespace, enabling security, and if they would like to use an existing overlay network or provide their own set of overlay nodes. When a user has been accepted into the group, they are able to download VPN configuration data, that assists in the self-configuration of the VPN. The configuration is self-contained and requries that the user provide it to the VPN, by either placing it in a specific directory, a GUI, or a command-line utility whose only input parameter is the path to a configuration file. Though in our system, we have only implemented the file system and command-line approaches. The configuration data contains the IP address space, VPN namespace, group website's address, and a shared secret. The shared secret uniquely identifies the user, so that the website can determine how to handle certificate requests. When making a certificate request, the user sends over HTTPS a public key and their shared secret, the website creates and signs a certificate request based upon the public key and the user's relevant information ensuring that users cannot trick the website into signing malicious data. Upon receiving the signed certificate, peers are able to join the private overlay and VPN enabling secure communication amongst the VPN peers.

[4]Note: this is an extension of the group work we have done for overlays as presented in [15], the entire environment is not novel, but the application to VPN is

VI. Evaluation of VPN Models

In this section, we compare our reference implementation of a P2P VPN as described in the body of this paper to OpenVPN and Hamachi to understand the advantages and constraints of the various approaches. We evaluted the VPNs using Amazon EC2 [33] to dynamically create various sized networks ranging from 1 to 129 with one client used as the control. In the bootstrap phase, the control machine initiates communication with a subset of the remaining clients in the VPN. Once the system has warmed, the control continues pinging this subset every 15 seconds for the next 10 minutes. We capture bytes transmitted into and out of the system, as well as the memory size of the VPN application at the end of each stage. As we test the varying network sizes, we begin by communicating with 0 peers and exponentially increase the amount (powers of two) until the control is communicating with all members of the VPN. For these evaluations, we used a licensed version of Hamachi, but due to very recent changes in Hamachi, the Linux client will not support networks larger than 16. Due to amount of data collected and space limitations, we only present figures for results that provide interesting data and summarize the rest in the text.

Memory, for the most part, exhibited an intuitive behavior: for each additional connection there was more memory used. The OpenVPN control client showed negligible additional memory usage for additional nodes, though the server showed a linear increment, around 1 MB, for each additional client in the system, while activity had negligible effect on the results. Hamachi had a base line of a less than 1 MB and like OpenVPN, each additional client in the system had a linear effect on memory, on the order of 4 KB, there was no change based upon activity. The effect of additional inactive nodes in an IPOP network had negligible effect on memory, unlike Hamachi and OpenVPN. The only time IPOP's memory consumption increased was during activity and it scaled at a 200 KB per additional node.

In Figure 7, we present the bandwidth results from our experiment. Though we show results up to 128 clients, Hamachi is limited to network sizes of 16 when using Linux, so we used a linear regression ($.049 + .002KB/s * N + .02KB/s * A$, .049 the bandwidth in a network size of 0, i.e., keep-alives with the central server, N is the network size, and A is the active links. Hamachi clearly shows P2P efficiency over centralized VPNs. Our approach requires improvement for these environments though unlike Hamachi it is not an academic not a commercial

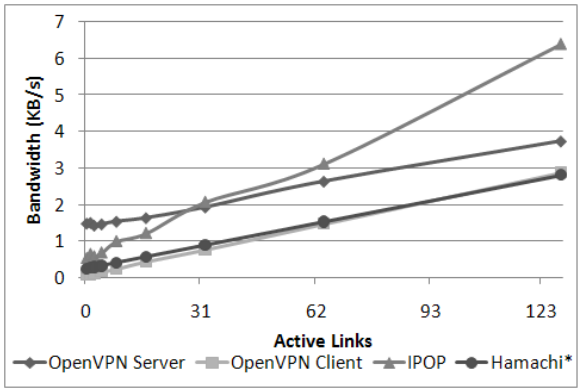

Fig. 7. Comparison of bandwidth costs for member activity versus network size. As stated in the text, Hamachi limits the system to network sizes of 16, so to estimate the Hamachi bandwidth we used a linear regression model.

project. Though one cost that IPOP does not share with the other approaches is the stateful keeping open of NAT holes.

VII. Conclusions

This paper presents a novel VPN approach designed to facilitate eace of use while maintaining security. We described how to construct a VPN using structured P2P overlays providing security, peer discovery, NAT traversal, and relay formation. In addition, we presented novel ways of configuring the VPN through a Web 2.0 envrionment and creating full tunnel VPNs that can be used on centralized, decentralized, and P2P VPNs.

This approach can be used to construct real systems. The VPN in question has been used to construct a publicly and freely available GroupVPN [34] and a SocialVPN [35]. The GroupVPN has been used to construct a Grid Appliance that enables the creation of distributed, decentralized, dynamic computing grids. Over the past 2 years, we have had an active grid deployed for computer architecture research, Archer [31]. Archer currently spans four universities with 500 resources, we have had 100s of users who connect seamlessly to these resources from home, school, hotels, etc. Most recently, a grid at La Jolla Institute for Allergy and Immunology went live with minimal communication with us. Researchers at the Clemson University and Purdue are using this as the basis of their future distributed compute clusters.

While this paper introduces solutions to creating a complete structured P2P based VPN, it introduces many new interesting research problems: 1) distributed load balancing of full tunnel gateways, 2) understanding the long term effects of different automatic relay selection models, 3) understanding the benefits beyond security of using a private overlays for a VPN such as multicast.

References

[1] 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.

[2] J. Yonan. (2009) OpenVPN. http://openvpn.net/.

[3] LogMeIn. (2009) Hamachi. https://secure.logmein.com/products/hamachi2/.

[4] K. Petric. (2009, August) Wippien. http://wippien.com/.

[5] G. LLC. (2009, September) Gbridge. http://www.gbridge.com.

[6] A. Rezmerita, T. Morlier, V. Neri, and F. Cappello, "Private virtual cluster: Infrastructure and protocol for instant grids," in *Euro-Par '06*.

[7] M. Tsugawa and J. Fortes, "A virtual network (vine) architecture for grid computing," *International Parallel and Distributed Processing Symposium*, vol. 0, p. 123, 2006.

[8] X. Jiang and D. Xu, "Violin: Virtual internetworking on overlay," in *In Proc. of the 2nd Intl. Symp. on Parallel and Distributed Processing and Applications*, 2003, pp. 937–946.

[9] A. I. Sundararaj and P. A. Dinda, "Towards virtual networks for virtual machine grid computing," in *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*. Berkeley, CA, USA: USENIX Association, 2004, pp. 14–14.

[10] G. Sliepen. (2009, September) tinc. http://www.tinc-vpn.org/.

[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*, 2008.

[12] W. Ginolas. (2009) P2PVPN. http://p2pvpn.org.

[13] 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*.

[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*, 2006.

[15] D. I. Wolinsky, K. Lee, T. W. Choi, P. O. Boykin, and R. Figueiredo, "Virtual private overlays: Secure group communication in NAT-constrained environments," in *TR-ACIS-09-004*, December 2009.

[16] M. Krasnyansky. (2005) Universal tun/tap device driver. http://vtun.sourceforge.net/tun.

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

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

[19] 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*, 2005.

[20] 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)*, November 2001.

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

[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, in *IPTPS '02*, 2002.

[24] S. Ratnasamy, P. Francis, S. Shenker, and M. Handley, "A scalable content-addressable network," in *In Proceedings of ACM SIGCOMM*, 2001, pp. 161–172. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.8434

[25] P. O. Boykin and et al., "A symphony conducted by brunet," 2007.

[26] J. R. Douceur and J. S. Donath, "The sybil attack."

[27] N. Modadugu and E. Rescorla, "The design and implementation of datagram tls." in *NDSS*, 2004.

[28] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," *IEEE/ACM Transactions on Networking*, 2004.

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

[30] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *SIGCOMM IMW '02*.

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

[32] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, 2003.

[33] (2009) Amazon elastic compute cloud. http://aws.amazon.com/ec2.

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

[35] R. Figueiredo, P. O. B. Boykin, P. St. Juste, and D. Wolinsky, "Social vpns: Integrating overlay and social networks for seamless p2p networking."