On the Design and Implementation of Autonomic, Decentralized GroupVPNs

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

***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.

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, 7) user interfaces for managing the VPN, and 8) efficient IP broadcast and multicast.

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

Centralized approaches (e.g. OpenVPN [1]) require dedicated infrastructures and disallow direct communication between peers, limiting bandwidth and creating inefficient multicast paths, though it is the only approach to support full tunnel operation and guarantee all-to-all communication regardless of NAT and firewall conditions. Centralized systems with P2P links (e.g. Hamachi [2], Wippien [3], Gbridge [4], PVC [5]) 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 rely on centralized relays if direct peer communication across NATs and firewalls fails. Decentralized approaches (e.g., ViNe [6], Violin [7], VNET [8], tinc [9]) require manual configuration of links between members of the virtual network and do not automatically reconfigure when links fail. Existing P2P approaches lack scalability (N2N [10] and P2PVPN [11]).

In this paper, we present a generic construction of a VPN and use this model to assist in comparing other VPN approaches. None of the VPN approaches discussed meet our requirements, leading to our proposal of a novel structured P2P overlay VPN. The contributions of this paper towards P2P VPNs include securing a VPN through the use of private virtual overlays; methods of using Web 2.0 group infrastructures to create and manage a VPN; decentralized, autonomic creation of relays; decentralized full tunnel VPN mode; and efficient multicast and broadcast. 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. We present our novel contributions for P2P VPNs consisting of decentralized relays in Section III, efficient multicast and broadcast in Section IV, full tunnel VPN modes for P2P VPNs in Section V, and user-friendly configuration of VPNs using groups in Section VI. To validate the P2P approach, we compare it against centralized VPNs in Section VII. We present conclusions and real usage of the system in Section VIII.

II. Virtual Private Networks

Common VPN architectures are presented in Table I. This reviews the fundamental features found in all VPN clients and then describes the their implementation in various VPN approaches with particular emphasis on address allocation, peer discovery, and authentication.

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

Figure 1 presents an abstraction of the common features found in all VPN client, 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.

Authentication with an overlay can be performed using various methods including user accounts, signed certificates, shared secrets, or a combination of them. Each approach has varying advantages and disadvantages. Though the most secure is signed certificates as it provides a mechanism to uniquely identify users directly without inquirying through a third party, yet the peer can easily be revoked from the system. Whereas user accounts require a third party to authenticate peers and shared secrets to not lend easy to peer revocation.

A VN device allows applications to communicate transparently over the VPN. By providing mechanisms for injecting incoming packets into and retrieving outgoing packets from the networking stack, VN device enable the use of common network APIs such as Berkeley Sockets, thereby allowing existing application to work over the VPN without modification. While there are various types of VN devices, TAP [12] stands out due to its open source and pervasive nature. TAP allows the creation of virtual Ethernet and / or IP devices and is available for almost all operating systems including Windows, Linux, Mac OS/X, BSD, and Solaris.

VN devices can be configured manually though command-line tools or OS' APIs or dynamically by the universally supported dynamic host configuration process (DHCP) [13]. When a VN device obtains an IP address, it triggers the OS to add a new rule to the routing table directing all packets sent to the VPN subnet 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.

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

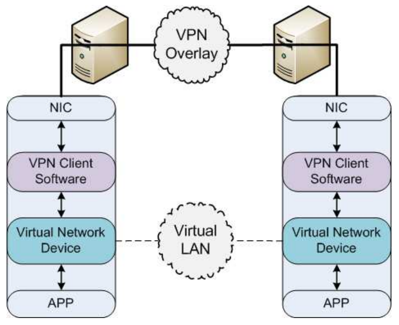


Fig. 1. A typical VPN client. A VN device makes application interaction with the VPN transparent by integrating with the OS network stack.

B. Centralized VPN Systems

OpenVPN is an open and well-documented platform for deploying centralized VPNs. In this paper, it is used 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 providing access to the servers' local resources and the Internet (full tunnel). Likewise, broadcast and multicast packets also must pass through the central server.

While central VPNs do support multiple servers, upon starting the client randomly selects from a list of known servers, implementing a simple load balance. Once connected, the servers provide 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. Servers require additional configuration to communicate with each other beyond adding them to a list.

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 [2] is the first well known centralized VPN that used the ambiguous moniker "P2P VPN". In reality, these systems are better classified as centralized VPN servers with P2P links. Similar VPNs include Wippien [3], Gbridge [4], PVC [5], and P2PVPN[3] [11]. The P2P in tehse systems is limited to direct connectivity between clients orchestrated through a central server, in Wippien it is an chat server while P2PVPN uses a BitTorrent tracker. If NAT traversal or firewalls prevent direct connectivity, the central server can act as a relay. Each approach uses their own security protocols with most using a server to verify the authenticity and setup secure connections between clients.

D. Decentralized VPN Systems

Some examples of systems that assist in distributing load in VPN systems are tinc [9], CloudVPN [14], ViNe [6], VNET [8], and Violin [7]. 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. Links are manually configured.

E. Unstructured P2P VPN Systems

Unlike centralized and decentralized systems, P2P environments require the user to connect to the overlay, which then automatically configures links. 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. The only example of an unstructured VPN is N2N [10]. In N2N, peers first connect to super node 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. 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

An alternative to dealing with scalability concerns in unstructured systems, structured 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 [15]. Some examples of structured systems are Pastry [16], Chord [17], Symphony [18], Kademlia [19], CAN [20], and Brunet [21]. In general, structured systems are able to make these guarantees by self-organizing into a structured topology, such as a 2D ring or a hypercube, deterministically by randomly generated, though uniformly distributed, node identifiers. A key component of most structured overlays is support for decentralized storage/retrieval of information by mapping keys to specific node IDs in an overlay called a distributed hash table (DHT). 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.

Our previous work,[22], describes a virtual, though not private, network using structured P2P overlays. This system provides the underlying VN component for the VPN described in this paper. In this system, 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 was the first to have written a value to that key resulting in a successfully reserved IP address. When an IP packet arrives in the system destined for an unknown host, the VPN queries the overlay's DHT using the hash of the VPN's namespace and remote peer's IP address. A successful query will result in retrieving an overlay ID, the location where the VPN should forward all IP packets for that VPN IP. Since packets are initially routed over the overlay the performance can be undesirably slow, thus after communication has surpassed a few packets, the peers form direct connections via the overlay. This environment is similar to I3 [23] or Internet Indirection Infrastructure, though I3 does not support direct communication and provides abstracted communication methods.

To provide privacy, a VPN must have secure links between communicating peers. In free-to-join structured overlays, this will not be enough as the DHT can easily become polluted or the overlay may become unreliable due to malicious peers [24]. There are many works towards tamper resistant overlays, they do not map well to existing security standards used in VPNs, such as a PKI or shared secrets. These approaches limit the availability of the overlay to trusted users making them no longer free-to-join. This issue has been address in our previous work, [25], which presents a method by which a public overlay is used to bootstrap a private overlay secured by PKI. Peers first join the public overlay using the public overlay's DHT to find other peers and routing to send encrypted and authenticated connection messages. By using this method, structured overlay based VPNs can use a free-to-join to organize a private overlay enabling the construction of an overlay constructed of entirley of peers behind NATs and firewalls, who would otherwise be unable to form their own overlay. In the private overlay, both IP and overlay control messages are secured using the PKI encrypted and authenticated using DTLS [26], thus if a peer misbehaves in the overlay, they can easily be revoked using both a revocation list or a broadcast revocation in the overlay.

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

[3]Due to the similarities between the name P2PVPN and focus of this paper, "P2PVPN" refers to [11] and "P2P VPN" to to the use of P2P in VPNs.

that have Internet connectivity the IPv4 limitation is becoming more and more apparent. To address this issue there have been two focuses: 1) the use of network address translation (NAT) to enable many machines and devices to share a single IP address but preventing bidirectional connection initiation and 2) IPv6 which supports 2^{128} addresses. IPv6 does not necessarily imply direct connectivity as there are no guarantees of NAT disappearing and outbound only firewalls allowing incoming connections.

Approaches for handling NAT and firewall traversal come from two main branches: hole punching and relaying. When performing hole punching, peers simultaneously attempt to create ”holes” in their NAT / firewall devices allowing direct TCP or UDP communication with a peer behind another NAT / firewall. This works by confusing the NAT into believing that the peer behind it initiated the connection. TCP NAT hole punching, though, does not work on systems that use stateful firewalls. Relaying, on the other hand, always works as peers behind NATs communicate with a third peer with whom they have have direct connectivity, though with the 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 centralized P2P approaches that lack dedicated servers, there exists no relay implementation. Though in unstructured networks peers can communicate through the overlay when traversal fails. Similarly 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 reducing bandwidth.

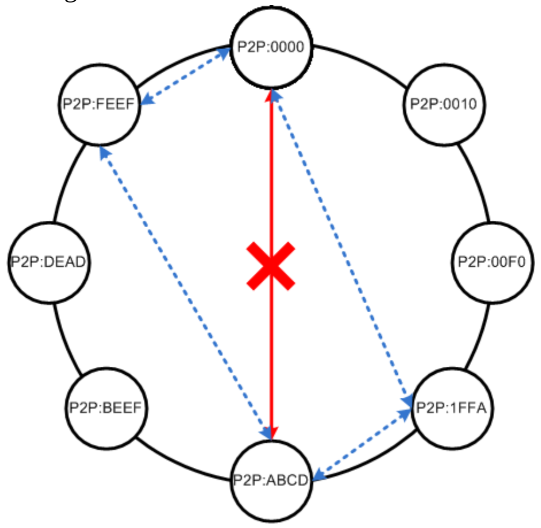


Fig. 2. Creating relays when direct connectivity is not possible. Two members, 0000 and ABCD, are unable to form direct connectivity, thus they exchange neighbor information through the overlay and connect to one of each other’s neighbors, creating an overlap. The overlap then becomes a relay path (represented by dashed lines) for two-hop communication.

P2P systems can address this issue through a distributed, autonomic relaying system, whereby peers form overlapping connections with each others peers, creating two-hop links. 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 through the overlay. Upon receiving this list, the two peers identify the overlap in the neighbor sets to form a two-hop connection. When peers are not located near each other in the overlay, most likely they will have no overlap. This can be addressed by having peers connect to each other’s neighbor set proactively creating overlap, as represented in Figure 2.

In addition to exchanging overlap, peers also exchange some state of their connections with the neighbors. For example, the data shared could be regarding node stability as measured by the age of a connection and proximity as measured by ping latency to the neighbor. When creating overlap or overlap changes, peers review these metrics to decide which subset of the overlap to use as a proxy leaving the remaining peers as reserves.

B. *Usefulness of Relays*

This experiment verifies the usefulness of two-hop over overlay routing using an event-driven simulator that reuses the code base of our prototype VPN to faithfully implement its functionality using event-driven simulated times to emulate WAN latencies. Pair-wise latency in this experiment is set using the MIT King Data Set [27], which consists of all-to-all latencies between 1,740 well-distributed Internet hosts, thus network size of up to 1,740 are evaluated. After starting the overlay and it has reached steady state, the simulator sends packets amongst all peers to derive the average all-to-all latency and hop count for messaging in the overlay. In the low latency relay model, each destination node forms 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. Only peers with two hops or more are analyzed, as a single hop would only benefit under triangular inequalities, which are not a consideration in this work. The simulations were performed on a distributed grid platform, Archer [28], that uses the prototype for its VPN component.

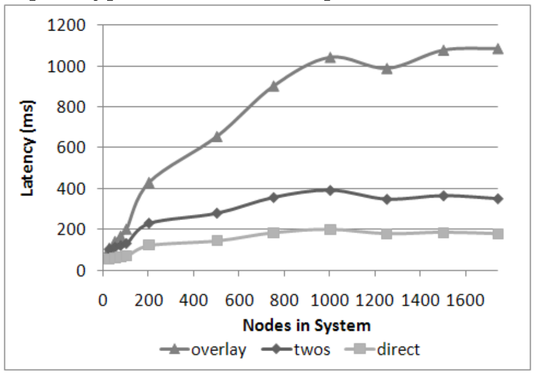


Fig. 3. A comparison of the average all-to-all overlay routing, two-hop relay, and direct connection latency in a ring based structured overlay.

Results are presented in Figure 3. The network sizes began 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, ensuring the validity of this 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 [29] 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 ± 1080 and 58.1 ms ± 35.5, respectively.

IV. Efficient Broadcast and Multicast

The use of a private virtual overlay enables a new method for sending multicast and broadcast packets. In the previous approach, when a broadcast or multicast packet arrived at the VPN, it would query a DHT key where all peers in the VPN would place their overlay address so that they could receive the packets, the packets were sent via the overlay using unicast messaging. Approaches such as broadcasting to the entire overlay were frowned upon because the overlay could consist of peers from other VPNs and those not even involved with VPN operations. In a private virtual overlay, the only peers in the overlay are peers for a single VPN.

An example of a valid ring based structured P2P overlay broadcast algorithm was presented in [25] used to broadcast certificate revocation. The algorithm is called bounded-broadcast, because it is capable of broadcasting to a subset of the ring, though it is also capable of broadcasting to the entire overlay. A bounded broadcast uses the following recursive algorithm: Begin with node x triggering a broadcast message over the region $[x, y]$. x has F connections to nodes in the range $[x, y]$. Denoting the i^{th} such neighbor as b_i, the node x sends a bounded broadcast over a sub-range, $[b_i, b_{i+1})$, to b_i, except the final neighbor. Differently stated, b_i is in charge of bounded-broadcasting in the sub-range $[b_i, b_{i+1})$. If there is no connection to a node in the sub-range, the recursion has ended. The final neighbor, (b_F), is responsible for continuing the bounded broadcast from $[b_F, y]$. When a node receives a message to a range that contains its own address the message is delivered to that node and then routed to others in that range. Figure 4 shows how this bounded-broadcast forms a local tree recursively. The time required for a bounded broadcast is $O(\log^2 N)$ as shown in [30]. To perform a broadcast on the entire overlay, a peer performs the bounded-broadcast starting from its node ID with the end address being the node ID immediately preceding its own in the address space.

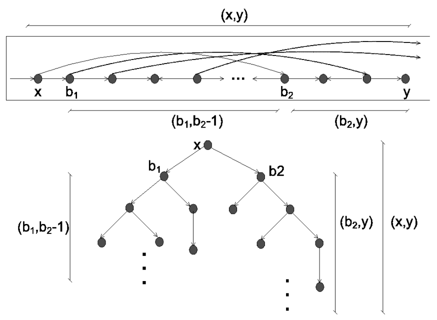


Fig. 4. Bounded Broadcast in range $[x, y]$

A validation of the overlay broadcast method for IP broadcast and multicast communication comparing it to the original DHT method is presented in Figure 5. Metrics measured include bandwidth used by the packet originator and time required for all peers to receive the packet. System bandwidth remains the same as bounded-broadcast requires exactly N messages to broadcast the message to the entire overlay. By using a broadcast algorithm to route multicast and broadcast IP packets instead of the unicast packets, the sending node has much less used bandwidth. The time required by the DHT approach involves the time to retrieve all entrees from the DHT and the latency to the peer farthest away. Whereas the broadcast approach has no DHT lookup and is somewhere around the average latency in the system multiplied by $O(\log^2 N)$.

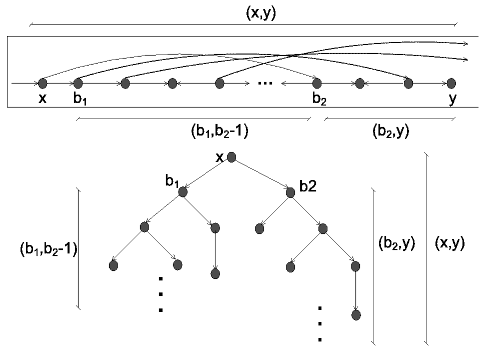


Fig. 5. Bounded Broadcast in range $[x, y]$

V. Full Tunnel VPN Operations

The configuration detailed so far describes a split tunnel: a VPN connection that handles *internal VPN traffic only, not Internet traffic*. Prior to this work, only centralized VPNs currently support full tunnel: providing 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 securely relaying all Internet traffic through a trusted third party, the VPN gateway. Both models are illustrated in Figure 6.

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,

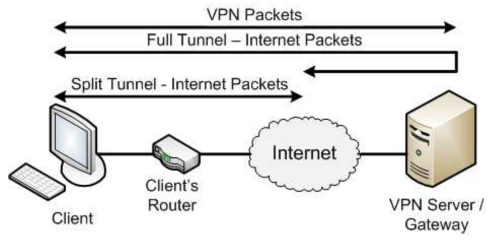

VPN Packets
Full Tunnel – Internet Packets
Split Tunnel - Internet Packets
Internet
Client
Client's Router
VPN Server / Gateway

Fig. 6. 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.

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 V.

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

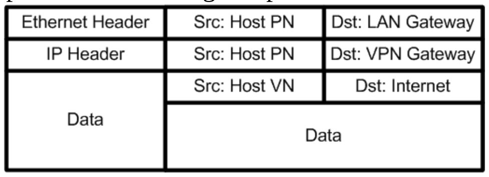
A gateway can be configured through NAT software, like masquerading in IPtables or Internet Connection Sharing with Windows. This automatically handles the forwarding of packets received on the NAT interface to another interface bringing the packet closer to its destination. Similarly, incoming packets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

Following from our previous work on VPNs [22], 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, a virtual IP address provides an endpoint for virtual DHCP and DNS servers. 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 address does, as shown in Figure 7, thus enabling the use of any and remote multiple 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. Typically these packets are rejected, though when in full tunnel 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 receiving peer should verify that the overlay address as being a certified VPN gateway prior to forwarding the packet to the TAP device.



Ethernet Header	Src: Host PN	Dst: LAN Gateway
IP Header	Src: Host PN	Dst: VPN Gateway
Data	Src: Host VN	Dst: Internet
	Data	

Fig. 7. The contents of a full tunnel Ethernet packet. PN and VN are defined as physical and virtual network, respectively.

Peers select the remote gateway by querying the DHT. As there may be multiple gateways, the peer randomly selects one, forwarding packets to that node. Peers occassionaly send pings to the gateway to ensure liveness. If the gateway stops responding, the next ensuing Internet packet will trigger the selection of a new gateway.

Dynamic connections in P2P systems introduce challenges to applying full tunnel VPN modes to P2P VPNs. A simple route rule switch does not work as peers do not know remote peers ahead of time. Watching incoming connection requests and adding routing rules as needed, only enables UDP as 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. Even with that limitation the approach has a significant draw back in 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.

An alternative approach to solving the security problem can be had with no additional routing rules. In this approach, the VPN is 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 [22] 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 the 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 determine if this is the case, this experiment compares it with the traditional routing rule switch. Figure V-C 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 a negligible difference between the full tunnel approaches. The reason why the latency to the gateways public address in the routing swap test is low is due to the ping being sent by avoiding the VPN stack.

	Google	Gateway Private	Gateway Public
Ethernet	70.6	12.9	13.9
Routing	71.4	13.2	11.0
Direct	66.1	N/A	10.9

TABLE II
LATENCY RESULTS FOR FULL TUNNEL APPROACHES IN MS.

VI. 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, the group environment provides 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 [25], revocation can be stored as a CRL on the group site or distributed through broadcast and DHT on the overlay. Though for these forms of systems a user revocation list (URL) as opposed to a CRL simplifies revocation in the case 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, which is used to self-configure the VPN. The configuration is self-contained and requires 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. In the prototype, the only the file system and command-lien approaches have been implemented. 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.

VII. EVALUATION OF VPN MODELS

This experiments explores bandwidth and latency in a distributed VPN system to motivate the usage of P2P links in a VPN. The VPNs used are include our prototype (IPOP), OpenVPN, and Hamachi. OpenVPN represents a typical centralized VPN, while Hamachi represents a well-tuned P2P-link VPN. The evaluation was performed on Amazon EC2 using small instance sized Ubuntu i386 instances to create various sized networks ranging from 1 to 32. OpenVPN uses an additional node as the central server and Hamachi has an upper bound of 16 due to limitations in the Linux version at the time of this evaluation. To perform bandwidth tests, the instances are booted and query an NFS for the list virtual IP addresses, peers are ordered such that half the peers are act as clients and the other half the peers creating a 1 to 1 mapping between all sets. Latency and bandwidth tests are performed using netperf's request-reply and streaming tests respectively. Prior to the start of the tests, peers have no knowledge of each other, except the virtual IP addresses, thus connection startup costs are included in the test. Test are run for 10 minutes diluting the connection initiation overhead but represent an example of real usage. Results from the clients are polled at all locations and averaged together, though the OpenVPN server is measured separately. IPOP and OpenVPN use authenticated 128-bit AES, while Hamachi does not allow configuration of the security parameters and uses the default Hamachi settings.

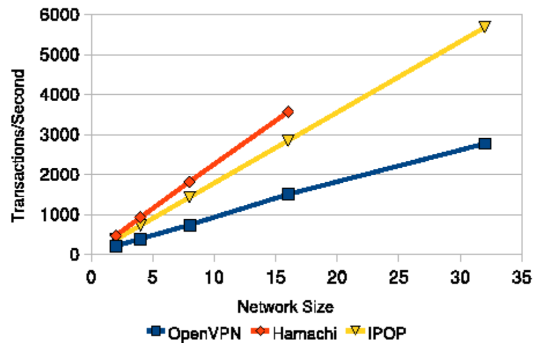


Fig. 8. System transaction rate for various VPN approaches.

Figure 8 and 9 present the results for latency and bandwidth respectively. Latency is measured in transactions of successful request/reply messages. In the latency test, it is obvious that having the central server increases the delay between the

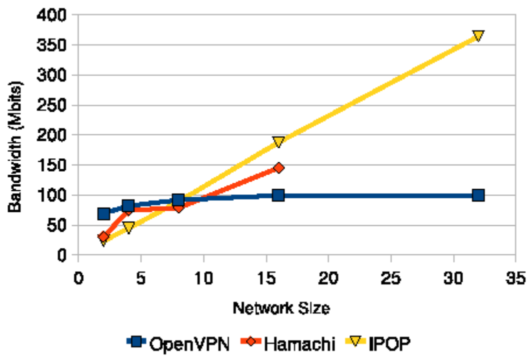


Fig. 9. System bandwidth for various VPN approaches.

client and server and the results degrade more quickly as additional peers are added to the system. In small systems, OpenVPN shines probably due to optimized software, though as the system grows, the system bandwidth does not. By the time 8 peers have entered into the system, both decentralized approaches perform better than the OpenVPN solution. To summarize, decentralized VPN approaches provide better scalability, which can be immediately noticed by low latency times and, as the system grows, available bandwidth.

VIII. Conclusions

As a canonical resource, this paper describe all the necessary components to design and implement decentralized VPN approach designed to facilitate ease of use while maintaining security and scalability. The methods in this paper define an abstraction for the VPN component and the features required in a structured overlay for this style of VPN. The VPN abstraction handles local networking components and interacts with the overlay for communication, link creation, and peer discovery. In addition, we presented novel ways of configuring the VPN through a Web 2.0 environment and creating full tunnel VPNs that can be used on centralized, decentralized, and P2P VPNs.

This approach has been used to construct real systems, such as the GroupVPN [31] and a SocialVPN [32]. 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 [28]. 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 have opted for this approach over centralized VPNs as the basis of their future distributed compute clusters and have actively tested networks of over 1000 nodes.

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] J. Yonan. (2009) OpenVPN. http://openvpn.net/.
[2] LogMeIn. (2009) Hamachi. https://secure.logmein.com/products/hamachi2/.
[3] K. Petric. (2009, August) Wippien. http://wippien.com/.
[4] G. LLC. (2009, September) Gbridge. http://www.gbridge.com.
[5] A. Rezmerita, T. Morlier, V. Neri, and F. Cappello, "Private virtual cluster: Infrastructure and protocol for instant grids," in *Euro-Par '06*.
[6] M. Tsugawa and J. Fortes, "A virtual network (vine) architecture for grid computing," *International Parallel and Distributed Processing Symposium*, vol. 0, p. 123, 2006.
[7] 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.
[8] 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*. USENIX Association, 2004, pp. 14–14.
[9] G. Sliepen. (2009, September) tinc. http://www.tinc-vpn.org/.
[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*, 2008.
[11] W. Ginolas. (2009) P2PVPN. http://p2pvpn.org.
[12] M. Krasnyansky. (2005) Universal tun/tap device driver. http://vtun.sourceforge.net/tun.
[13] R. Droms, *RFC 2131 Dynamic Host Configuration Protocol*, March 1997.
[14] E. Exa. (2009, September) CloudVPN. http://e-x-a.org/?view=cloudvpn.
[15] M. Castro, M. Costa, and A. Rowstron, "Debunking some myths about structured and unstructured overlays," in *NSDI'05: Proceedings of Symposium on Networked Systems Design & Implementation*.
[16] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *International Conference on Distributed Systems Platforms*, November 2001.
[17] I. Stoica and et al., "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *SIGCOMM*, 2001.
[18] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: distributed hashing in a small world," in *USITS*, 2003.
[19] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *IPTPS '02*, 2002.
[20] S. Ratnasamy, P. Francis, S. Shenker, and M. Handley, "A scalable content-addressable network," in *In Proceedings of ACM SIGCOMM*, 2001.
[21] P. O. Boykin and et al., "A symphony conducted by brunet," http://arxiv.org/abs/0709.4048, 2007.
[22] 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*, November 2009.
[23] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," *IEEE/ACM Transactions on Networking*, 2004.
[24] J. R. Douceur, "The sybil attack," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, pp. 251–260.
[25] 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 *1001.2569*, January 2010.
[26] E. Rescorla and N. Modadugu. (2006, April) RFC 4347 datagram transport layer security.
[27] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *SIGCOMM IMW '02*.
[28] R. Figueiredo and et al., "Archer: A community distributed computing infrastructure for computer architecture research and education," in *CollaborateCom*, November 2008.
[29] 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.
[30] J. Kleinberg, "The small-world phenomenon: an algorithm perspective," in *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, 2000.
[31] D. I. Wolinsky and R. Figueiredo. (2009, September) Grid appliance user interface. http://www.grid-appliance.org.

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