Virtual Private Overlays: Secure Group Communication over Decentralized Public Overlays in NAT-Constrained Environments

David Isaac Wolinsky, Kyungyong Lee, Tae Woong Choi, P. Oscar Boykin, and Renato Figueiredo
Advanced Computing Information Systems Lab
University of Florida

***Abstract*—Structured P2P overlays provide a framework for building distributed applications that are self-configuring, scalable, and resilient to node failures. Such systems have been successfully adopted in large-scale Internet services such as content delivery networks and file sharing; however, widespread adoption in small/medium scales has been limited due in part to security concerns and difficulty bootstrapping in NAT-constrained environments. Nonetheless, P2P systems can be designed to provide guaranteed look-up times, NAT traversal, point-to-point overlay security, and a distributed data store. In this paper we propose a novel way of creating overlays that are both secure and private and can be bootstrapped from a public overlay. Private overlay nodes use the public overlay's distributed data store to discover each other, and the public overlay's connections to assist with NAT hole punching and as relays providing STUN and TURN NAT traversal techniques. The security framework utilizes groups, which are created and managed by users through a web interface that can be hosted on the Internet. Each group acts as a Public Key Infrastructure (PKI) relying on the use of a centrally-managed web site providing an automated Certificate Authority (CA). We present a reference implementation which has been used in a P2P VPN (Virtual Private Network). Additionally, we evaluate our contributions using both the P2P VPN in the PlanetLab wide-area testbed as well as with event-driven simulations of the overlay using simulated time delays.**

I. Introduction

While Structured P2P overlays provide a scalable, resilient, and self-managing platform for distributed applications, their adoption rate has been slow outside data distribution services such as BitTorrent and eDonkey. General use of structured P2P systems, especially in applications targeting home and small/medium businesses (SMBs), has been limited in large part due to the difficult nature of securing such systems to the level required by these users. Applications in home and SMBs may need a greater level of trust than what can be guaranteed by anonymous contributors in free-to-join overlays, but these users lack the resources for bootstrapping private P2P overlays particularly in constrained wide-area network environments where a significant amount of or all peers are behind Network Address Translation devices (NAT). This paper presents a novel approach that enables virtual private overlays to be created and managed by members of small/medium groups, leveraging public overlays for bootstrapping, NAT traversal and relaying.

There are many different P2P applications used in home and small business, primarily for collaboration and sharing, including data storage, media sharing, chat, and system maintenance and monitoring. Applications that currently provide these functionalities fall into two categories: anonymous, fully decentralized free-to-join P2P systems and distributed systems with P2P communication that rely on a third party to provide discovery and management. While using a third party service provides a desirable level of trust for many users, it has significant drawbacks such as vendor lock-in, which may result in lost data, down time, and scalability constraints.

An example of a useful small business application that falls into the latter category is LogMeIn's [1] software products LogMeIn Pro and Hamachi. LogMeIn Pro allows users to remotely manage and connect with their machines so long as they are willing to use LogMeIn's software and infrastructure. Hamachi allows users to establish decentralized VPN links using centralized session management. Both applications assist in the remote maintenance and monitoring of computers without requiring the user to implement the networking infrastructure provided by LogMeIn.

Some examples of P2P applications for homes and small businesses in development are P2PSIP and P2Pns. P2PSIP enables users to initiate, through decentralized means, visual and audio communication, while P2Pns allows users to deploy a decentralized naming service. Both applications allow users to contribute and benefit from all members of the system without the regulation of a third party, but lack the ability to allow users to centrally secure and manage their own subset of the systems.

Distributed data store applications like Dynamo [2] and BigTable [3] have the ability to store data using a completely decentralized system. Though these systems are highly scalable and fault tolerant, the software uses an untrusted overlay. Therefore all instances need to run in a secure environment, whether that be in a single institution or across a largely distributed environment using a VPN.

In this paper, we describe the architecture of a system that attempts to balance the benefits of third party services with P2P infrastructures. Two main contributions of this paper are the architecture of a P2P messaging framework that integrates existing datagram-based security and supports bootstrapping multiple virtual private overlays that efficiently multiplex a peer's network resources subject to constraints such as NATs. Our system relies on a completely open infrastructure using

mechanisms that reduce the maintenance and deployment burden that exist with decentralized P2P systems without the loss of ownership due to use of third party systems. The components of our system follow:

1) An application layer security system based upon an existing datagram-based transport security (DTLS) framework to provide secure end-to-end (EtE) and point-to-point (PtP) in P2P overlays[1] while transparently handling security concerns of relay NAT traversal.
2) The bootstrapping of private P2P overlays using existing public overlays. The term "public overlay" refers to a free-to-join bootstrap overlay; nodes in the public overlay are not required to have public IP addresses and can be behind NATs.
3) A group web interface providing an automated front-end handler for a Public Key Infrastructure (PKI) that allows users to create their own secure systems relying on P2P overlays.

Our approach provides an easy mechanism to create trusted overlays in constrained environments using a publicly available untrusted overlay as illustrated in Figure 1. Akin to other virtualization techniques, the virtual private overlay model allows developers to focus on the application, while complexities associated with enforcing isolation and security and messaging are abstracted away. This approach also has benefits for system deployers, whereas insecure systems would require modifying the overlay software to support security, application level security, or the deployment of a decentralized and scalable VPN in addition to the overlay software.

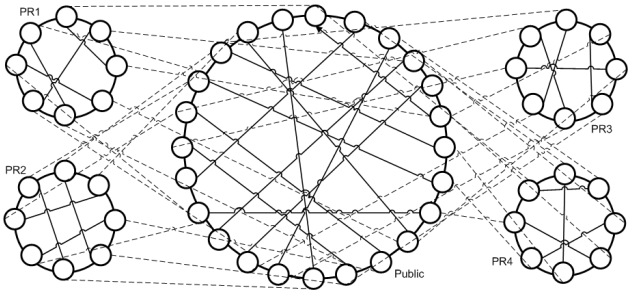

Fig. 1. The use of a single, public overlay to bootstrap multiple, isolated private overlays. The center pool is the public pool. Each node in the private pools has a corresponding partner in the public pool, the relationship is represented by a dashed line. A public overlay node can be multiplexed by more than one private overlay.

The rest of this paper is organized as follows. Section II-A provides background and related work. Section III describes our contributions. In Section IV, we present different usage models and present a real system using the techniques described in this paper, the GroupVPN. In Section V, we use both real and simulated systems to provide further understanding of the approach. We conclude the paper in Section VI.

[1]For our discussion a point-to-point or PtP relationship refers to direct connections between two peers, such as a UDP or TCP socket; whereas, an end-to-end or EtE relationship refers to messages passed through the overlay, which may be routed over many PtP links.

II. Background

In this section, we begin by reviewing structured P2P overlays, followed by constraints that make the creation of secure and private P2P overlays difficult, and related work.

A. Structured P2P Systems

Structured P2P systems provide distributed look up services with guaranteed search time in $O(\log N)$ to $O(\log_2 N)$ time, in contrast to unstructured systems, which rely on global knowledge/broadcasts, or stochastic techniques such as random walks [4]. Some examples of structured systems can be found in [5], [6], [7], [8], [9]. In general, structured systems are able to make these guarantees by self-organizing a structured topology, such as a 2D ring or a hypercube.

The node ID, drawn from a large address space, must be unique to each peer, otherwise an address collision occurs which can prevent nodes from participating in the overlay. Furthermore, having the node IDs well distributed assist in providing better scalability as many algorithms for selection of shortcuts depend on having node IDs uniformly distributed across the entire address space. A simple mechanism to ensure this is to have each node use a cryptographically strong random number generator. Another mechanism for distributing node IDs involves the use of a trusted third party to generate node IDs and cryptographically sign them [10].

As with unstructured P2P systems, in order for an incoming node to connect with the system it must know of at least one active participant. A list of nodes that are running on public addresses should be maintained and distributed with the application, available through some out-of-band mechanism, or possibly using multicast to find pools [5].

Depending on the protocol, a node must be connected to closest neighbors; optimizations for fault tolerance suggest that it should be between 2 to $\log(N)$ on both sides. Having multiple peers on both sides assist in stabilizing the overlay structure when experiencing churn, particularly when peers leave without warning.

Overlay shortcuts enable efficient routing in ring-structured P2P systems. The different shortcut selection methods include: maintaining large tables without using connections and only verifying usability when routing messages [5], [8], maintaining a connection with a peer every set distance in the P2P address space [6], or using locations drawn from a harmonic distribution in the node address space [7].

Most structured P2P overlays support decentralized storage/lookup 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 nearby. This sort of mapping and data storage is called a distributed hash table (DHT) and is a typical component of most structured P2P systems.

B. Constraints in Structured P2P Systems

1) Communication Between Nodes: There are two mechanisms for message routing in a P2P overlay: iterative or

recursive. In iterative routing, the sender of a packet will contact each successive member in a path directly until it find the destination node, at which point it sends the packet directly to the destination. In recursive routing, messages are sent through the overlay via forwarding from one peer to the next until arriving at the destination. Iterative routing can easily be secured using stream and datagram based security such as TLS [11] and DTLS [12], because the sender initiates all messages. In contrast, in recursive routing, EtE communication cannot be secured in the same fashion by through socket-based TLS and DTLS, the sender because messages are routed through intermediate nodes.

2) Private and Secure Overlay Subsets: While EtE authentication and privacy is key for many applications, it does not increase the reliability of the structured overlay. In a free-to-join system, malicious peers can easily intercept packets for eavesdropping, discard or tamper them. To deal with this issue, each overlay node could participate with a subset of other nodes in a secure system where each peer would maintain a routing table containing a subset of those involved. Each node would have to distinguish between the different messages, handing them to unique routers for each group, handle connectivity of the subset during churn, and providing a distributed data store for use by this subset. Achieving this functionality can require substantial modifications to the core overlay messaging and storage primitives. In this paper we advocate a virtualization approach that multiplexes private overlays that provide the same abstraction of the underlying public overlay, and thus can reuse core overlay primitives without modifications.

3) NATs: Another key issue faced by P2P systems is the handling of NATs. A recent study [13] has shown that 30%-40% of nodes in P2P systems are behind NATs. In environments where there are NATs, iterative routing can be significantly more difficult to deploy, since each message sent may require multiple NAT traversals. There are a few types of NATs that can be traversed to support bidirectional communication through UDP-hole punching, known as STUN. The remaining categories of NATs either do not support UDP-hole punching due to how the NATs do port mapping or support only TCP communication and block all UDP communication. In [14], the authors evaluated many common NAT boxes and determined that STUN works on approximately 70% of NAT devices. Unlike UDP hole punching, most TCP techniques require either super-user access or be configured with external software packages that require super-user access. While the remaining device require an intermediary or relay to pass packets between the peers behind the NATs. Though relays or TURN style NAT traversal presents issues similar to EtE communication in recursive routing, namely, each node will authenticate itself with the relay, but they will not readily be able to authenticate each other using socket-based TLS or DTLS.

C. Related Works

BitTorrent [15], a P2P data sharing service, supports stream encryption between peers sharing files. The purpose of BitTorrent security is not to keep messages private but to obfuscate packets to prevent traffic shaping due packet sniffing. Thus BitTorrent security uses a weak stream cipher, RC4, and lacks peer authentication as symmetric keys are exchanged through an unauthenticated Diffie-Hellman process.

Hamachi [16] provides central group management and a security infrastructure through a Web interface. Their security system has gone through two revisions as documented in [17]. Initially peers learn of each other through Hamachi's central system, which leads to the creation of secure links. In their original approach, they use a system similar to a Key Distribution Center (KDC), which requires that all security sessions initiate through Hamachi's central servers. In the latest version, this model has been retained but with the addition of an external PKI, which avoids the man-in-the-middle attack but with has the additional cost of maintaining both an external CA and certificate revocation list (CRL). Hamachi also supports STUN, or NAT hole punching, and TURN style NAT traversal, though TURN requires the use of Hamachi's own relay servers. Because Hamachi is closed, it disables users from hosting their own infrastructures including session management and relay servers.

Skype [18], like P2PSIP, allows for decentralized audio and video communication. Unlike P2PSIP, Skype is well-established and has millions of users and is also closed. While Skype does not provide documentation detailing the security of its system, researchers [19], [20] have discovered that Skype supports both EtE and PtP security. Though similar to Hamachi, Skype uses a KDC and does not let users setup their own systems.

The RobotCA [21] provides an automated approach for decentralized PKI. A RobotCAs receives request via e-mail, verifies that the sender's e-mail address and embedded PGP key match, signs the request, and mails it back to the sender. RobotCAs are only as secure as the underlying e-mail infrastructure and provide no guarantees about the person beyond their ownership of an e-mail address. A RobotCA does not provide features to limit the signing of certificates nor does it provide user-friendly or intuitive mechanisms for certificate revocation.

Three approaches that propose a public overlay to create sub-overlays are [22], [23], and [24]. The approach described in [22] proposes the use of a universal overlay as a discovery plane for service overlay. The argument is that a participant of an overlay must support all services provided by that overlay such as multicast, DHT, or distributed search. Our work has the same foundations as this paper, but takes the idea further by using the universal overlay for NAT traversal additionally we provide mechanisms for applying PKI techniques for PtP and EtE messaging in the service overlays. Similarly, Randpeer [23] uses a common overlay along with a sub-netting service to create individual networks for applications

and services, though the project has seen little activity and lacks implementation details. Unlike the previous two, [24] limits the sub-overlay for the purpose of establishing multicast groups, though their approach lacks discussion on how nodes discover and form a new overlay as such their approach is limited to simulations.

Distributed data store applications like [2], [3] require that all machines have symmetric connectivity additionally like [25], suggesting the use of a third party application to ensure trust amongst all overlay participants. This is an example use case that is explicitly targeted by our system on the presumption that there are not sufficiently easy to use decentralized VPN software applications [26], [27] and even if there were it is undesirable to have additional setup requirements.

While there has been much research [10] in securing overlays through decentralized mechanisms that attempt to prevent a collusion known as a Sybil attack [28], these mechanisms do not create private overlays. While one approach mentioned does provide a natural lead into such environments, which is the use of a pay-to-use service to mitigate the chances of an overlay attack, whereby the pay to use service uses a CA to sign node IDs. The work does not describe how to efficiently implement such a system. Other similar research attempts to create trusted systems [29], [30] but with anonymous members, though it could be reasonably argued these services are not applicable to small or medium business, which would prefer to have a private overlay. None of the works discuss how to apply such models to systems that are constrained in network connectivity, e.g by NATs.

III. Bootstrapping Secure and Private Overlays

In this section, we explain the individual components of our contribution. We begin by describing the application of SSL-derived protocols for EtE and PtP security in overlays, which leads to a discussion on the use of a group web interface to provide a user-friendly PKI, and concludes with our approach to bootstrapping private and secure overlays in constrained environments using public overlays.

A. Secure Overlays

Securing EtE and PtP communication in overlays using iterative routing with servers using static IP addresses can easily be done using TLS or DTLS with certificates bound to the servers static IP address. This approach does not port well to systems that use recursive routing. First, TLS cannot easily be used because overlay routing in NAT-constrained environments is traditionally done through datagrams and not streams, which would require overlays to implement a reliable stream to use TLS. DTLS can be applied because it supports lost and out of order messages, though the implementation must support usage without a socket. The problem then becomes how to deal with identity. In this section, we discuss how to bind security protocols and a certificate model to an overlay system.

The key to our approach is abstracting the communication layer, making EtE and PtP traffic appear identical. In this approach, all messages are datagrams that are sent over abstracted senders and receivers as filters illustrated in Figure 2. This allows us to use secure tunnels over these links with no application changes.

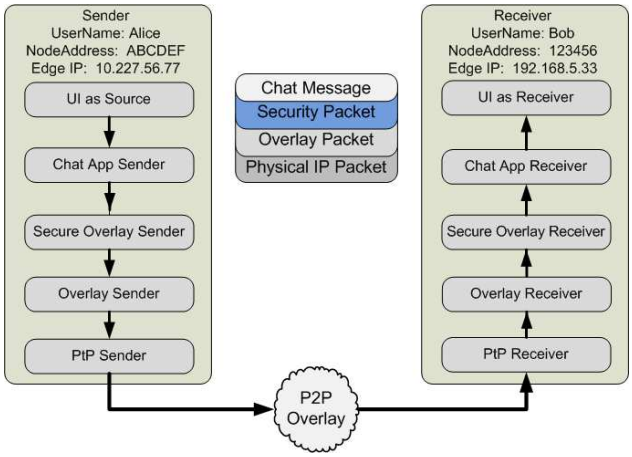

Fig. 2. An example of the abstraction of senders and receivers using a EtE secured chat application. Each receiver and sender use the same abstracted model and thus the chat application requires only high-level changes, such as verifying the certificate used is Alice's and Bob's, to support security.

Exchanged certificates need a mechanism to verify authenticity. Like an Internet browser, this verification should happen automatically with no user intervention. Typically for TLS and DTLS, the certificates have the Web site's IP address or domain name as part of the certificate's common name field. In our system, we bind the certificate to each individual node ID. That way, a single certificate cannot be used for multiple peers in the overlay, making it difficult for an adversary to launch Sybil attacks.

1) Forming the Connection: In overlay systems, a peer's connection manager requests to make an outgoing connection to another peer in the system. This triggers the creation of a socket (UDP or TCP), which is wrapped in the abstracted sender and receiver models. The abstracted models arrive into the security handler, which authenticates in both directions and creates a secure session. The session is wrapped in the same abstracted model and presented to the overlay system as a direct connection to the remote peer. To keep the system abstract, the security model and the wrapped sockets know nothing about the overlay, and so the overlay should verify the certificate to ensure identity.

Because EtE communication is application-specific, it requires a slightly different path. For that purpose, we have a specific module that allows an application to request a secure EtE sender; the application needs to be prepared to deal with the process of handling verification. Once an application requests the sender, the module passes a sender / receiver model to the security handler, like in the PtP process. Once the security initialization has completed, the resulting sender / receiver is verified automatically for proper identity. If that

succeeds, messages sent using the EtE sender will arrive at the remote party, decrypted and authenticated by the security handler, and delivered to the overlay application, who will deliver to the remote party's handler for such messages. Since overlay application will be sending and receiving unencrypted as well as encrypted EtE traffic, the handler must verify that the packet was sent from a secure end point. This assumes that an application using an overlay has already implemented verification of node ID to some application mapping. For example, an application could be aware that node ID X maps to user Y, therefore if a secure message coming from node ID Z says that it is user Y, an application should drop the packet.

2) Datagram Constraints: Since UDP is connectionless, applications that use it can easily be victims of denial of service attacks. This is because packets sent to the receiver can have a spoofed source address, unless the outgoing gateway prevents this from occurring. Whereas with TCP, it would be significantly harder to perform the same attack unless performed as a man-in-the-middle. To reduce the potential of these spoofing attacks prior to establishing a secure connection, DTLS (like Photuris [31]) uses a stateless cookie for each remote peer. In DTLS, the cookie is usually based upon the remote peer's IP and the current time. In our model, which deals with abstracted systems and IP addresses are likely to be NAT-translated, this approach does not work. Because we are building on existing senders and receivers that already have state, we use the object's memory pointer or hash value instead. Though this leads us down a similar path of denial of service, it does so at a much slower rate.

B. Private Overlays

The main components involved in the starting and maintaining a private overlay are 1) dissemination of the security credentials and its name, 2) connecting with and storing data in the public overlay, and 3) discovering and connecting with peers in the private overlay. Step 1) can be application-specific; we propose a generic interface that is useful in many applications, through the use of groups as described in Section Section III-C. For 2), we presume the usage of a structured overlay as described in Section II-A. In this section, we discuss 3), the steps involved in creating and connecting to a private overlay after the user has obtained group information and has connected to a public overlay.

To connect with and create a private overlay, the application performs the following steps: 1) connect to the public overlay; 2) store node ID in the public overlay's DHT at the private group's key; 3) query the public overlay's DHT at the private group's key; 4) Start an instance of the private overlay with the well-known end points being those of the node IDs retrieved from the DHT; and 5) Upon forming a link with a member in the private overlay, the node follows the general approach for linking to neighbors and shortcuts but using security to restrict connections to only members of the private overlay

The node should maintain membership in the public overlay when connected with the private overlay. This is needed for two reasons: first, so that other peers can discover the node while following the same set of steps; and second, for NAT traversal purposes, as discussed in the next paragraph. Because the public overlay and its DHT provides a means for discovery, nodes must maintain their node ID in the public key's DHT. A data inserter must constantly update the lease for the data object, otherwise the data will be removed. This is because DHTs are implemented as leasing systems or as soft state, whereby data objects are inserted with a time to live and removed upon expiration.

During the formation of the private overlay, peers may find that they are unable to form direct connections with other members of the private overlay even while using STUN based NAT traversal. We propose two solutions to address this problem: 1) to use TURN NAT traversal in the nodes overlay as discussed in [27] and 2) use the public overlay as an extra routing massive TURN infrastructure. The TURN NAT traversal technique has both peers connect with each others near neighbors in order to form a 2-hop connection with each other. The 2-hop route can either be enforced through a static route or through EtE greedy routing. Due to the abstractions in the system, the public overlay can be treated as another mechanism to create PtP links, thus while packets may use EtE routing on the public overlay the private overlay nodes treat it as a PtP connection thus all communication is secured. This approach can be further enhanced by allowing the private overlay to apply the TURN NAT traversal technique to the public overlay. To do this, the private overlay must be capable of requesting a direct connection between its node and the remote peer in the public overlay. This would trigger the eventual creation of a 2-hop relay connection as presented in Figure 3.

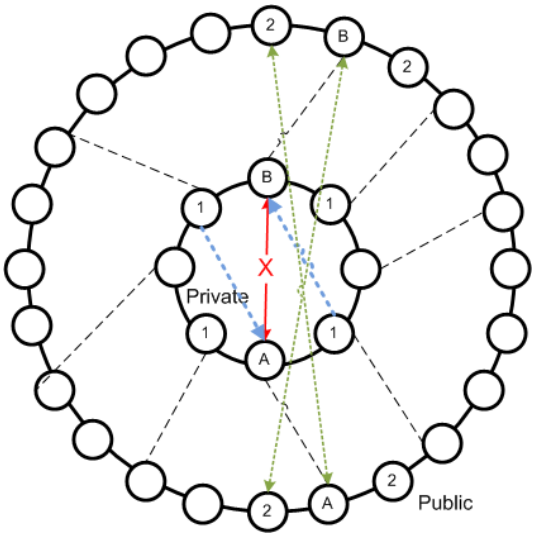

Fig. 3. Creating relays across the node address space, when direct connectivity is not possible. Two members, A and B, desire a direct connection but are unable to directly connect, perhaps due to NATs or firewalls. They exchange private, 1, and public, 2, neighbor information through the private overlay and connect to one of each other's neighbors, creating an overlap. The overlap then becomes a PtP relay path (represented by directed, dashed lines), improving performance over routing across the entire overlay.

Another concern we address is the cost of having to maintain additional connections for each additional private overlay. For this, we propose the use of *pathing*, which multiplex a single UDP or TCP socket to support multiple overlay nodes. This model is supported because of the abstraction done on senders and receivers. Thus a UDP or TCP connection can be easily wrapped inside an arbitrary packet, in this case, a pathing connection. Upon sending a packet, the message is prepended with path information. When the remote side receives a packet, it parses and removes this pathing information and relays it to the appropriate receiver, i.e., overlay node. We validate this approach in Section V-G.

When overlays are small and have significant churn, it is easy for data stored in the overlay's DHT to be lost. This can be improved by also supporting broadcast in the private overlay. In this model, each peer acts as a storage point for all data critical to itself. If another peer cannot successfully find data stored at a specific key in the overlay, it can make use of broadcast over the entire overlay in an attempt to find the result. The technical details of our broadcast implementation are described at the end of Section V-A and an evaluation of our revocation approaches can be found in Section V-F.

During our evaluation, we discovered that in certain cases the private overlay would not form a proper well-formed state but rather more than one distinct or partitioned overlays creating a fragmented overlay. The underlying issue was that the partitioned overlays believed they were in a well-formed state and thus never reviewed the DHT list to determine if there were peers that should be their neighbors. This caused the overlay to remain fragmented until either a new peer joined or enough peers left causing the nodes to believe they are in a non-well-formed state and require bootstrapping links or a more proactive method. The reason the issue even existed was that states of significant churn, especially during a bootstrapping of a significant amount of new nodes in the system, the DHT list can become unstable with each set of node potentially seeing different lists. Of course by the end, the lists would be well-formed, but at that point, the overlay would have already fragmented.

To proactively solve the fragmentation issue, the node performs the following steps: 1) continuously query the DHT; 2) upon receiving the DHT query result, the node determines if there is a peer with whom it should be connected to such as that it is closer in the address space then any of its current neighbors; 3) form a connection with that peer; and 4) the system should automatically at this point in time realize the network fragmentation and heal itself. In our system, this involved creating a bootstrapping connection with the peer. Upon a successful connection, the system automatically causes the networks to heal.

C. Group Overlays

To establish trusted links, we use the PKI model, where a centralized CA (for a group) signs all client certificates and clients can verify each other without CA interaction by using the CA's public certificate. However, setting up, deploying, and then maintaining security credentials can easily become a non-negligible task, especially for non-experts. Most PKI-enabled systems require the use of command-line utilities and setting up your own methods for securely deploying certificates and policing users. While this can be applied to an overlay, our experience with real deployments indicates that usability is very important, leading us to find a model with easy to user interfaces. In this section, we present our solution, a partially automated PKI reliant on a redistributable group based web interface. Although this does not preclude other methods of CA interaction, our experience has shown that it provides a model that is satisfactory for many use cases.

1) Joining the Group Overlay: Membership of an overlay maps a set of users as a group. This led us to the model of using a group infrastructure as a means to apply a PKI. Using our system, a user can host an individual or multiple groups per Web site. The creator of the group becomes the default administrator, and users can request access to the group. Upon an administrator approving, users are able to download configuration data containing overlay information and a shared key used by the overlay application to communicate securely with the web interface. The shared key uniquely identifies the user to the web site allowing the application to securely send certificate requests. By default, the Web site automatically signs all certificate requests, though it is not limited to this model. Two other approaches are 1) require the user to submit a request and wait for an administrator to verify each request and 2) set a maximum amount of automatic request signings then requiring administrative approval for more.

As stated in Section III-A, the certificate request is bound to the application's node ID, which can be generated by the CA or the application. Additionally in the group system, the certificate also contains the user who made the request and the group for which the certificate is valid. Not only does this ensure that a single certificate can only be used for each node instance, but it reduces the amount of state necessary to revoke a user from a system. Specifically, to revoke a user, the CA would only need to provide a signed revocation notice containing the user's name and not every one of the previously signed certificates.

Upon receiving a signed certificate, the overlay application can connect to the overlay where all PtP traffic will be secured and, optionally, so can EtE traffic. It is imperative that any operations that involve the exchanging of secret information, such as the shared secret, be performed over a secure transport, such as HTTPS, which can be done with no user intervention.

2) Handling User Revocation: Unlike decentralized systems that use shared secrets, in which the creator of the overlay becomes powerless to control malicious users, a PKI enables the creator to effectively remove malicious users. The methods that we have incorporated include: a user revocation list hosted on the group server, DHT events requesting notification of peer removal from the group, and broadcasting to the entire P2P system the revocation of the peer.

A user revocation list offers an out-of-band distribution mechanism that can not easily tampered, whereas communi-

cation using the overlay can be hampered by Sybil attacks. The revocation list is maintained on the Web site and updated whenever an administrator removes a user, or a user leaves the group. Additionally it can be updated periodically so that a user can verify that the revocation list is up to date.

However, because the user revocation list requires centralization, users should not query it prior to every communication nor periodically during conversations. In addition to support for polling the revocation list, the use of the DHT and broadcast provides active notification of user revocation. Revocation through the DHT method allows a peer to request notification if another peer is revoked from the group. To subscribe for this notification, the peer inserts its node ID at the peer's revocation notification key, which we represent as a hash of its node ID. Upon revocation, the CA will first insert a revocation notice at this key and then query the key for all node IDs notifying each of them of the revocation. The insertion of the revocation notice handles a race condition, where a peer may insert its ID but never receive a notification. Thus after inserting the request for notification upon revocation, the peer should ensure that a revocation has not occurred by querying the DHT to verify the CA has not inserted a revocation.

When the group is securing PtP traffic, the DHT approach does not effectively seal the rogue user from the system until all peers have updated the revocation list. A peer may continuously connect to all peers in the system until they have all queried the DHT key prior to verification. Due to this issue, we do not employ this model in the group overlay. Instead a broadcast will ensure that all peers in the system do know about the revocation. The strength of the DHT approach exists when using a public free-to-join and an application secured by the group, as in [27].

In Section V-F, we present more technical details and an evaluation with focus on latency and bandwidth of these three methods.

Because the security framework is based on PKI, another approach that is also supported is the use of certificate revocation lists (CRL) found in most CA systems. The advantage of a CRL and revoking individual certificates is the ability to remove a subset of a user's node, particularly useful in the case that the user was not malicious but that some of their nodes had been tampered or hijacked.

IV. Applications

In this section, we present applications and potential ways to configure them to use a private overlay. The applications we investigate include instant message, VPNs, and social networks. The key to all these applications is that users can easily host their own services and be discovered through the use of a NAT-traversing, structured overlay network.

A. Chat Rooms

Chat rooms provide a platform for individuals with a common interest to find each other, group discussion, private chat, and data exchange. One of the most popular chat systems for the Internet is Internet Relay Chat (IRC). As described in [32], IRC supports a distributed tree system, where clients connect to a server, and servers use a mixture of unicast and multicast to distribute messages. The issues with IRC are documented by [33], namely, scalability due to all servers needing global knowledge, reliability due to connectivity issues between servers, and lack of privacy. Private overlays could be extended to support the features of IRC and potentially deal with these inherent issues. Each chat room would be mapped to a private overlay and the public overlay would be used as a directory to learn about available chat rooms and request access. Structured overlays do not require global knowledge and can be configured to handle connectivity issues. Additionally, IRC by default uses clear text messaging and even if security is used a server will be aware of the content of the message, two issues resolved by using PtP security in a private overlay chat room.

B. Social Networks

Social networks such as Facebook and MySpace provide an opportunity for users to indirectly share information with friends, family, and peers via a profile containing personal information, status updates, and pictures. Most social network structures rely on hosted systems, where they become the keepers of user data, which creates privacy and trust concerns. Private overlays can remove this third party, making users the only owner of their data. For this model, we propose that each user's profile be represented by a private overlay and that each of their friends become members of this overlay. The overlay will consist of a secured DHT, where only writes made by the overlay owner are valid and only members of the overlay have access to the content stored in it. In addition to bootstrapping the private overlays, the public overlay would be used as a directory for users to find and befriend each other. For fault tolerance and scalability, each user provides a copy of their profile locally, which will be distributed amongst the private overlay in a read-only DHT, therefore, allowing the user's profile to be visible whether they are offline or online. Each user's social network would than consist of the accumulation of the individual private overlays and the public overlay.

C. P2P VPNs

As described in [27], private overlays enable P2P VPNs. The most common type of VPNs are centralized VPNs like OpenVPN, which requires that a centralized set of resources handle session initialization and communication. Another approach taken by Hamachi and many others is to maintain a central server for session initialization but allow communication to occur directly between peers and providing a central relay when NAT traversal fails. Using a structured private overlay allows users to host their own VPNs, where each VPN end points is responsible for its own session initialization and communication. The private overlay also provides mechanisms for handling failed NAT traversal attempts via relaying.

D. Multicast

The topic of secure multicast has been a focus of much research citations. Using an approach similar to CAN [24], a

private ring can form a ring where all nodes are members of the multicast group with the additional feature that you can trust that your audience is limited to those in the overlay. The main advantage of such multicasting technologies would be for wide-area, distributed multicast for the purposes of systems supports different types of multicast applications such as light weight multicast DNS / DNS-SD (service discovery) or more heavy weight but fault tolerant like streaming audio and video.

V. Implementation Details and Evaluation

In this section, we describe our prototype implementation of a secure, private overlay followed by evaluation to quantify network overheads.

A. Our Implementation – Brunet

Our implementation uses the Brunet [34] library, which is a P2P system based upon the concepts introduced by Symphony [7]. Its topology is a one-dimensional ring, where peers connect with up to the four peers closest to themselves in both directions in the node ID space. Shortcuts are based upon a harmonic distribution and the use of proximity [35]. The system supports distributed versions of STUN or hole-punching and TURN-like or relay based NAT traversal [27]. The system uses a DHT as a distributed data store using a replication algorithm that spreads a single key, value pair throughout the ring as described in [36]. Additionally, the overlay has support for autonomic [37] and manual creation of single-hop connections and double-hop through relaying when NATs prevent direct connections. Furthermore, we have developed a P2P VPN on top of this stack, which has been used for over three years to support a ad hoc distributed grid [38], [39].

The focus of this paper is on security and the bootstrapping of private pools. In the previous sections, we have abstracted the implementation to generic overlays, while in this section, we will present our experiences and lessons learned in applying security protocols to the overlay. To provide security, we investigated two approaches: reusing OpenSSL's DTLS implementation or making our own platform-independent DTLS using C# and cryptography routines provided by .NET. Because we are sending messages over unreliable mediums such as UDP sockets, relays, and an overlay, we could not reuse SSL or TLS. Additionally we want EtE security for relays and overlay communication, the security needed to be implemented through a filter or only in memory and not on a socket.

While OpenSSL is a de-facto standard, with US federal government approved code, and platform portability, during our use of the platform we experienced several issues. The portability provided by OpenSSL is limited to API (Application Programming Interface) and not ABI (Application Binary Interface), which requires a platform-specific library be either installed or distributed with the application. Whereas the purpose in using C#, like Java, is that a single binary can run on any platform. To use unmanaged libraries from a managed language requires a marshaling wrapper to handle the translation; there is such a wrapper for OpenSSL [40]. A constraint we found when using the library was the naming of the OpenSSL libraries was not consistent across platforms and, additionally, Windows lacked a formal installation method. In using the OpenSSL library with DTLS, in version 0.9.8k, renegotiation of security parameters was broken and would result in a deadlocked DTLS session, whereas in 1.0.0-beta3 (the latest released beta) DTLS renegotiation worked; however, it would often segmentation fault.

Due to the constraints of OpenSSL, we have architected the system to support multiple security frameworks; OpenSSL DTLS is one supported option, but the system also supports a custom stack written in C#. To provide for flexibility between the two approaches, we created a security overlord that treats each approach like a filter. Treating each implementation as a filter allows incoming control and data messages to be pushed into the object and data and control messages to be pulled out of the object.

Implementation of an OpenSSL DTLS filter was non-trivial, as documentation is sparse providing the possibility for varied approaches. Traditionally, DTLS uses the DGRAM (datagram) BIO (I/O abstraction) layer, which provides a reliable UDP layer. Because we need a filter, so that we can do both EtE and PtP traffic, we used one memory BIO for incoming traffic and another for outgoing traffic. Memory BIOs provide pipes using RAM: data written to the BIO can then be read in a first in, first out ordering. Incoming messages written to or outgoing messages read from the DTLS read or write BIOs, respectively, are either encrypted data packets or handshake control messages. Sending and receiving clear text messages occur at the DTLS SSL object layer. The pathway for sending a clear text packet begins with the user performing an SSL_write operation, retrieving the encrypted data by performing a BIO_read on the write BIO, and sending the data over the network. At the remote end, the packet is passed to the SSL state machine by performing a BIO_write on the incoming BIO followed by a SSL_read; the result will be the original clear text message. This process also needs to handle control messages; we provide clear context in Figure 4. As an aside, OpenSSL supports a SSL filter BIO, though it will not work for this purpose as BIOs that are inserted are expected to have two pipes, like a socket or two memory buffers. Also the only benefit of using the filter BIO would be that it manages auto-renegotiation, which can be implemented in user code by monitoring time and received byte count. Other operations such as certificate verification and cookie generation are handled by SSL callbacks, which hook into our security framework.

While we believe OpenSSL's DTLS to be a superior choice due to its prevalence and being well studied, it is non-trivial to make available for all platforms. Because our goal is to provide a safe yet easy to use package, we leave the decision up to the user which protocol to use. We believe home users and those interested in testing the system will start with the .NET security stack and migrate to the OpenSSL DTLS stack.

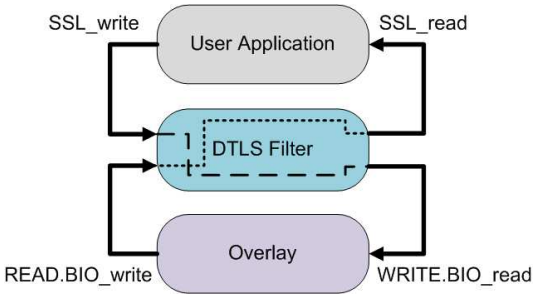

Fig. 4. The DTLS filter.

An important component of security is the handling of revocations handled by our broadcasting mechanism. Though we call our broadcasting model bounded-broadcast because it is capable of broadcasting to a subset of nodes, 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 5 shows how this bounded broadcast forms a local tree recursively. The time required for a bounded broadcast is $O(\log^2 N)$. For security purposes, revoked peers are not included in the set of neighbors during the recursion stage.

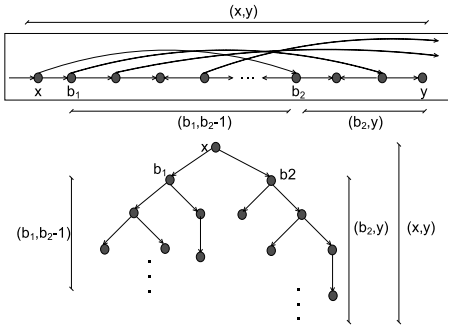

Fig. 5. Bounded Broadcast in range $[x, y]$

B. Experimental Setup

We use three quantitative methods to evaluate our proposed approach: network modeling, simulation and deployments on an actual wide-area system, PlanetLab [41].

1) Network Modeling: To model our system, we create a structured overlay of the usual dimensions: N peers with each peer having 3 near neighbors on both sides and $O(.5 \log N)$ shortcuts where N is the network size. The system is fully connected and shortcuts are optimally chosen based upon their location in the node ID space using a harmonic distribution. Once the model has been fully generated, we can perform tests to determine the time it takes to route packets on the overlay using the same modules that are used in routing in the real system. We employ the MIT King data set [42] to determine the pairwise latency between nodes. The MIT King data set consists of latency between DNS servers distributed globally on the Internet. Since the MIT King data set only covers 1740 nodes, we randomly distributed the nodes in the address space placing multiple nodes at the same "physical" location when necessary.

2) Simulations: To simulate our system, we have developed a module that applies an event-driven time simulation to our P2P software. This allows us to verify correct behavior in the system prior to testing out on real systems, such as PlanetLab, as well as to perform experiments in a controlled environment, which simplifies the evaluation while faithfully modeling the overlay behavior. Like the Netowrk Modeler, we also employ the MIT King data set [42] for pairwise latency between peers. To run the simulations, we employed Archer [38], which uses our P2P VPN software to form a distributed and decentralized computing grid for research into computer architecture.

3) PlanetLab: PlanetLab [41] is a consortium of research institutes sharing hundreds of globally distributed network and computing resources. PlanetLab provides a very interesting environment as there is constant unexpected churn of machines due to the extreme load placed on the resources and system restarts. Complementary to simulation, PlanetLab gives us a glimpse of what to expect from the P2P software stack when used in an actual environment subject to higher variance due to resource contention and churn. Due to the time required and complexities in working with PlanetLab, we limit the number of experiments we used it for, focusing on the connection time to a private overlay.

C. Connecting to the Private Overlay

In this experiment, we seek to determine the overall time it takes for a node to become connected to the public overlay, and then for a subsequent node to become connected to the private overlay. By connected, we mean a node has a connection with the nodes whose IDs are closest in the ring (i.e., neighbors with IDs both smaller than and larger than the node who is joining). The results of this experiment show the time it takes to connect to a public overlay, query the DHT for private overlay information, and then connect to a private overlay with and without security.

To summarize the connection process, we first boot the public node. Upon becoming connected, the private node first inserts into the DHT its subring information using an automated lease extender. Afterwards, we start the private node and begin constantly querying the DHT key for information from other nodes. As soon as this is retrieved from the DHT, it is appended to the list of nodes for the state machine to attempt connecting with. Once the state machine reaches a

connected state, the test is terminated. The reason for starting the private node after the public node was due in part to earlier experiments. We noticed that starting the public and private overlays at the same time took longer due to the state machines in the private node using exponential time back-offs of up to a minute when unable to retrieve the DHT keys.

For PlanetLab, we created several overlays with random distribution of private nodes. We start from a base network of 600 nodes on PlanetLab then had each node randomly decide to join the private overlay in order to obtain private overlays consisting of 5 to 600 members. We crawled the network to verify the amount of private overlay nodes. Then we began the experiment by connecting to the ring from the same network location 100 times using a different node ID each time, causing the node to join different peers and create an distribution of connection times. During this process, we measured the time it took for the public node and private nodes to become connected. To ensure that the PlanetLab public overlay was fully connected, we waited an hour prior to starting the experiments to provide enough time for it to reach steady state. This is a conservative value, in our experience with overlays we run on PlanetLab, pools of this size form a complete ring in the order of minutes.

For Simulation, we followed similar steps though we were able to tighten the exact amount of peers in both the public and private overlays. We began with a pool of 700 public nodes and 2 to 300 members in the private overlay. We then waited for the overlays to become fully connected, whereupon we waited 60 simulated minutes to ensure that the system was at steady state, i.e., trimming unnecessary connections and having a steady state machine. At which point, we added an additional public node with a matching private node. We took time snapshots of how long they took to become connected.

For the Network Modeler, we repeated the steps done for the simulation, though we were able to evaluate the timing of up to 100000 nodes, so we performed evaluations to compare with the PlanetLab and Simulation results as well as much larger network sizes 7. For timing, we reviewed the typical connection steps when connecting to the public overlay, querying the DHT, and then connecting to the private overlay using this process as the basis for our evaluation.

Our results, Figures 6, 7 are well correlated with near identical slopes. In all cases the time to become connected with the private overlay remains reasonable and scales logarithmically as network size grows. Connection times differ amongst the set, because the Simulator has let all the state machines waiting at maximum delay due to the lack of churn in the system, PlanetLab has limited overhead due to this delay, while the Network Modeler does not worry about state machine state, churn, nor bad connection attempts. An important result is that PtP security does not significantly add to time to joining the overlays, most likely due to the majority of the time being occupied by overlay routing.

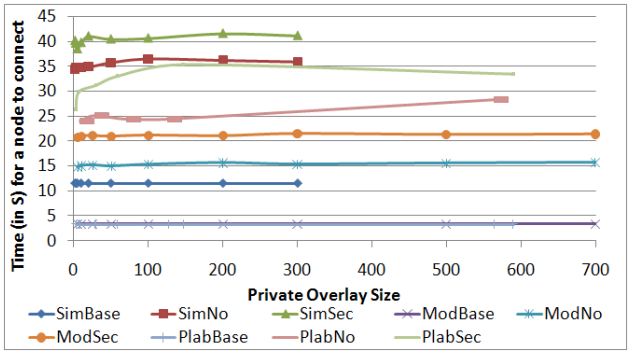

Fig. 6. The time it takes for a single node joining a public overlay and then a private overlay. Since the public overlay size is the same in all tests, we averaged all results together for the individual evaluations. The format for the legend is defined using the following convention: EnvironmentOverlay, where the environment is PlanetLab, Simulator, or (Network) Modeler and overlay is Base for public overlay, No for no security private overlay, and Sec for security enabled private overlay. ModBase is network modeler Public overlay connection time. SimSec is simulator security enabled private overlay.

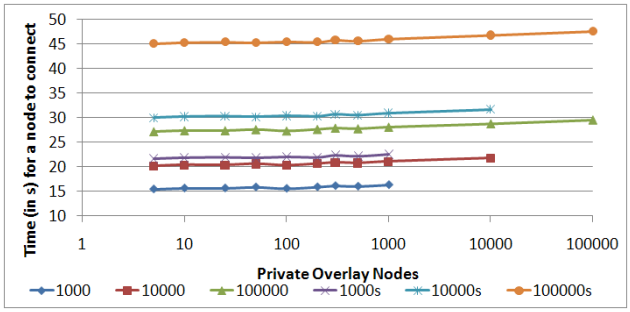

Fig. 7. The time it takes for a single node joining a public overlay and then a private overlay. The legend consists of number[s], where the number represents the number of peers in the public overlay and the optional "s" specifies whether the private overlay has security enabled.

D. Instantaneous Pool Creation

In this experiment, we determine the amount of time required to bootstrap a private overlay including their matching public overlay nodes using an existing network. We start with a network size of 200 public nodes and then add the nodes who will become part of the private overlay. Because PlanetLab is difficult to control in terms of attempting to start 200 public nodes simultaneously and ensuring that they do not restart mid-experiment, we ran this experiment using the simulator. Though we can state from previous experience, that using PlanetLab, we typically see a network form within minutes of the last host's overlay software starting.

In Figure 8, we present our results with and without security for various sized networks. The results present a slightly different picture than the previous experiment. In this case, there is a slightly logarithmic growth to the time it takes to complete an overlay. As in the previous test, the use of security has a small relative impact on the overall time to form the ring.

E. Measuring Bandwidth

The previous simulation experiments were in fact ran during the same test, but in order to ensure stability, we always wait

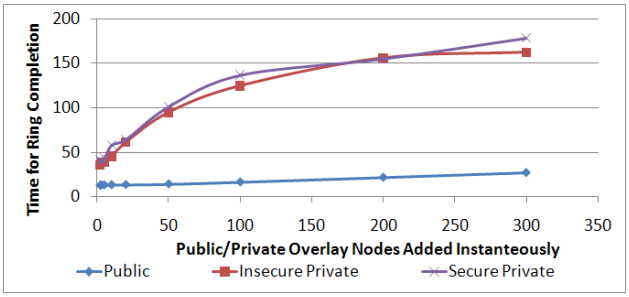

Fig. 8. The time it takes for various sized pools to instantaneously form on a public overlay with 200 nodes.

60 virtual minutes before proceeding to the next step. Reusing the simulation results changes no results and cuts down on the most expensive part of the procedure, the bootstrap phase. In this section, we present the network bandwidth used by each process, see Figure 9.

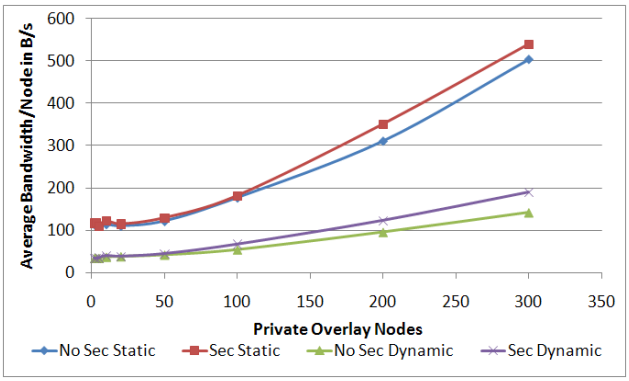

Fig. 9. Bandwidth used in a systems with and with security during steady state operations consisting of 200 public nodes and various sized paired public / private nodes. Those lines labeled "static" have DHT lists queried every 5 minutes whereas in "dynamic" queries are made on an exponential time scale starting at 30 seconds with a maximum of 60 minutes.

As shown in Figure 9, when cmparing the querying of the DHT using static and dynamic timers, the static timer's bandwidth is dominated by DHT queries. Since the system is at steady state, i.e., no new nodes in the system, only a single DHT query is made using dynamic timers. In both cases, the time to form a complete ring as performed in Section V-D is the same. The dynamic timer causes bandwidth to grow slower than a logarithmic pace. Given that the bandwidth used grows slowly, it appears that overlays in general use a negligible amounts of bandwidth and that even using security does not increase it significantly.

In regards to selecting a proper timeout, it can logically be surmised that if an existing public and private overlay are going through heavy churn, there will always be a base of nodes connected to each other. If because the DHT is currently fragmented, a new node forms a partitioned private overlay, the node should eventually and quickly find out about the original overlay and reform the split overlays. A dynamic timeout puts the weight for fixing ring partitions on the nodes that created them rather than the older more stable nodes. Though it would appear that ring partitions occur most frequently during massive joins, which is solved by using both static and dynamic timeouts while querying.

F. Evaluating Revocation Implementations

In Figure 10, we present the cost of revoking a peer in varying sized networks. To evaluate the cost, we estimate that the average size of a revocation is 300 bytes, which includes information such as the user's name, the group, time of revocation, and a signature from the CA's private key. We then evaluate the bandwidth and latency cost using a network modeler that reuses the same code base, i.e. routing tables and routing algorithms, as our structured P2P overlay software.

In progress...

Fig. 10. Broadcast revocation.

Some concluding remarks.

G. Multiplexing Sockets

If systems were to start making heavy use of the private overlay model, the most straightforward and simple approach would be to have many sockets and threads to constantly read from them. One option is to make a multisocket device and use select, but that requires a unique solution for each socket type. As we discussed early, we proposed an approach whereby an abstracted transport could easily be multiplexed without the need for additional threads, sockets, and other resources. We call this pathing and what we do is prepend each packet with a unique ID to differentiate the owner. So while it may reduce system resource consumption it may increase network consumption, so we test it to determine both. We present CPU, memory, and network in Figure 11.

In progress...

Fig. 11. Pathing evaluation.

Some concluding remarks.

VI. Conclusion

In this paper, we presented a novel architecture for deploying secure, private overlays in constrained environments through the use of public overlays. The public overlay at a minimum must provide a distributed data store, like a DHT, so that peers of the private system can rendezvous with each other. For constrained environments, we used NAT traversal techniques including STUN and TURN with both private and public overlays to support the relaying of packets. We evaluated the use of DTLS in our system and, to deal with excessive amounts of sockets, we introduced the notion of

pathing, which showed that system resource consumption can be reduced with negligible effect on network performance. Most importantly, we presented how a group infrastructure can be used as a user-friendly and intuitive mechanism to create and maintain private and secure overlays. For verification of usability, we show that the time to become connected to the private overlay occurs in less than 30 seconds and that the bandwidth required from each peer while idling was insignificant, on the order of 100 bytes per second. For future work, we recognize the constraint imposed on a user that requires them to host their own web server on the Internet and will investigate steps that will make it accessible via a VPN or directly through the overlay. Also, we envision applying this approach to social networks and to establish multicast groups as done in [9].

REFERENCES

[1] LogMeIn, Inc. (2009) Logmein. http://logmein.com.

[2] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. New York, NY, USA: ACM, 2007, pp. 205–220.

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15.

[4] M. Castro, M. Costa, and A. Rowstron, "Debunking some myths about structured and unstructured overlays," in *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, 2005.

[5] 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.

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

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

[8] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric," in *IPTPS '02*, 2002.

[9] 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

[10] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Security for structured peer-to-peer overlay networks," in *5th Symposium on Operating Systems Design and Implementaion (OSDI'02)*, December 2002.

[11] T. Dierks and E. Rescorla. (2008, August) RFC 5246the transport layer security (TLS) protocol.

[12] E. Rescorla and N. Modadugu. (2006, April) RFC 4347 datagram transport layer security.

[13] J. Liang, R. Kumar, and K. W. Ross, "The fasttrack overlay: a measurement study," vol. 50, no. 6. New York, NY, USA: Elsevier North-Holland, Inc., 2006, pp. 842–858.

[14] S. Guha and P. Francis, "Characterization and measurement of tcp traversal through nats and firewalls," in *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. Berkeley, CA, USA: USENIX Association, 2005, pp. 18–18.

[15] ludde, uau, The 8472, Parg, and Nolar. (2007, December) Message stream encryption. http://www.azureuswiki.com/index.php/Message_Stream_Encryption.

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

[17] LogMeIn, Inc. (2009) LogMeIn hamachi2 security.

[18] S. Limited. Skype. http://www.skype.com.

[19] D. Fabrice. (2005, November) Skype uncovered. http://www.ossir.org/windows/supports/2005/2005-11-07/EADS-CCR_Fabrice_Skype.pdf.

[20] S. Guha, N. Daswani, and R. Jain, "An experimental study of the skype peer-to-peer voip system," in *IPTPS'06*, 2006.

[21] (2005, October) RobotCA. http://www.wlug.org.nz/RobotCA.

[22] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "One ring to rule them all: Service discover and binding in structured peer-to-peer overlay networks," in *SIGOPS European Workshop*, Sep. 2002.

[23] Randpeer development team. (2007, May) Randpeer. http://www.randpeer.com.

[24] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," in *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*. London, UK: Springer-Verlag, 2001, pp. 14–29.

[25] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *18th ACM Symposium on Operating Systems Principles (SOSP'01)*, October 2001.

[26] 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.

[27] D. I. Wolinsky, L. Abraham, K. Lee, Y. Liu, J. Xu, P. O. Boykin, and R. Figueiredo, "On the design and implementation of structured P2P VPNs," in *TR-ACIS-09-003*, October 2009.

[28] J. R. Douceur, "The sybil attack," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 251–260.

[29] M. Jacob, "Design and implementation of secure trusted overlay networks," http://www.cs.princeton.edu/research/techreps/TR-865-09, August 2009.

[30] R. Dingledine, "Tor: anonymity online," https://www.torproject.org/, August 2009.

[31] P. Karn and W. Simpson, *RFC2522 - Photuris: Session-Key Management Protocol*, March 1999.

[32] J. Oikarinen and D. Reed. (1993, May) RFC 1459 internet relay chat protocol.

[33] C. Kalt. (2000, April) RFC 2810 internet relay chat: Architecture.

[34] P. O. Boykin and et al., "A symphony conducted by brunet," http://arxiv.org/abs/0709.4048, 2007.

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

[36] A. Ganguly, D. Wolinsky, P. Boykin, and R. Figueiredo, "Decentralized dynamic host configuration in wide-area overlays of virtual workstations," in *International Parallel and Distributed Processing Symposium*, March 2007.

[37] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "Wow: Self-organizing wide area overlay networks of virtual workstations," in *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, June 2006.

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

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

[40] F. Laub. (2009, October) OpenSSL.NET. http://openssl-net.sourceforge.net.

[41] 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.

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