Reflection, Rendezvous, and Relaying: Addressing the P2P Boostrap Problem for Small Overlay Networks

***Abstract*—Peer-to-Peer (P2P) overlays provide a framework for building distributed applications consisting of few to many resources with features including self-configuration, scalability, and resilience to node failures. Such systems have been successfully adopted in large-scale Internet services such as content delivery networks, file sharing, and data storage. The bootstrap problem, finding an existing peer in the overlay, remains a challenge to enabling these services for small-scale P2P systems. In large networks, the solution to the bootstrap problem has been the use of dedicated services, though creating and maintaining these systems requires expertise and resources, which constrain their usefulness and make them unappealing for small-scale systems. Decentralized, P2P systems can be useful in small-scale systems for privacy concerns as well as supporting applications that lack dedicated, centralized bootstrap servers.**

In this paper, we identify how existing services or overlays can be used to solve the bootstrap problem. The required components of our solution are a method for reflection in order to obtaining a publicly reachable address, so peers behind network address translators and firewalls can handle incoming connection requests; a rendezvous service for discovering remote peers, when the overlay lacks stable membership; and a communication relay to exchange connection information and communicate when direct communication is not feasible. We survey existing overlays and discuss their application towards these requirements. Based upon the results of the survey, we have implemented a prototype of the system using two of the overlays: XMPP overlays, such as Google Talk and Live Journal Talk, and Brunet, a structured overlay based upon Symphony. We present a qualitative overview of our prototype, quantitative evaluation of the prototype using simulation, and experiences from deployment. Uniquely differentiating our approach from similar techniques, we have implemented this system and verified its usefulness by deploying it on PlanetLab.

I. Introduction

While P2P overlays provide a scalable, resilient, and self-configuring platform for distributed applications, their adoption rate for use across the Internet has been slow outside of large-scale systems, such as data distribution and communication. General use of decentralized, P2P applications targeting homes and small/medium businesses (SMBs) has been limited in large part due to difficulty in decentralized discovery of P2P systems, the bootstrap problem, further inhibited by constrained network conditions due to firewalls and NATs (network address translators). While these environments could benefit from P2P, many of these users lack the resources or expertise necessary to bootstrap private P2P overlays particularly when the member state is unsteady and across wide-area network environments where a significant amount of or all peers may be unable to initiate direct communication with each other due to firewalls and NATs.

Examples of large-scale P2P systems include Skype, BitTorrent, and Gnutella. Skype is a voice over P2P system, whereas BitTorrent and Gnutella are file sharing programs. Bootstrapping in these system follows a common pattern: the overlay maintainers setup high availability systems for bootstrapping and bundle their connection information with the application for distribution. Upon starting the application, it queries these high availability peers and connects to other peers in the system. Alternatively, some services constantly crawl the network and place peer lists on dedicated web sites, a new peer wishing to join the network queries the web site and then attempts to connect to the peers on that list.

In smaller-scale systems, interest in P2P focuses more on the decentralized component. Users may desire to run an application at many distributed sites, but the application lacks dedicated central servers to provide discovery or rendezvous service for peers. Alternatively, dedicated, centralized P2P service providers, such as LogMeIn's Hamachi [1], a P2P VPN, may collect usage data, which the users may wish to remain private. Many applications fit into these roles, such as multiplayer games, especially those that never or no longer have dedicated online services; private data sharing; or even distributed file systems. Clearly, a small P2P system could be bootstrapped by one or more users of the system running on public IP addresses, distributing addresses through e-mail or phone, instructing his peers to connect to add that address to their P2P application, and then initiate bootstrapping; but but these types of situations are an exception and not the norm. Instead, We focus on the components that can make decentralized bootstrapping transparent to the user, so that user interaction with the P2P component of the application is minimal and intuitive.

The basic bootstrapping process can be broken down into two components finding a remote peer and then successively connecting to more peers. When a node bootstraps into a P2P system, it contacts bootstrap servers, until it successfully connects with one. Upon connecting with a bootstrap server, the two exchange information. The bootstrap server may inquire into the overlay for the best set of peers for the new peer and respond with that information or it may respond with its existing neighbors. At which point, the peer attempts to connect with those peers. This process continues aggressively until the peer arrives at the location in the overlay, where they

belong or the peer has satisfied the amount of connections required to maintain stability in the overlay. Afterward, the P2P logic arrives at a steady state and only reacts based upon churn, that is new incoming peers or outgoing peers.

Overlays with support for constrained peers, those behind NATs and firewalls, require a few more components. Before that, let us investigate why a peer may want to connect to a peer behind a NAT rather than communicate across the overlay. The user may be hosting a desirable file or a component of one. It may be the destination of a voice over P2P call. In both cases, the potential latency and bandwidth overheads of the overlay may make the performance of the service unacceptable to the user. If nothing else, then for privacy, so that overlay peers cannot easily profile peer communication habits.

There are many forms of NAT traversal, but many require a third party to assist in notifying both peers to simultaneously communicate with each other. Thus the overlay must have some means of relaying communication between peers. In some cases, peers will not be able to form direct connections, in which case, the overlay will need to provide relaying of arbitrary data in order for the peers to communicate.

Thus to instantiate a private version of these large-scale P2P systems would require the use of overlay-centric dedicated components and rely on out of band mechanisms to relay this information to others, such as an e-mail or a phone call. Even if this is feasible, this sort of interaction is undesirable, P2P systems should be self-discovering so that users need to do minimal amount of work to take advantage of them and adhoc systems stress this point. In addition, these approaches rely on centralized components, thus if they become unavailable, which is very likely since most users lack the expertise in configuring highly available systems, the system will not be accessable.

To address this, we propose the use of existing public overlays as a means to bootstrap into private overlays. There are many existing wide-spread systems with high availability, such as Skype, Gnutella, XMPP, and BitTorrent; by leveraging these systems, system integrators can easily enable users to seamlessly bootstrap their own private P2P systems. In the preceding paragraphs, we identified the components necessary for bootstrapping in a homogeneous system, now we expand them for environments to support the bootstrapping of a private overlay from a public overlay with consideration for network constrained peers. The public overlay must support the following mechanisms:

1) **Reflection** - Constrained peers must have some method of determining their Internet connection information, to share with other constrained peers to enable direct connectivity.
2) **Rendezvous** - Peers seeking members of a private overlay in a public overlay must be able to identify each other.
3) **Relaying** - Peers must be able to exchange arbitrary data to share connection information and to enable direct links across NATs when possible or, otherwise, as a relay.

This work motivates from the belief that what prevents use of P2P systems is due to lack the resources, technical knowledge, and lack of ability and desire to create and manage high availability bootstrap services. A public overlay can be used to transparently bootstrap a private overlay with minimal user interaction.

These requirements will be presented and verified in the context of two prototype implementations: a XMPP (Jabber) [2] and Brunet [3]. XMPP or Extensible Messaging and Presence Protocol based overlays are commonly used as chat portals, for example, GoogleTalk and Facebook Chat use the protocol. XMPP also supports a P2P overlay amongst servers forming a XMPP Federation allowing inter-domain communication amongst chat peers, so that users from one XMPP server can communicate with those using another. Brunet provides generic P2P abstractions as well as an implementation of the Symphony structured overlay. We present the architecture for these systems, the lessons learned in constructing and evaluating them, as well as provide quantitative analysis of peer connectivity to a private Brunet overlay.

The organization of this paper follows. Section II presents common P2P overlay technologies, unstructured and structured P2P systems, NAT challenges in P2P systems, and then an overview of existing bootstrapping techniques. In Section **??**, we present a survey of overlays, applying the requirements for private overlay bootstrapping to them, and then show in detail how they can be applied to Brunet and XMPP. Various applications that can be enabled by this architecture are described in Section IV. In Section **??**, we then perform a timing evaluation of bootstrapping overlays using our prototype running on PlanetLab. Finally, we conclude the paper with Section VI.

II. Background

A. Common P2P Systems

Large-scale P2P systems typically come in two flavors: unstructured and structured. Unstructured systems [4], [5] are generally constructed by peers attempting to maintain a certain amount of connections to other peers in the P2P system, whereas structured systems organize into well-defined topologies, such as tree,s 1-D rings, or hypercubes. Though unstructured systems are typically simpler to bootstrap and maintain, they rely on global knowledge, flooding, or stochastic techniques to search for information in an overlay and thus creating potential scalability constraints. Alternatively, structured systems [6], [7], [8], [9], [10]. have guaranteed search time typically with a lower bound of $O(\log N)$, though recent research has shown that these systems can be constructed to support $O(1)$ search time [11].

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. DHTs can be

used by peers of systems to coordinate allocation and discovery of resources, making them attractive for self-configuration in decentralized collaborative environments. DHTs provide the building blocks to form more complex distributed data stores as presented in Past [12] and Kosha [13]. While most publications of structured overlays focus on their ability to scale from thousands to millions of nodes, practical applications have employed them in the order of 10s to 100s of nodes. For example, Amazon's shopping cart runs on Dynamo [14] using a "couple of hundred of nodes" or less. Facebook provides an inbox search system using Cassandra [15] running on "600+ cores". As mentioned in the publications, this is due to the high availability and autonomic features provided by dedicated structured overlays.

Another subset found in various scale environments are "P2P" systems that are not fully decentralized, such as "P2P VPNs" like Hamachi [1], older systems like the original Napster, and tracker-based BitTorrent. These types of systems provide a rendezvous services for peers to discover each other to form direct, or P2P, connections with each other for the purpose of network connectivity or data sharing. BitTorrent differentiates itself by using the trackers as a gateway into the overlay, once inside, peers exchange connection information with each other directly relegating the tracker as a fall back. This approach has enabled BitTorrent to be modified to support trackerless torrents through using a DHT.

B. Network Address Translation Hampering P2P Systems

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 (approximately 4 billion) available. With the Earth's population of over 8 billion and each individual potentially having multiple devices that have Internet connectivity, the IPv4 limitation is becoming more and more apparent. To address this issue there have been two approaches: 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. The use of NATs complicates the bootstrapping of P2P systems as it prevents peers from simply exchanging addresses with each other to form connections, as the addresses may not be public. In addition, firewalls may prevent peers from receiving incoming connections. Thus while the eventual widespread use IPv6 will cause the demise of NATs, though this is not guaranteed, it does not deal with the issue of firewalls preventing P2P applications from communicating.

When a machine, *A*, behind a typical NAT, *B*, sends out a packet to an Internet host, *C*, the NAT device translates the packet so that it appears it is coming from the NAT device making the NAT device a router. When the the packet is sent from *A* to *C*, the source and destination are listed as $IP : port$ pairs, where the source and destination are $IP_A : Port_A$ and $IP_C : Port_C$, respectively. *A* forwards the packet to *B* who transforms the source from $IP_A : Port_A$ to $IP_B : Port_B$, where $Port_A$ may or may not be equal to $Port_B$. This creates a NAT mapping so that incoming packets from $IP_C : Port_C$ to $IP_B : Port_B$ are translated and forwarded to $IP_A : Port_A$.

There are a handful of recognized NAT devices as presented in [16], [17]. The following list focuses on the more prevalent types:

- **Full cone** - All requests from the same internal IP and port are mapped to a static external IP and port, thus any external host can communicate with the internal host once a mapping has been made.
- **Restricted cone** - Like a full cone, but it requires that the internal host has sent a message to the external host before the NAT will pass the packets.
- **Port restricted cone** - Like a restricted cone, but it requires that the internal host has sent the packet to the external hosts specific port, before the NAT will pass packets.
- **Symmetric** - Each source and destination pair have no relation, thus only a machine receiving a message from an internal host can send a message back.

Peers on cone NATs can easily be traversed, so long as a third-party assists in determining the port allocated by the NAT and another acts as a medium for the peers to exchange this information, this method is referred to as STUN [18]. Peers behind symmetric NATs cannot easily communicate with each other, since there is no relation between remote hosts and ports and local ports. Further complicating the matter is that there are various types of symmetric NATs, having behaviors similar to full, restricted, and port restricted cone NATs. [19] describes methods to traverse these NATs so long as there is a predictable pattern to port selection. The approaches that attempt to form direct connections between peers, hole punching, generally use UDP. UDP, unlike TCP, lacks state, thus traversal attempts do not need to be concerned with potentially replaying message or handling potential reject methods that may occur when attempting to perform TCP hole punching , though there is reasonable amount of work describing TCP NAT traversal such as [20]. TCP NAT traversal is complicated by stateful firewalls, or those that watch connections and connection attempts preventing messages from closed TCP channels from passing through the NAT. For situations in which both peers are behind NATs and firewalls that prevent inbound communication inhibiting NAT traversal, a third-party can act as a relay between the two, known as triangulation or TURN [21] (traversal using relay NAT).

These NAT traversal services in themselves only deal with a small portion of the bootstrap problem, reflection. That is, peers are able to obtain a public address for receiving incoming connections. These services provide no means for users to exchange these addresses with each other and cannot make any guarantees on how long the mappings will be valid. Thus a peer using an address that was not exchanged recently may find that there is no peer or a completely different service is now listening on the remote end. To address this issue many systems incorporate these NAT traversal libraries and use intermediaries to exchange addresses. In order for peers

to form direct connections, peers still need a mechanism to discover each other or rendezvous and then a relay pathway to exchange their network information.

An example of an application that uses these types of services combines with rendezvous and relay is Teredo [22]. Teredo enables all peers to have a globally identifiable IPv6 address. The IPv6 address maps to a specific Teredo gateway, client pair. When two peers using Teredo attempt to connect with each other, the Teredo servers exchange the clients IP information obtained using a STUN-like procedure. While this gives users a public address, it does not do so in a scalable or fault tolerant way. If the Teredo gateway goes offline or becomes saturated, the peers outgoing and incoming requests may be ignored.

C. Bootstrapping P2P Systems

As described in the introduction, the simple case of bootstrapping is limited to one peer attempting to find an active peer in the overlay in order for itself to become a member. The large-scale providers have resources not readily available to small-scale overlays. This section presents the existing techniques and those being developed and describes their application to small-scale systems.

When using dedicated bootstraps, a service provider hosts one or more bootstrap resources. Peers desiring to join the overlay query the bootstrap nodes, until a successful connection is made to one. The bootstrap server will then assist in connecting to other nodes in the P2P system. The bootstrap nodes are either packaged with the application at distribution time or through a meta data file, such as in BitTorrent. In small, ad hoc pools, draw backs to this approach are the same server would have to be used every time to bootstrap the system or users would have to reconfigure their software to connect to new bootstrap servers over time. A bootstrap server creates a single point of failure, especially when not being deployed using high availability techniques. This approach requires at least one peer to have a publicly accessible address.

Another commonly used approach for large-scale systems is the use of a host cache [23]. Clients post current connection information to dedicated web services, a host cache, that cache the connection information and URLs to other host caches. For small, ad hoc networks, a host cache acts no differently than a centralized rendezvous point, requiring that at least one peer has a publicly accessible address.

Research has shown that peers can use the locality properties of recent IP addresses in a large-scale P2P system can be used to make intelligent guesses about other peers in the P2P system in an approach called random probing [24], [25]. The results show for networks in the order of tens of thousands that a peer can find an active peer node within as few as 100 attempts and as many as 2,000 guesses. The application of this approach to small, ad hoc groups is challenging. If peers are behind NATs, the port mapping may change randomly or not be the same for multiple peers, further more, if peers are widely distributed and very small, this approach may have to query the majority of the IP address range to find a peer. Additionally, the approach assumes that all peers were on public addresses and that each used a constant port to host the P2P application. The results were not tested in real systems, but instead using overlay traces.

Rather than distribute an IP address, which points explicitly to some location in the Internet, a small P2P network can apply a name abstraction around one peer in the overlay using Dynamic DNS [26]. In this approach, peers share a common DNS entry, providing a common means to bootstrap the overlay. When the peers detect that the IP directed to by the host is offline, they use a random time back off algorithm to replace it. The application of this approach to small, ad hoc groups is actually quite nice, as the service could be distributed across multiple Dynamic DNS registrations. The significant drawback to the approach is that the dynamic DNS server could be attacked, since the login information must be shared amongst all peers in the overlay. Also the approach requires that at least one peer be publicly addressable and know that it is, if a non-publicly addressable peer updates the cache, it could delay or permanently prevent peers from creating a P2P system. The results were simulation based and did not determine how well a dynamic DNS handles rapid changing of name to IP mappings.

IP supports multicasting to groups interested in a common service. In the case of bootstrapping a P2P system [6], [25], all peers would be members of a specific group. When a new peer comes online, it would query this group for connection information and then connect to those that respond. The drawback to this approach is that it only works in environments connected by IP multicast, typically local area networks only, thus for a wide-area, distributed P2P system, this approach will most likely not work.

A large-scale structured overlay [27], [28] could enable peers to publish their information into a dedicated location for their service or application and then query that list to obtain a list of online peers. Peers could then come online, search for peers in their overlay, and then connect with them using their connection information. Since the service would be a large-scale system, it could easily be bootstrapped by a dedicated bootstrap or host caches. As it stands, the described works were position papers and the systems have not been fully fleshed out. The primary challenge in relationship to small, ad hoc networks is that it lacks details bootstrapping of peers behind NATs into overlays as it provides only a means for rendezvous and no reflection nor relaying.

III. Bootstrapping Private Overlays

As presented in the preceding sections, a solution to bootstrapping small P2P overlays must address several challenges, namely reflection, rendezvous, and relaying. In this section, we present a generic solution to this problem. At the basis of our solution is the use of a publicly available free-to-join public overlay. In order to support these features the public overlay must have mechanisms for peers to determine their network identity and maintain these mappings, reflection; search for other peers that are bootstrapping the same P2P

service, rendezvous; and send messages to peers through the overlay, relaying.

These are the minimum requirements to bootstrap a decentralized, P2P system when all peers are behind NATs. Without a reflection utility, two peers would be unable to communicate directly with each other, when they exchange connection information, they would only be able to share local networking information. If they are not on the same network, the information will be useless. Furthermore, if they are on the same network, then an approach using multicast would be sufficient. Thus assuming that the other two capabilities are available, rendezvous and relaying, peers would only be able to communicate through the overlay, which may introduce significant latency and bandwidth overheads.

A system, lacking the ability to exchange messages via the overlay or relaying, could limit the types of NATs peers could traverse. Assuming reflection and rendezvous, a peer could obtain connection information for another peer through an entry in the DHT, much like Tracker-less torrents, and connect to it so long as the peer is either on a public address, behind a cone NAT, or have a TURN connection, though a TURN connection would violate the expectation of the overlay lacking relay capabilities. In other all other NAT cases, a peer must initiate simultaneous connection attempts in order to form a direct connection. Furthermore, while TURN may provide relaying, it is not very scalable unless the features is built decentrally into the overlay.

Without rendezvous, peers would have no mechanisms to discover each other, in this case, random probing would be the only mechanism for peers to discover each other. If the peers are behind NATs, not even random probing would be sufficient for them to form connections with each other. Thus rendezvous is the most important component without it, peers would be unable to find each other.

Table I reviews various overlays, the majority of which are public free-to-join overlays, though some research only overlays are included. The overlays were chosen based upon their scale and availability. From this the most attractive two services are Brunet and XMPP. Brunet provides traditional P2P infrastructure, though lacks the large-scale and high availability features due to being rooted in an academic project. XMPP, on the other hand, is a non-traditional overlay and limits private overlays to being bootstrapped through friendship connections. While most users seeking a private overlay will do so with friends, this may not always be the case, consider video game players searching for people to play with online. On the other hand, XMPP is built on top of production systems thus has more applicability to real world usage.

A single public overlay does not need to provide all the components for bootstrapping the private overlay. For example, peers could use the the Limewire / Gnutella Kademlia DHT, Mojito, as a means to register for rendezvous. The peer could leave their globally unique XMPP identity in the DHT. Another peer interested in the same service would discover this identity and then could become friends through XMPP, an automatable process. Once the friendship has been formed, they can use XMPP as a means for relaying.

A. Bootstrapping Private Overlays Using Brunet

As mentioned earlier, Brunet, based upon the Symphony structured P2P overlay, is a freely available P2P library, supporting STUN and TURN-like NAT traversal. In this section, we describe how one could use a public Brunet overlay to bootstrap private overlays.

Brunet bootstraps using dedicated bootstrap services, though each peer can support a bootstrap list of 10,000 potential bootstrap nodes. With every connection Brunet makes, peers inform each other of their view of the remote peers network state, in other words, a form of passive reflection. When a peer attempts to connect to another, they route “ConnectToMe” messages across the overlay informing of the remote peer of the desire to connect. This message causes the peers to exchange their network information and both peers attempt to form connections simultaneously. This approach of bootstrapping handles all forms of NATs besides those that fall under symmetric and firewalls that prevent UDP. To address these types of NATs, Brunet also supports forming “Tunnels” [31], where peers exchange near neighbor information. If there exists overlap, then a connection will form. As presented in their work, the probability of overlap for overlay required connections, peers near each other in the overlay address space.

The approach for rendezvous in a DHT follows. All peers interested in a specific service or private overlay obtain a DHT key based upon a hashing of the service’ or private overlay’s name. Peers can then query this entry in the DHT to obtain a list of peers in the private overlay. Note, that Brunet’s DHT implementation supports a single key may have many values. Since DHTs are soft state, or lease systems, where data is released after a certain period of time. Peers must actively maintain their DHT entry.

The data stored in the DHT is the peers public overlay address. Doing this enables a peer to use the Brunet overlay for relaying, we call this a “Subring” transport. This requires that peers remain actively connected to the public overlay, so long as they are members of the private overlay, otherwise the peers could not use the service for relaying.

In order to take advantage of Brunet’s reflection service for the private overlay, we use a technique for multiplexing transports called “Pathing”. Pathing works by wrapping a multiplexing system around an underlying transport mechanism, such as UDP or TCP. As an example, when two peers desire to have communication with each other, they exchange end point information. In a system without Pathing, this would be IP addresses and port numbers. Pathing extends this to include a URI (universal resource identifier) path, like “udp://192.168.1.1:15222/path”. The Pathing transport recognizes the path as belonging to one of the various overlays and forwards it to the appropriate listener. By using the same sockets and bindings as the public overlay peer, a private overlay peer has access to the public overlays reflection

	Description	Reflection	Rendezvous	Relay
BitTorrent	Basic BitTorrent systems rely on a centralized tracker to provide the initial bootstrapping. At which point, the peers no longer require use of the tracker, but it can be used to monitor the state of the file distribution and ensure good distribution of file providers, seeds. BitTorrent specifies a protocol, though each client may support additional features not covered by the protocol.	In general, the current specification does not support NAT traversal, though future versions may potentially use UDP NAT traversal. At which point, BitTorrent may support a reflection service.		
Gnutella	Gnutella is a large-scale unstructured overlay, consisting of over a million active peers. Gnutella's is primarily used for file sharing. Skypes general make up consists of a couple thousand super, ultra, peers to provide reliability to the overlay. Gnutella is free-to-join and requires no registration to use.	Work in progress. Peers attempt to connect to a sharers resource, though a "Push" notification reverses this behavior. Thus a peer behind a NAT can share with a peer on a public address.	Peers can perform broadcast searches with TTL up to 2, when networks consist of millions of peers, small overlays will most likely not be able to discover each other.	Not explicitly, could potentially utilize ping messages to exchange messages.
Skype	Skype is a large-scale unstructured overlay, consisting of over a million active peers. Skype is primarily used for voice over P2P communication. Skype, like Gnutella, also has super peers, though the owners of Skype provides authentication and bootstrap servers. Though Skype is free-to-join, it requires registration to use.	Skype APIs provide no means for reflection.	Skype allows extensions through applications. It is possible for peers to broadcast queries to each other, transparently from the user, to determine if a given user has an application installed. Thus Skype does support rendezvous.	Skype applications are allowed to route messages via the Skype overlay, but because Skype lacks reflection, all communication must traverse the Skype overlay.
XMPP	XMPP consists of a federation of distributed servers. Peers must register an account with server, though registration can be done through XMPP APIs without user interaction. XMPP is not a traditional P2P system, though it has some P2P features. XMPP servers allow peers on distinct XMPP servers to communicate with each other. The links between servers are created based upon client demand. The links are trusted due to a certificate provided, when the server joins the XMPP Federation.	While not provided by all XMPP servers, there exist extensions for NAT traversal. GoogleTalk, for example, provides both STUN and TURN servers.	Similar to Skype, XMPP friends can broadcast queries to each other to find other peers using the same P2P service. THus XMPP supports rendezvous.	The XMPP specification allows peers to exchange abitrary out of band communication with each other. Most servers support this behavior, even when sent across the Federation. Thus XMPP supports relaying.
Kademlia [9]	There exists two popular Kademlia systems, one used by many BitTorrent systems and the other used by Gnutella, called Mojito. Kademlia implements an iterative structured overlays, where peers query each other directly when searching the overlay. Thus all resources of a Kademlia overlay must be publicly addressable.	Existing implementations of Kademlia does not support mechanims for peers to determine their network identity.	Peers can use the DHT as a rendezvous service, storing their connectivity information in the DHT at key location: $hash(SERVICE)$.	An iterative structured overlay has no support for relaying messages.
OpenDHT [29]	OpenDHT was until recently a freely available DHT running on PlanetLab, though it was recently decommissioned. OpenDHT is built using Bamboo, a Pastry-like protocol [6]. Pastry implements recursive routing, where peers in the overlay forward messages for each other from source to destination.	Existing implementations of Bamboo and Pastry do not support mechanims for peers to determine their network identity. Though this is ongoing work.	Peers can use the DHT as a rendezvous service, storing their connectivity information in the DHT at key location: $hash(SERVICE)$.	Because Pastry uses recursive routing, it can be used as a relay. Furthermore, extensions to Pastry have enabled explicit relays called virtual connections [30].
Brunet [3]	Brunet like OpenDHT is a freely available DHT running on PlanetLab, though still in active development. Brunet creates a Symphony [8] overlay using recursive routing.	Brunet supports inherent reflection services, when a peer forms a connection with a remote peer, the peers exchange their view of the relationship.	Peers can use the DHT as a rendezvous service, storing their connectivity information in the DHT at key location: $hash(SERVICE)$.	Like Pastry, Brunet supports recursive routing and even supports relays called tunnels [31].

TABLE I

service. This requirement means that in order to properly take advantage of Brunet reflection without significant code modification to Brunet, peers would need to reuse the Brunet's transport library.

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 overlay's DHT. A data inserter must constantly update the lease for the data object, otherwise the data will be removed, due to the soft state or leasing nature of DHT, whereupon (key, value) pairs are removed after a lease has expired.

B. Bootstrapping Private Overlays Using XMPP

The key features that make XMPP attractive are the distributed nature of the federation and the openness of the protocol. As of December 2009, there are over 70 active XMPP servers in the XMPP Federation [32]. These include GoogleTalk, Jabber.org, and Live Journal Talk. Even recently Facebook Chat deployed XMPP capabilities. Though unfortunately, the Facebook implementation is incomplete acting as a proxy into their private chat servers and thus providing only a limited subset of the XMPP features.

In XMPP, each user connects to a single XMPP service provider and maintains this single connection for the lifetime of the application. Messages between users are routed through this server and the XMPP Federation. Each user in the XMPP has a unique identifier of the form "username@domain". For two users in the same domain, the domain name makes for little consequence. Though when two users are in separate domains, the XMPP servers for those domains register with each other to proxy messages for the two users as well as other pairs that have an active XMPP relationship. When a user sends another user a message, it is encapsulated in XML, the destination is set to the remote peer, and it is sent to the server. The server relays the packet to the destination. Similary, when two peers are on separate domains, the server takes the domain portion of the user identifier and forwards it to that XMPP server, which will then relay the message to the destination peer.

To differentiate amongst multiple sessions, each instance of a XMPP client a user has logged in receives or sets a resource identifier, this is appended to the user identifier represented by "username@domain/resource". Thus two peers can explicitly state the destination XMPP client instance for outgoing messages.

Peer relationships are maintained by the server, though peers initiate them through a subscription mechanism. One peer requests to subscribe to the other, and the other can either accept or deny. The process is performed through an XMPP client using an XMPP query service. Once two peers have subscribed to each other, they are notified when an instance of each others XMPP client comes online. These are called presence notifications. The presence notification provides users the state of the XMPP client and its complete user identifier, including user name, domain, and resource.

The final baseline feature of XMPP is the XMPP query service. Peers can send to the server and each other "IQ" messages, which are control messages that are transparent to the user. An example of an IQ is the subscription messages used to establish a friendship. The use of IQs enables XMPP to be extensible, peers that support or implement an extension respond with a positive result and those that do not return a negative. Recent work by Google engineers has used the IQ to implement discovery of STUN and TURN servers called "Jingle" [33]. These features are provided free of charge through GoogleTalk.

Peers discover each other peers by using the resource portion of the user identifier. If the base portion of the resource string matches what the peer is seeking, it has found a peer seeking the same service. For example, common resource strings are "BitlBeeABC1234" or "P2P.CDEFABC". Thus a peer seeking a P2P service would be under the impression that the node "P2P.CDEFABC" is also. The node would then send an IQ to verify this and exchange network configuration information obtained through using their existing connection information for peers on a public address or the results from querying Jingle. If the peer was part of the network, it would respond back with its network configuration in kind, otherwise it would return an error for unsupported query type.

Once the peers have exchange this information, the two can attempt to perform hole punching if necessary. If the two are unsuccessful in forming a direct connection, they can either use a Jingle provided TURN server or route communication packets through XMPP IQs.

IV. Applications

In this section, we present applications and potential ways to configure them to use a private overlay. The applications we investigate include chat rooms, social networks, VPNs, and multicast. The key to all these applications is that users can easily host their own services and be discovered through the use of a free-to-join public overlay.

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 [34], 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 [35], 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

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. Using an approach similar to CAN [36], a virtual private overlay forms 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. Examples of such services include light weight multicast DNS / DNS-SD (service discovery), as well as audio and video streaming.

V. Evaluating Overlay Bootstrapping

In this section, we describe our prototype implementation, the environments used for evaluations, and evaluate bootstrapping time for single peers to join the overlay and for the creation of overlays.

A. Our Implementation

Our implementation makes heavy use of the transports library provided by Brunet [3]. We have extended the Brunet transports library to allow for sending messages between peers in one overlay to another overlay as well as an XMPP overlay. The XMPP library we used is called Jabber-Net. Each connection between peers is uniquely identified by employing socket like concepts, i.e., a pair of addresses and ports. The basic representation for this constitutes a pair of identifiers of the form "brunet://P2P_ID:PORT", where each peer has a unique ID and port associated for the local and remote entity. The XMPP implementation has a similar format: "xmpp://USERNAME@DOMAIN:PORT/RESOURCE", again one identifier for the local peer and one for the remote.

For evaluation purposes, we reused the Brunet structured overlay service to construct a private P2P overlay. Using a structured overlay may seem strange in such a small environment, though it actually applies quite cleanly. In small networks, several structured overlays act as $O(1)$ systems, this includes Brunet and Pastry. This value is configurable at run time, enabling systems to support form all-to-all connectivity with peers without user intervention. A structured overlay applies unique identifiers to each peer and in the case that two peers are not directly connected, greedy routing methods can be used to route messages to each other. Unlike unstructured systems, which may require that the peers know the structure of the entire system or use stochastic techniques to route packets between each other.

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 [37].

1) Simulations: Timing the decentralized bootstrapping of a P2P overlay from the view point of all participants is difficult to do using a real system. To evaluate the bootstrapping, we employ time-driven simulation that reuses the core components of Brunet and timing delays between peers and XMPP servers using the King data set [38]. Since the simulator models many aspects of the overlay, the simulation is limited to system sizes around 1,000.

2) PlanetLab: PlanetLab [37] 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 unscheduled 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.

C. Connecting to an Existing Overlay

This experiments provides light on the overall time required for a node to connect to a public overlay, XMPP or Brunet, and then for a subsequently to connect to the private overlay. Connected, for Brunet, implies that a node has 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 determines the time it takes to connect to a public overlay, find peers in the private overlay, and then connect with them. The process for bootstrapping has been described in Sections III-A and III-B for Brunet and XMPP, respectively. The test was performed using simulations and PlanetLab.

For PlanetLab, several overlays with random distribution of private nodes were created. When bootstrapping from Brunet,

the base public overlay consisted of over 600 nodes on distinct PlanetLab hosts, each node randomly decides whether or not to join the private overlay in order to obtain private overlays consisting of 5 to 600 members. The same nodes were also instructed to connect with the same XMPP server using the same login credentials. Crawling the overlay provided the actual amount of nodes in the private overlay. The setup was then evaluated by connecting to the public overlay, both through XMPP and Brunet, from the same physical location using various different XMPP servers and node IDs. This results in the peer obtaining results from different peers and creating a distribution of connection times. During this process, the time it took for the public node and private nodes to connect is measured. The experiments do not begin until an hour after the overlay has been deployed and verified to reach steady state. This is a conservative value; experience suggests that overlays run on PlanetLab with various sizes can form a complete ring in the order of minutes.

Simulation follows similar steps though with the peer count tightened to exact amount of peers in both the public and private overlays. The Brunet public pool consists of 700 nodes, the XMPP pool and the private pool 2 to 300 members. After waiting for the overlay to become fully connected and additional 60 simulated minutes is executed to ensure steady state. At which point, an additional public node with a matching private node are added to the system. The results measures are the time required for them to become connected.

D. Creating an Overlay

VI. Conclusion

In this paper,we have established the requirements for bootstrapping small-scale P2P overlays, reflection, rendezvous, and relaying. Reflection is required, so that peers behind NATs and firewalls can obtain a public address to share with remote peers. Rendezvous is a common problem, even for large-scale systems, peers must have a mechanism to find other peers connecting to are are in the same overlay. Finally, without relaying, peers behind NATs would be unable to exchange with each other their public addresses and perform NAT hole punching.

As we surveyed existing overlays, we found two systems that can easily satisfy the requirements, XMPP and Brunet. While Brunet can efficiently provide P2P services, it is an academic setup, XMPP, on the other hand, is production ready and distributed across many different providers, each offering interoperability. The only issue with using XMPP alone is that peers must already be friends in order to rendezvous. For future work, we will investigate how peers can leverage existing DHT deployments, such as Kad or Mojito, for rendezvous, form friendships automatically in XMPP, and continue the bootstrap process using XMPP. Our implementation successfully deals with the bootstrapping problem for small-scale overlays using decentralized technique.

References

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

[2]

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

[4]

[5]

[6] 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.

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

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

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

[10] S. Ratnasamy, P. Francis, S. Shenker, and M. Handley, "A scalable content-addressable network," in *In Proceedings of ACM SIGCOMM*, 2001.

[11]

[12] 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.

[13] A. R. Butt, T. A. Johnson, Y. Zheng, and Y. C. Hu, "Kosha: A peer-to-peer enhancement for the network file system," in *IEEE/ACM Supercomputing*, 2004.

[14] 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.

[15]

[16] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. (2003) Stun - simple traversal of user datagram protocol (udp) through network address translators (nats).

[17] P. Srisuresh, B. Ford, and D. Kegel, *RFC 5128 State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)*, March 2008.

[18]

[19] J. Rosenberg, "Interactive connectivity establishment (ICE): A protocol for network address translator (NAT) traversal for offer/answer protocols," http://tools.ietf.org/html/draft-ietf-mmusic-ice-19, October 2008.

[20] 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.

[21] J. Rosenberg, R. Mahy, and P. Matthews. (2009) "traversal using relays around nat (turn)". http://tools.ietf.org/html/draft-ietf-behave-turn-16.

[22]

[23] H. Damfpling. (2003) Gnutella web caching system. http://www.gnucleus.com/gwebcache/specs.html.

[24] C. GauthierDickey and C. Grothoff, "Bootstrapping of peer-to-peer networks," 2008.

[25] C. Cramer, K. Kutzner, and T. Fuhrmann, "Bootstrapping locality-aware p2p networks," in *in: The IEEE International Conference on Networks (ICON)*, 2004.

[26] M. Knoll, A. Wacker, G. Schiele, and T. Weis, "Bootstrapping in peer-to-peer systems," 2008.

[27] 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.

[28] M. Conrad and H.-J. Hof, "A generic, self-organizing, and distributed bootstrap service for peer-to-peer networks," in *International Workshop on Self-Organizing Systems (IWSOS)*, 2007.

[29] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "Opendht: a public dht service and its uses," in *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, 2005.

[30] A. Mislove, A. Post, A. Haeberlen, and P. Druschel, "Experiences in building and operating epost, a reliable peer-to-peer application," in *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, 2006.

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

[32]
[33]
[34] J. Oikarinen and D. Reed. (1993, May) RFC 1459 internet relay chat protocol.
[35] C. Kalt. (2000, April) RFC 2810 internet relay chat: Architecture.
[36] 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*, 2001.
[37] 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.
[38] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *SIGCOMM IMW '02*.