# A PEER-TO-PEER ARCHITECTURE FOR SOCIAL NETWORKING APPLICATIONS

By

PIERRE ST. JUSTE

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2014

I dedicate this to my mother, Alicita Atus, for sacrificing so much for my education.

ACKNOWLEDGMENTS

It is clear that I would not have attained a PhD without my advisor Dr. Renato J. Figueiredo. He decided to take a chance on me despite my own lack of self-confidence in my technical abilities. Under his tutelage, I have learned the value of collaboration and the importance of meeting the needs of the research community. His patience and steady guidance served as a fundamental pillar of my work allowing me the freedom to explore various avenues. His encouragement and positive outlook always kept me in good cheer throughout my PhD tenure. I will forever be in his gratitude and feel a great sense of pride and honor for serving as his student.

I am also extremely thankful for Dr. Jose Fortes' involvement in my degree. He has always been ready to provide much needed advice for my work and has steered me towards the path of clarity. He is always ready to lighten the mood with humor and made the ACIS lab an extremely welcoming environment. I am also thankful for Dr. P. Oscar Boykin's influence on my PhD work. He was always available and ready to explain complex concepts and pushed me to tackle technically challenging problems. I was continually inspired by his deep theoritical and practical knowledge of computing and the fundamental laws that govern the field. I have benefited tremendously from his candid nature and inclusive attitude. Moreover, I am also very grateful to have Dr. Xiaolin Li, Dr. Shigang Chen, and Dr. James Poe serve on my PhD committee. Their critical feedback has allowed me to fine tune my dissertation and meet the highest expectations of scholarly research. I would also like to thank all of my ACIS colleagues for being such great friends and collaborators.

I would also like to acknowledge the many funding agencies and organizations that have promoted and funded my PhD career. First, I would like to thank the McNair Scholars Program for introducing me to undergraduate research and for planting the seed of a doctorate degree in my mind. I would then like to thank the McKnight Fellowship foundation for funding 5 years of my PhD, I truly do not think I would have

obtained this degree without their financial support. Moreover, I would like to thank the SouthEast Alliance for Graduate Education and the Professoriate for supplementing my funding for multiple years during my PhD. My work has also been funded by various grants from the National Science Foundation through various awards IIP-0758596, CCF-0622106, 0910812, 0855031, and 1127965.

Finally, I would like to make a special acknowledgement to my family, specifically my mother Alicita Atus, my brothers, Robert St. Juste and Ericarl Rincher, and my step-father, Erick Rincher. They have always been supportive of my educational aspirations and have been my most ardent cheerleaders through this process. This doctorate degree would not mean anything without them and it also belongs to them as well. Thank you all and I am eternally grateful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

A PEER-TO-PEER ARCHITECTURE FOR SOCIAL NETWORKING APPLICATIONS

By

Pierre St. Juste

May 2014

Chair: Renato J. Figueiredo
Major: Electrical and Computer Engineering

The immense popularity of social networking services such as Facebook, Twitter, and Google Plus has increased the demand for social networking applications. Currently, the majority of social networking applications uses the common client-server model due to its design simplicity and controllability. However, centralized social networking applications need a substantial amount of costly infrastructure in order to service a large number of users. The centralized approach also suffers two other well-known shortcomings: lack of privacy and single point of failure.

This dissertation develops a peer-to-peer architecture which provides a trusted social messaging layer that developers can utilize to design and deploy their social networking applications. In this environment, social peers are able to share content directly with each other rather than through a centralized backend. The goal is to create an architecture that simplifies the development of social networking applications while leveraging the benefits of peer-to-peer networking. This approach provides users with self-sustaining social networking services that can run entirely on their own personal devices.

This dissertation makes several contributions in the area of virtual private networking. It is the first to integrate social networking (e.g. Facebook, Google Hangouts) with virtual networking to enable seamless establishment of encrypted end-to-end virtual IP links. This approach, dubbed SocialVPN, reuses XMPP overlays to

bootstrap trusted connections using both structured and unstructured P2P libraries. By exposing the Berkeley sockets API as the basis for communication, my research enables design simplicity for social application development that is unavailable in traditional P2P frameworks. Beyond IP connectivity, this work also demonstrates the design of a decentralized domain naming system called SocialDNS. This naming service is designed specifically for P2PVPNs, such as SocialVPN. SocialDNS allows nodes to set their own domain names, perform lookups on each other's DNS caches, and resolves domain name conflicts using a social ranking heuristic. Finally, there is Litter, a P2P microblogging system implemented to show the applicability of leveraging SocialVPN for the deployment of P2P social networking applications.

# CHAPTER 1
# INTRODUCTION

The immense popularity of social networking services such as Facebook, Twitter, and Google Plus has increased the demand for social networking applications. Hence, there is a tremendous amount of innovation aimed at incorporating social features in many applications such as photo-sharing, location-based services and search/recommendation systems [1]. Moreover, the widespread use of Internet-ready mobile devices has also increased the demand for social networking applications since they provide an array of sensors which enhance users‘ social experiences(e.g. GPS, camera). As a result, over the past few years, the Internet has transformed from a system for accessing information and services to a medium where users are constantly connected and instantaneously share their experiences with friends.

Currently, the majority of social networking applications uses the traditional client-server model due to its design simplicity and controllability. In this approach, social users communicate through a central entity which provides most of the necessary resources for processing, storage, and communication. However, centralized social networking applications need a substantial amount of costly infrastructure in order to service millions of users. Some may argue that cloud computing has minimized the infrastructure costs by giving developers the option of starting small and scaling up on demand for the necessary resources. However, even though cloud computing takes away the requirement of major initial investments in infrastructure, it still incurs high costs to support millions of users in a centralized fashion.

The centralized approach also suffers two well-known shortcomings besides infrastructure costs: lack of privacy and single point of failure. Centralized social networking services facilitate the sharing of content with friends and family through the use of massive datastores under the control of private entities. As a condition of using these social services, most users have to relinquish ownership of their content to

these third-party entities [2]. To offset infrastructure costs, social networking services (SNS) providers utilize user data at their own discretion to generate revenue through data mining and targeted advertisements. Such unspecified use of private user data may violate the implied trust that users attribute to these social networking services.

There has been growing unrest, with some of the most popular SNS providers, on the issue of user privacy and data sharing practices with unknown third parties without user consent. This presents a dilemma for both users and SNS providers because users desire a certain quality of service through these social networking services which is only possible if the providers are able to generate enough revenue to maintain the necessary infrastructure to satisfy user expectations. As a result, there is a constant struggle between privacy advocates and SNS providers in an attempt to strike the proper balance on the usage of private user information. There are also privacy concerns over permanent storage of user data. Various sources [3] have reported that SNS providers do not delete user information when expected (e.g. deleting a photo or account) for a significant amount of time. For some users, the thought of having all of their interactions permanently stored in a corporate database may hinder or limit their usage of these social networking services.

Finally, the single point of failure vulnerability is the most prominent disadvantages of the client-server model, thus centralized social networking services automatically inherit this problem. Due to their centralized nature, blocking these online social networking services is common and there are a multitude of well-known types of denial of service (DoS) attacks to achieve this [4]. An example of this weakness is when governments in parts of the world deny users access to certain social networking services through DNS filtering and IP blacklisting [5]. There have also been instances where rogue hackers have targeted centralized social networking services with distributed denial of service attacks (DDoS) [6]. Additionally, high profile user accounts

on centralized social networking services have been hacked through man-in-the-middle attacks [7].

As the amount of private user information accumulated by SNS providers increases, their infrastructures become more appealing targets to malicious actors who are constantly attempting to breach these systems and steal private user data. As data mining techniques become more advanced and widely used, user data is becoming a gold mine and serves a crucial role as part of a company's assets. Therefore, ensuring trusted data access and privacy are top priorities and become major technical challenges. Consequently, adopting the typical client-server model for deploying social networking applications may not only be costly, but it requires extreme care to ensure data privacy while maintaining the ability to generate revenue. It also necessitates the foresight to anticipate the shortcomings of the single point of failure problem, as well as the expertise to secure the infrastructure to avoid data breaches of valuable private user information. This dissertation promotes an alternative peer-to-peer paradigm for designing social networking applications which deals with the issue of a single point of failure while providing more flexibility in dealing with privacy concerns.

## 1.1 Background and Related Works

Peer-to-peer (P2P) systems are known for their independence from centralized backends. File-sharing P2P systems such as Gnutella, eMule, FreeNet, and Bittorent have been around for about a decade and exist completely on end-user resources [8]. When applied to social networking services, the two most common P2P related models are DHT-based and Darknet-based designs. The first type is the structured distributed hash table (DHT) method. A structured DHT is a well-researched distributed storage system which provides simple put/get primitives. The resources used in this distributed storage are usually made up of hundreds to millions of geographically dispersed machines belonging to the users of the system. This approach scales well because each user contributes resources to the overall infrastructure. By replacing the

centralized datastore with a distributed alternative, developers can design functionally equivalent social networking services which sidestep both the cost of maintaining an infrastructure as well as the single point of failure.

However, the deployment of a full-fledged social networking service such as Facebook on a DHT introduces a host of issues which are nonexistent in the centralized version. Security is by far the most complex and hardest to handle. Since developers rely on untrustworthy machines connected to the Internet, in other words, completely out of their control, it makes it very difficult to trust these end-user nodes. This has been addressed by several works [9] but their main drawback is the need to manage cryptographic keys.

There is also the issue of heterogeneity. Developers can expect a certain guarantee on the performance of nodes housed in a data center, but such expectations are unrealistic for end-user machines. Therefore, the nodes that make up the infrastructure for a DHT range from powerful multicore workstations with stable high bandwidth connections to low-power single-core thin clients with intermittent Internet connections. Adapting the system to such variability while maintaining an acceptable level of quality of service (QoS) for an online social network can be tremendously challenging. A few projects attempt this model [10, 11]; however, they suffer from the bootstrap problem because they fail to attract enough users to accumulate the necessary resources to sustain the service.

The second peer-to-peer design for a social networking application is the darknet model. In this model, peers only connect to trusted friends through encrypted connections. Instead of sharing content through a centralized or distributed datastore, they contact each other directly and synchronize updates through gossip/epidemic algorithms [12]. By limiting content dissemination only to trusted peers, the privacy issues are much simpler to handle because users have full access control over their data.

Assuming that the cryptographic key management problem is handled, the other major hurdle for this approach is ensuring high data availability, in other words, minimizing propagation latency. Here is an example of the problem: Alice wants to share a photo of her new car with her ten friends, but only five of them are online when she shares the photo through the peer-to-peer social networking application. By the time her remaining five friends come online, Alice is offline. Eventually, they all get the update from Alice only when her online time overlaps with each of her friends' online time. As a result, some of Alice's friends may have to wait days before they notice Alice's new car. Such a delay hinders the usage of this type of P2P social networking application. There are currently existing efforts to develop social networking services using this model [13], but they have not been successfully deployed.

The challenges of building peer-to-peer social networking services have created the impetus for this dissertation. The goal is to identity the common hurdles that preclude the development and successful deployment of social networking applications and provide an architecture which mitigates these pitfalls. By addressing issues such as trusted connectivity among peers, automatic cryptographic key management, a well-known application programming interface (API), and multi-hop social routing this work aims at creating a strong foundation that will drastically lower the barriers of designing distributed social networking applications.

## 1.2 Design Models

Several programming paradigms exist for building social networking applications. This chapter covers some of the more common approaches by summarizing their general concepts along with their most salient benefits and shortcomings. The goal is to provide brief context and basis for comparison to the proposed P2P approach. This will allow for deeper understanding of the direction of this dissertation. This section describes four well-known techniques for designing social networking applications: the

integrated client-server model, the decoupled client-server model, the decentralized federated model, and the peer-to-peer model.

### 1.2.1 Integrated Client-Server Model

The integrated client-server model is the most widely used method for creating a social networking application (SNA). In this model, SNA developers rely primarily on components under their administrative control to provide the key functions needed by social applications. Therefore, SNA developers use their own servers — although these servers may be hosted by a third-party cloud service provider — to manage and store social relationships, along with necessary resources to enable the content sharing such as updates, messages, photos and other forms of media. Since the service is totally independent, developers have to provide the capability for users to find and create social connections within the system. Well-known examples of this model are Facebook and Twitter. For instance, the usual first step after creating a Facebook account is to find friends who already have accounts and send them *friend requests*. Upon acceptance, a social connection is created on the system which is stored in Facebook's database. Hence, Facebook provides all of the necessary functionalities to create and manage these social connections.

The main advantage of this approach is total autonomy over the design of the social networking service. Developers are free to enable or disable any feature at their own discretion. The additional benefit of this approach is the complete ownership of user data representing their social relationships, activities, and interactions. This information is not only a boon for sociological research but also for targeted advertising firms and even governmental agencies. Design simplicity is another advantage of the integrated client-server model.

There are also disadvantages to this integrated client-server model. The first common problem of any social networking service is the bootstrap problem. The popularity of online social networking has created dozens of competing social networking

services. Therefore, generating the critical mass necessary to have a self-sustaining system can be quite challenging. Most users are comfortable with only using a few popular social networking services (e.g. Facebook, Twitter) and are not compelled to create and manage social relationships across dozens of social networking services. Thus, a common key requirement for the success of a social networking service is the recruitment of large number of users. With enough users, the developers can attract additional investment capital, and generate revenue through advertisements.

The second pitfall is the single point of failure issue which plagues all client-server-based systems. For example, a network administrator can easily deny access to a centralized social networking service either through DNS filtering or IP blacklisting. Moreover, since centralized services have well-known endpoints, malicious agents can target the endpoint with DoS attacks. These issues make securing the centralized backbone of a social networking service a top priority because it serves as an attractive target for attackers.

### 1.2.2 Decoupled Client-Server model

SNA developers typically want to sidestep the task of requiring users to create and manage social connections on their own system. They prefer reusing existing relationships already identified through an integrated social networking service such as Facebook. Independent social networking services, described in the previous section, typically expose an application programming interface (API) enabling third-party developers access to their social networking database [14]. In this scenario, users have to manage social relationships only on the integrated social service and the social relationships are automatically updated through other social networking applications which are linked to their accounts. For example, Alice wants to play *Words with Friends*, an online crossword puzzle, which uses the Facebook API to identify her friends and send invites to play the game. As Alice adds or removes friends from her Facebook account, *Words with Friends* automatically retrieves the updated list of friends through

the Facebook API. In this decoupled client-server model, the developers still use their own infrastructure to provide the functionalities specific to their application. Social networking applications based on this model primarily depend on the integrated SNS for managing friends and other messaging features.

Utilizing this model benefits both developers and users due to reduced social relationship management. From the developers' perspective, they do not have to dedicate extra resources for managing social relationships. Moreover, they sidestep the bootstrap problem. By leveraging the popularity of existing integrated social networking services, the developers significantly reduce the barrier to the adoption of their social applications. Users also benefit from a management standpoint because they are not required to create new accounts and repeat the steps of finding and creating social links on every SNA. Users are able to reuse their existing social relationships seamlessly.

The decoupled client-server approach however suffers from a few weaknesses. The tradeoff for not managing social relationships and using another system's social networking features is ultimately total reliance on the API and policies. Therefore, changes to the API and policies can have serious detrimental effects on the social networking applications [15]. Before choosing to depend on a particular social networking API, developers need some assurance from the social networking system that the interface is stable and changes will be slow and incremental. This decoupled model also suffers the same weaknesses of any client-server model of privacy and single point of failure. Although not all of the components reside in the same administrative domain, it is still fairly simple to block access to a decoupled client-server social application through DNS filtering and IP blacklisting. Also, the privacy issue may be exacerbated because they are now two entities involved in accessing and managing a user's social activities and interactions. There has been a history of these types of third-party social applications causing data leaks of private user information [16].

### 1.2.3 Decentralized Federated Model

This model is a departure from the two previous approaches because it does not require any centralized infrastructure from the developers. It reuses existing, decentralized, and federated messaging systems. Examples of such systems are the Extensible Messaging and Presence Protocol (XMPP), Skype, or IRC. Since XMPP is an open standard and is adopted by most major messaging system, it is used as the primary example. XMPP is the main instant messaging technology used on the Internet and is supported by Google, Facebook, AOL, Windows, and Skype. Although its initial use was primarily for instant messaging, its extensibility has allowed it to support social networking services. For example, there is currently an experimental extension draft to support a microblogging service on top of the XMPP protocol [17].

The XMPP system works as a federation similar to the email system. Users are free to choose the XMPP provider of their choice — as long as these providers are part of the same federation managed by the XMPP foundation — users are able to communicate with each other. The XMPP protocol require less infrastructure than the integrated and decoupled client-server approaches. XMPP servers mainly handle the following functions: registration of users, storage of buddylists, and routing of XML messages from source to destination. The servers can also support additional functionalities but they are not part of the core requirements.

By relying on the XMPP layer for social relationship management as well as message routing, SNA developers can design services that leverage the XMPP layer for peer discovery and messaging while utilizing users' resources to enable additional features. This is a key advantage for developers because they can essentially deploy a social networking application without having to provide centralized resources to handle computation, storage, and communication among peers. Also, by reusing existing social connections available through these federated social services, this approach minimizes social relationship management. For example, if a microblogging service is

built on top of XMPP, the user will automatically send updates to peers in her XMPP buddylist without having to recreate existing social connections. Additionally, since the messaging layer is composed of a multitude of XMPP servers communicating with each other, the single point of failure issue is mitigated. For instance, Alice@jabberA.com, Bob@jabberB.com, and Carol@jabberC.com are friends; if Bob's XMPP server jabberB.com is brought down due to a failure, Alice@jabberA.com can still communicate with Carol@jabberC.com since XMPP servers jabberA.com and jabberC.com are still operational. It requires more resources to attack all of the XMPP providers simultaneously in order to bring down the entire social application.

There are also drawbacks to this approach in terms of reliability and quality of service because of its dependency on unreliable end-user machines. End-user computing environments are highly heterogeneous ranging from low-power single core thin-clients, to multi-core workstations running many different operating systems with varying degrees of bandwidth. In the client-server model, developers control the resources which support the essential components of a social networking application and can therefore allocate the adequate amount of CPU, storage and bandwidth for the desired quality of service. Moreover, system administrators continuously monitor these servers to ensure that these services run with minimum failure.

In the decentralized, federated model, the performance of the system is highly dependent to the stability of end-user resources. Consider Alice is using a social application which allows her to share photos with her 100 friends. In the client-server model, Alice simply uploads her 1 MB photo to the centralized SNS using only 1 MB of her upload bandwidth. Once uploaded, her 100 friends can view the photo which utilizes 100 MB of the centralized system's bandwidth. The developers ensure that there is ample bandwidth to serve Alice's 100 friends.

In the federated model, the XMPP servers simply route data without storing it. Consequently, Alice is responsible for serving her 1 MB photo to her 100 friends using

100 MB of her upload bandwidth. With a typical ISP connection with an average upload bandwidth of a 1MB/sec or less, it takes at least 100 seconds to send the photo to all of Alice's friends in comparison to the centralized approach which typically takes less than one second. There are Bittorent-like swarm-based techniques that can ameliorate the situation. However, the performance of the social application becomes more costly to the clients due to reliance on their restricted resources.

Another issue to overcome with this model is content availability. For instance, Alice shares the photo when only half of her friends are online, and by the time the other half of her friends come online, Alice may be offline. In most cases, if Alice's friends are online, these updates could be propagated through friends or friends of friends. However, if Alice is not very popular or some of her friends do not know each other, it may take a long time before all of Alice's friends are able to see her shared photos. These types of social networking applications through the XMPP protocol are possible but they are not very popular due to their poor quality of service and low content availability.

### 1.2.4 Traditional Peer-to-Peer Model

Finally, there are decentralized social networking applications based on the peer-to-peer model. In this model, the users provide all of the resources necessary to support the social networking application. Unlike the previous three models, which require some publicly-hosted infrastructure, the peer-to-peer model has zero infrastructure cost. In the peer-to-peer model, users directly connect with their trusted friends to share content without a middleman. Developers either write their own P2P protocol or by utilize existing peer-to-peer technologies. One example is the FreePastry library developed by the authors of the Pastry P2P protocol [18]. This library handles message routing and NAT traversal among peers. By building social networking applications on top of the peer-to-peer library, social networking developers can ensure that the system is self-sustaining without the need for costly infrastructure.

In the previous federated model, users are required to trust third-party XMPP service providers, and assume that they will not monitor or record their social interactions. Moreover, in the previous models, censorship is possible since a government can request specific content to be taken down, or demand that XMPP servers filter certain content. In the peer-to-peer model, it is harder to censor the system since social peers are connecting directly to each other without the use of any hosted services.

The peer-to-peer model also has many challenges to overcome. Due to its strict independence from centralized services, it has to overcome two bootstrap problems. The first bootstrap problem is the one which is necessary for the peer-to-peer system to be self-sustaining. There needs to be enough bootstrap nodes who are openly reachable through public IP addresses enabling other nodes behind NATs and firewalls to connect through NAT traversal and relaying. Bootstrapping a peer-to-peer system is not simple; therefore, most peer-to-peer systems depend on a few well-known set of bootstrap nodes which serve a critical role in allowing new nodes to join the peer-to-peer system. Disabling these well-known bootstrap nodes can disable the P2P network and make it impossible for nodes to join the network.

The second bootstrap problem deals with peer discovery and social relationship management. In systems like Pastry [18] and FreeNet [19] users have to email each other cryptographic information to create encrypted end-to-end connections. Because this system is independent, users have to manage their own social connections through the peer-to-peer social networking application. Such an approach can serve as a major barrier to widespread adoption because it requires multiple steps before users can deem the social networking application usable.

Aside from the bootstrap problems inherent to P2P systems, this model also inherits the shortcoming of the decentralized, federated model. In terms of quality of service, the social networking application is at the mercy of end-user resources. If most users have a stable, high-speed Internet connection with a powerful multi-core machine with high

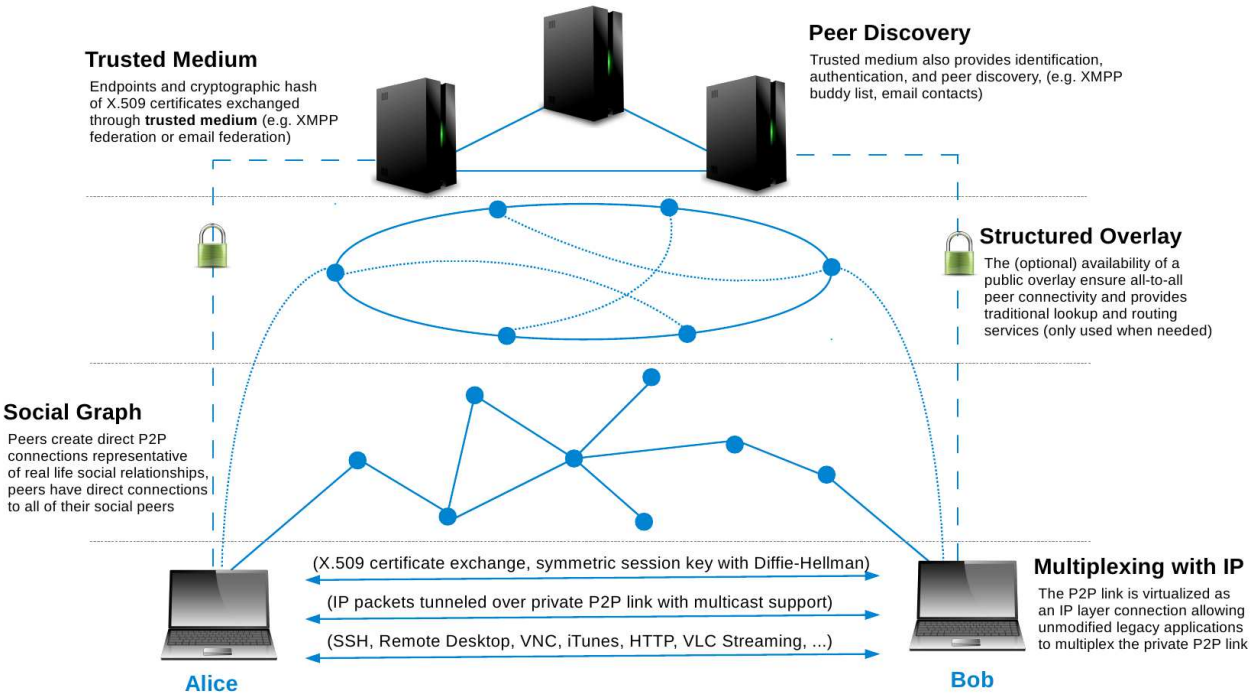



Figure 1-1. Summary of P2PVPN Alternative

session times, then the quality of service may be close to that of a centralized social networking applications. However, if the end-user nodes have unstable connections, then ensuring any acceptable quality of service becomes quite challenging. In terms of data availability, the same circumstances of the previous decentralized approach still applies where users may have to wait a long time before receiving all of the social updates from friends. Hence, although it is possible to use existing P2P technologies to design a social networking application, the increased complexity impedes the successful deployment of such a service.

## 1.3 The Proposed Peer-to-Peer VPN Alternative

This dissertation advocates a different way of building social networking applications through the use of peer-to-peer virtual private networks (P2PVPN). A P2PVPN creates a virtual network which tunnels IP packets directly to peers through encrypted P2P tunnels (see Figure 1-1). There are currently various P2PVPN solutions available [20–22].

Here is a quick description of how a P2PVPN can facilitate in the design of a social networking application.

The main goal of a P2PVPN is to provide users private IP connectivity to each other. Since most common users access the Internet from behind NATs and firewalls, there typically is not a well-known IP address that applications can use to connect with a user. A P2PVPN therefore makes it possible for users to contact each other directly using private IP addresses. P2PVPNs also do not depend on a VPN gateway server like traditional VPNs thus making it more difficult to carry man-in-the-middle attacks.

By leveraging the private IP connections that become available among users, SNA developers can design services that communicate directly over IP. For example, in a centralized social networking application, Alice contacts Bob by telling the system to post a message on Bob's inbox. With the presence of a P2PVPN, Alice is able to send Bob a message by using Bob's private virtual IP address to create a TCP/IP connection and send that message directly to Bob's machine. Direct TCP/IP connectivity to users give developers more flexibility in the design of their social networking applications and can optimize the performance of their applications.

In the XMPP model, applications can only communicate through the use of XML messages. Since XMPP only supports text-based messaging, it is inefficient to share other forms of media such as photo through the XMPP layer. Developers are thus required to convert binary data to a string representation (e.g. base64 encoding) before sending it through the XMPP layer. This process incurs some overhead in the social networking applications. On the other hand, through a P2PVPN, developers get an IP connection and are free to send raw data and can leverage existing techniques for optimizing message transmission over TCP/IP connections.

This dissertation develops a peer-to-peer framework which provides a trusted social messaging layer that developers can utilize to design and deploy their social networking applications. In this environment, social peers are able to share content

directly with each other rather than through a centralized backend. Developers can thus create a service which has essentially zero infrastructure cost. The goal is to create an architecture that makes it simple for developers to create social networking applications that can leverage the advantages of peer-to-peer networking and provides users with self-sustaining services that run entirely on their own resources.

Although there are existing peer-to-peer libraries that developers can integrate into their social applications, they typically fall short in terms of ease of use, social peer discovery, and cryptographic key management. The main novelty of this work lies in the fact that the proposed P2P architecture handles these complex issues. First, by providing a well-known interface through IP virtualization, developers can quickly utilize this architecture based on well-established network programming techniques (i.e. Berkeley sockets). The proposed architecture also lets users import existing social relationships from popular social networking infrastructures such as Facebook and Google Plus. This feature is crucial because it enables zero-configuration social peer discovery and social connection management. Finally, by providing simple mechanisms for cryptographic key management, users can rest assured that their social interactions are private. The result is a self-configuring, low-management, and secure platform which facilitates the deployment of decentralized social networking applications.

### 1.3.1 Cryptographic Key Exchanges

Ensuring privacy is a common hindrance to both centralized and decentralized social networking services. By leveraging existing trusted social networking infrastructures such as the XMPP federation [23], this architecture makes it possible for peers to reuse their existing Facebook or Google Plus accounts. It also automatically discovers their social peers without requiring users to manually recreate these social connections. The system works by using the encrypted connection to the XMPP servers to exchange self-signed X.509 certificates. The asymmetric public keys within the certificates serve as the basis to bootstrap authenticated, peer-to-peer connection among social peers

with end-to-end encryption. This approach thus securely makes cryptographic key management seamless to users which is a very important step in addressing the privacy issue.

### 1.3.2 Private IP Networking

Another dissuading factor in designing peer-to-peer social networking applications is the learning curve of integrating with rapidly changing, custom, peer-to-peer APIs. As peer-to-peer libraries attempt to provide more features and serve as a generic platform, their APIs tend to grow in complexity to support these new capabilities. Moreover, the programming framework can also be a limiting factor. For example, if a library is written in Java, developers are ultimately locked into that programming language because creating bindings for another programming language further complicates the situation.

My research, dubbed SocialVPN, uses network virtualization as the means to provide secure communication among social peers. More specifically, instead of using a peer-to-peer library API for communication in the network, developers are presented with the abstraction of a private network which supports the IP protocol and assigns private addresses to endpoints automatically. When a social networking application sends a packet to a predefined private IP address of a social peer, it is sent to a virtual networking interface and the operating system hands the packet over to the P2P architecture which tunnels it over a direct, encrypted, peer-to-peer connection. This mechanism creates a virtual private network consisting of only trusted social peers. This abstraction enables useful features such as the ability to reach all friends through the use of multicast or broadcast IP addresses. Through virtualization, developers are not required to learn new APIs and can apply well-known network programming techniques which greatly reduces the coding complexity of social networking applications.

Another key advantage to providing a virtual network made up of social peers is the natural support for unmodified legacy applications. Consider a user that enables iTunes media sharing over the social virtual private network; automatically, iTunes

becomes a social application that makes it possible for friends to share media with each other. The benefit is that, although social peers are geographically dispersed, no modification is required to legacy applications to extend their reach from the local area network to the wide area. By running a service over the network, it becomes privately accessible to social peers and forms the foundation to effortlessly enable social networking applications.

### 1.3.3 User Defined Naming

Private IP connectivity provides a basis for social P2P communication. However, these IP addresses are dynamic therefore hindering their use as static endpoints to services. Private IP addresses are dynamic due to the address space limitations of IPv4, along with the use of DHCP to temporarily map endpoints to IPv4 addresses. A similar situation exists in the SocialVPN architecture where IP addresses assigned to social peers have to be reassigned to avoid IP conflicts. Consider Alice's friend Bob is assigned 172.31.23.41 as his private IP address, then reassigned a different IP address of 172.31.115.98. Therefore, discovering Bob's virtual IP address may become a burdensome task. The advent of IPv6 further increases the need to utilize domain names rather IP addresses due to their extended length. Since the Internet provides both IP connectivity as well as a domain naming system (DNS) for location transparency and user-friendliness, providing only IP addresses through the proposed architecture is not a complete solution. Consequently, a naming service is also required to help users refer to their friends' services using easy to remember names.

To address this issue, the proposed architecture provides a distributed domain naming system called SocialDNS. By leveraging the underlying P2P messaging system, SocialDNS gives users the ability to map social peers to canonical domain names. The global DNS system, currently used on the Internet, does not allow users control over the domain names associated with their machines. Most user machines do not have a domain name mapping. If users require domain names, that usually involves a network

administrator or a complicated process of running a DNS server (e.g. BIND) along with complex configurations. SocialDNS lets users define their own domain names without the need for a network administrator or a DNS server.

There are existing decentralized DNS systems for the local area network such as Apple's multicast DNS [24], or the Windows Internet Naming Service (WINS) [25]. However, these solutions are not well suited for P2PVPNs because they assume all-to-all connectivity amongst all peers in the private network which is not always the case in P2PVPN environments. By transparently mapping social peers to domain names, developers can use predictable endpoints such as *bob.sdns* which resolves to Bob's current virtual private IP address, without requiring developers to take extra unnecessary steps to discover Bob's private IP address.

### 1.4 Case Study: Designing a Peer-to-Peer Microblogging Service

Microblogging is a simple, yet popular social networking service which has become a primary mechanism for publishing information in a concise and practical manner. The research community has been discussing the need for a distributed microblogging service in the wake of the frequent Twitter blockages that have occurred in the Middle East due to the Arab Spring [5]. Various solutions have been proposed but only through simulations without a real implementation [12] while others still depend on a centralized component [26]. Therefore, this motivated the creation of a lightweight, resilient, yet user-friendly service, called Litter.

By leveraging the secure IP connectivity, multicast support and certificate management of the proposed P2P architecture, developing Litter involved primarily writing the code which handles the network communication and the local database storage. Designing a peer-to-peer microblogging service which leveraged the various aspects of this architecture served as a prototype to understand its limitations. The private social connections and the key management aided the discovery and identification of friends in a simpler fashion than traditional P2P approaches. The use

of standard IP communication made the Litter design more portable by allowing its deployment on both the local area network and the virtual private network without any code changes. Overall, Litter provides a solution to the problem of creating a practical distributed microblogging service and the availability of the P2P architecture provides more flexibility and portability to the design.

## 1.5 Contributions

The primary contribution of this dissertation is a comprehensive, peer-to-peer architecture that focuses on providing a private networking environment that simplifies the development and deployment of distributed social networking applications. This is accomplished by providing these novel capabilities:

- **Social peer discovery and cryptographic key management.** Through the use of social networking APIs, the proposed architecture automatically discovers existing social peers, exchanges and manages cryptographic information to bootstrap secure networking connections among friends.
- **IP Connectivity with zero-configuration IP addressing and resolution.** The use of network virtualization combined with a dynamic IP translation mechanism makes it possible for the system to assign virtual IPv4 addresses to social peers without global synchronization which assures no subnet conflicts and addresses the IPv4 address shortage problem.
- **Decentralized domain naming system.** A social DNS system lets users choose short names which also incorporates social context and uniqueness, along with a ranking mechanism for handling DNS conflicts.

## 1.6 Outline

The rest of the dissertation is as follows: Chapter 2 explains the techniques that handle cryptographic management and social peer discovery through existing social networking systems It also discusses the virtualization techniques of SocialVPN. Chapter 3 discusses the support of unstructured overlays including enabling deployments for mobile devices. Chapter 4 details the design of SocialDNS which is a P2P social naming system complementing the global domain naming system. Chapter 5 describes

the implementation of a peer-to-peer microblogging service as a case study for the framework. The remaining challenges and future work are detailed in chapter 6.

# CHAPTER 2
# SECURING IP CONNECTIONS THROUGH STRUCTURED OVERLAYS

Despite the widespread usage of social networks, direct social connectivity, in other words, private peer-to-peer (P2P) connectivity between friends, is still a major hurdle for social networking systems due to the heterogeneous structure and constraints of the Internet such as limited IPv4 address space, Network Address Translators (NATs) [27] and private network configurations. Furthermore, most of these social networking applications require centralized administration to authenticate, control, secure and mediate interactions amongst peers. These centralized architectures necessitate complex support and management to meet continuous demand from users, as well as significant infrastructure investment in order to robustly handle millions of users.

Peer-to-peer (P2P) systems, on the other hand, are architected to achieve scalability and availability in a distributed fashion without relying on centralized servers. However, they lack a comparable framework for authentication, access control, and security which are commonly available in centralized infrastructures. This work thus advocates an approach where social networking infrastructures are utilized to bootstrap secure social connections over P2P overlay networks. The synergy of these two models produces a scalable, secure and reliable system capable of supporting larger numbers of users with significantly less infrastructure support and management complexity. This combined system can be perceived in two ways: either as enhancing social networking capabilities with P2P connectivity or evolving P2P overlays into secure, social networking platforms.

This chapter presents the concept of a social virtual private network (SocialVPN), an approach aimed at bridging the gap between social and overlay networking. At the heart of SocialVPN lies the ability to automatically establish *direct peer-to-peer Layer 3 network links* as a result of connections or friendships established through social networking infrastructures. This work is mainly concerned with a social networking

infrastructure as a system which allows for the discovery of peers and the binding cryptographic public certificates (e.g. X.509 certificates or X.509 certificate fingerprints) to these identities. Hence, a social networking infrastructure can range from a full-fledged online social network such as Facebook to an encrypted Google Hangout chat session, and even a PGP-signed email exchanges amongst peers.

Assuming the aforementioned infrastructures are trusted, users can seamlessly leverage social networking relationships to establish private network-layer channels. In this context, social networking is key to enable a decentralized system where users are able to maintain their individual trust relationships with friendly interfaces. Therefore, P2P overlay networking becomes the essential messaging substrate enabling the formation and maintenance of direct private tunnels without any centralized backend. SocialVPN lets users communicate securely using existing TCP/IP applications such as desktop sharing (e.g. VNC and RDP), shared file folders (e.g. SMB and NFS), audio/video-conferencing, and multi-user games in the presence of NATs and firewalls and without modification, a feature which is not currently supported by social networking infrastructures. By using private IP addresses to create direct TCP/IP connections to social peers, social networking application developers are not restricted to any particular protocol or message format. Such a capability empowers developers with access to all of the tools necessary to optimize their design for their specific social applications.

Towards this goal, the proposed overlay architecture is novel in the following respects: 1) it enables automatic assignment and dynamic translation of virtual private IPv4 addresses to hosts in a non-intrusive manner which avoids IP conflicts with current network deployments and requires no user configuration; 2) it supports automatic exchange and discovery of peer credentials (e.g. X.509 certificates) through multiple social networking infrastructures, allowing end-to-end authentication and encryption of all communication among trusted peers. In this approach, the only configuration required from users is the creation and management of *social connections*;

the configuration and maintenance of *IP network connections* is self-managing and completely transparent to users. The SocialVPN connections are thus accomplished without burdening users with the complex, error-prone configuration typically required to bring up public key and network tunneling infrastructures in VPNs.

This chapter also describes and evaluates a prototype implementation based on the IPOP [28] virtual network, different social networking infrastructures (including the Facebook platform, a Web back-end based on the Drupal content management system, an PGP-signed email exchanges by peers), and an PKI-based IP packet encryption security systems using X.509 certificates. Experiments in both local- and wide-area networks are used to demonstrate the capabilities and measure the performance of these social IP links. The experiments are conducted in realistic, large-scale wide-area environments, including over 500 SocialVPN routers distributed across five continents over the PlanetLab infrastructure, and over 100 SocialVPN virtual endpoints deployed dynamically over the Amazon Elastic Cloud (EC2) infrastructure.

## 2.1 Background and Related Works

Current online social networks do not provide decentralized communication amongst peers. As a result, Web-based social networks (WBSNs) aggregate user content in centrally-administered domains requiring users to relinquish control of potentially private data. Although this centralization facilitates peer discovery and content sharing, it precludes point-to-point application access such as multi-media streaming, interactive multiplayer games, and desktop sharing because users can only interact through the predefined constructs supported by the WBSNs. Furthermore, WBSNs require massive infrastructures that can store huge amounts of user content (e.g. hundreds of millions of photos and videos) and serve the high numbers of simultaneous user requests, further increasing the cost and maintenance of these social systems [29]. SocialVPN's ability to securely connect peers through IP-layer network links in a seamless and automatic manner enables new methods of sharing

as well as application communication currently not supported by social networks. Consequently, this peer connectivity drastically reduces the centralized infrastructure requirements since it no longer needs to mediate every user interaction. For example, peers can share files directly with one another while controlling their content since no intermediate storage is needed on social networking backends.

### 2.1.1 Key Management

Efficient distributed peer discovery and cryptographic key management is still an open problem in P2P systems. Various key management models exist; some are based on a central key distribution center, others based on IP-multicast or distributed hash-tables [30, 31]. This work presents several key distribution schemes from a simple PKI-based model where all binding security credentials (i.e. X.509 certificates) are retrieved from a single trusted source to a web of trust model when users can perform certificate exchanges over PGP-signed emails. From a usability standpoint, a centralized social backend greatly simplifies the peer discovery and key management process, but it is not a requirement. (Section 2.2.2).

Cryptographic key management in peer-to-peer networks is a requirement for forming authenticated end-to-end communication channels and access control [31, 32]. Previous work on security frameworks for collaborative computing provides a usage-control model which incorporates a hybrid model based on attribute acquisition and event-updates to control decisions for resource access [33]. Domingo-Ferrer [34] proposes the use of public-key cryptography in social networks to reduce the overhead of managing private relationships which alleviates the requirements of the social networking infrastructure. Previous works have not studied the possibilities of using a trusted social backend to simplify the peer-to-peer key management problem.

In the case of peer-to-peer communication, two approaches are possible. One approach is to exchange shared session secret keys through the centralized backend, but this requires communication with the backend to refresh session keys periodically.

The other option is to use the secure backend channel for a one-time public key exchange (more specifically X.509 certificate) between the peers. Once the peers have obtained each other's cryptographic public keys, they can directly negotiate and generate symmetric session keys for encrypted peer-to-peer communication. The latter approach is desirable from bandwidth and scalability standpoints, because it does not necessitate constant interaction with a centralized backend, and it is the approach taken in this dissertation.

### 2.1.2 Support for Legacy Applications

Overlay networks have been very successful in delivering content in a P2P fashion – such as Skype for VoIP or BitTorent for file sharing. However, overlay networks typically connect users at the application layer, also effectively precluding a variety of legacy applications from being used – for instance, one cannot stream iTunes music or play a multiplayer game through Skype. There are a large number of software packages based on the Internet Protocol (IP), through the use of the Berkeley sockets API. These existing and legacy software can be used with a new networking paradigm, such as one based on social connections. By means of virtual networking, it is possible to produce a virtual local area network enabling the reuse of legacy TCP/IP based software.

### 2.1.3 IPv4 Connectivity and Address Limitations

Internet users today are regularly behind network address translation (NAT) devices using dynamically assigned IP addresses. Even though software exists to enable file transfer, conferencing, and collaboration, these tools typically require dedicated central servers in order to handle cases where users are not directly addressable by one another. The proposed overlay network with NAT traversal support and self-optimizing direct connections can drastically simplify the development and deployment of social networking applications.

Virtual private networks require unique IP addresses for users. Public IPv4 addresses are scarce and often not available to end users, and private IPv4 addresses

are not sufficient to enable each user of a typical social network to obtain a unique virtual IP address. For instance, the number of Facebook users (currently over 900 million) is larger than the number of IP addresses available in the 10.0.0.0 class A private address space [29]. Despite the fact that the IPv4 address space support billions of unique addresses, most users will only require direct communication with a few tens or hundreds of users [29]. Furthermore, the structure of these implicit communication networks is highly clustered [35] based on the small world phenomenon. These facts about social communication patterns are leveraged to enable a dynamic IP assignment and translation scheme which avoids IPv4 address space conflicts with existing network configurations while maintaining support for legacy TCP/IP applications (Section 2.2.5).

### 2.1.4 Existing Virtual Private Networks

Virtual private networks have been the popular choice for enabling wide-area access to resource in private organizational networks. Popular software products such as OpenVPN [36] are great tools that provide private IP layer tunneling in wide-area communication among peers. This approach suffers from two major drawbacks: a single point of failure, and error-prone configuration. In this client-server model, all encrypted IP tunneling is conducted through a publicly addressed VPN server which makes this approach highly centralized. Also, the complexities of configuration and key distribution make this approach unappealing when forming ad-hoc virtual organizations for wide-area communication. Various virtual networking projects for grid and cluster computing environments exist (such as VINE [37], VNET [38], or VIOLIN [39]), but they utilize routing tables that are managed and not self-configuring, making it difficult to establish private connections among peers in an ad-hoc P2P fashion.

Recently, P2P VPNs such as Hamachi [20], N2N [40], or ELA [41] have become popular peer-to-peer alternatives to centralized VPNs. In Hamachi, backend STUN-like servers are used to enable NAT traversal and establish direct peer-to-peer connections among users; these servers also generate session keys for encryption and administer

group access control. This control in Hamachi is done through shared secret keys where individual users do not initially control who has access to the network. This approach differs from SocialVPN architecture in two ways: 1) SocialVPN's NAT traversal is not centralized because it uses existing nodes in the overlay to perform UDP hole punching for direct peer-to-peer connectivity, and 2) nodes negotiate their own session keys and manage access to their network locally without a centralized backend. N2N, on the other hand, does not require a centralized backend but it provides layer 2 networking and uses a different peer-to-peer network than the ring-structured overlay in SocialVPN [42]. The N2N peer-to-peer network uses supernodes that act as relay nodes for edge nodes that cannot communicate directly and they also store edge node information. N2N also requires more configuration; for example, automatic DHCP configuration is not available and pre-shared keys are used for link encryption. SocialVPN requires virtually no configuration and provides automatic DHCP and DNS services. ELA is also a peer-to-peer VPN with DHCP support; however it uses a different, hierarchical P2P overlay while SocialVPN uses a flat ring-based P2P overlay. Also, these VPNs do not integrate with online social infrastructures which is one of the SocialVPN's key strengths. SocialVPN also uses a dynamic IP translation mechanism which requires no global knowledge and coordination among the nodes for IP allocation. Such coordination is usually required to avoid IP collisions since all endpoints in said VPNs commonly use the same IP address space.

## 2.2 Design

The SocialVPN architecture contains the following components: 1) connection privacy through encrypted end-to-end authenticated channels, 2) peer discovery and certificate exchange through a trusted social backend, 3) direct peer connectivity and legacy application support through IP-over-P2P overlay networking, and 4) dynamic IP allocation and translation which avoids network conflicts.

### 2.2.1 Structured Overlay Routing

A fundamental cornerstone of the proposed P2P architecture is its use of a structured P2P overlay called Brunet [43]. Brunet is a Symphony-based structured overlay that organizes all nodes in a ring structure with shortcut connections to guarantee logarithmic routing. The main benefit of using the Brunet overlay is to enable all-to-all connectivity amongst all the peers in the system, even those behind network address translators (NATs) and firewalls. Therefore, through the use of a 160-bit P2P address, a node is guaranteed a path to any other node in the peer-to-peer network by simply using a greedy routing algorithm to route a message towards its destination. The Symphony-based structured overlay also guarantees efficient routing that takes on average $\log(\mathrm{N})$ hops to reach its destination. Brunet also provides a distributed hashtable (DHT) service which this work utilizes as a global storage system for X.509 certificates; thus eliminating the need to depend on a centralized backend to provide that storage capability.

Nodes, that are part of the structured overlay, can also create direct P2P connections with social peers once they discover their P2P addresses. Brunet also supports encrypted direct connections if both sides have the appropriate X.509 certificates; however, the management of these X.509 certificates are not defined. Adding the capability to manage the X.509 certificates by leveraging existing social networking infrastructures is one of the key contributions of this dissertation.

### 2.2.2 Peer Discovery and Certificate Exchange

Peer discovery is the process of obtaining a list of unique peer identifiers (e.g. email addresses) that represent social peers. Any system that can generate such a list can thus be considered a social networking infrastructure, provided that the users of the system trust the infrastructure. Once the social peers are identified, the X.509 certificates bound to the peer identities are obtained through a trusted backend, which may or may not be part of the same social networking infrastructure. In general,

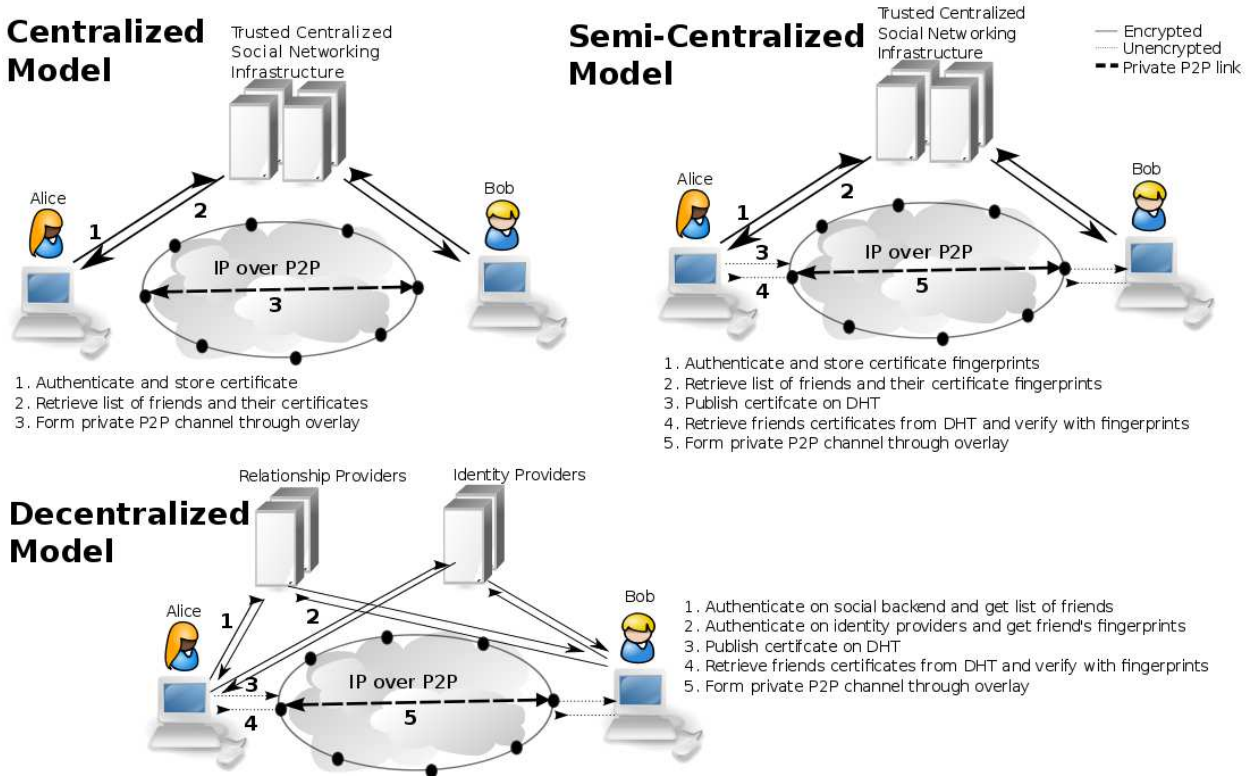



Figure 2-1. Certificate Exchange Models. Top left: In the centralized model, the list of friends and their certificates are obtained from a single social networking backend. Top right: In the semi-centralized model, the list of friends and their certificate fingerprints are obtained from one or more centralized social networking backends; certificates are stored and retrieved from a distributed data store (DHT). Bottom: In the decentralized model, the list of friends can be obtained from multiple social backends; certificate fingerprints are retrieved from multiple identity providers, and certificates are exchanged over the DHT. Peers themselves can verify identities locally, e.g. following on a PGP-based web of trust model. In the last two cases, the certificate fingerprints are used to verify the integrity of DHT-acquired certificates.

the proposed architecture requires the following capabilities from social networking infrastructures: 1) the ability to query for a list of peers, 2) the ability to retrieve binding information about the peer, 3) and the ability to exchange public X.509 certificates among peers. These features are available either through a social networking API (e.g. Facebook API), or through an XMPP library which is supported by most of the major social networking systems such as Facebook, Google Plus, Skype, MSN, and so forth.

The security of the proposed architecture is based on the public key infrastructure (PKI) model. PKI systems are well-understood, and robust implementations are available; however, the management of keys is complex, error-prone and overwhelming to an end user who is not familiar with security concepts. By querying an online social networking infrastructure, the proposed system handles the management and distribution of X.509 certificates transparently in a secure manner. Following is the description of three possible methods of performing peer discovery and certificate exchange through three different types of online social networking infrastructures.

**Centralized Model.** In this method, peer certificates are automatically exchanged through the trusted centralized social backend to form direct, private connections. The centralized model is the easiest to manage and design, and this is illustrated with a Facebook prototype. Facebook is a Web-based social network (WBSN), and it provides peers with the ability to create relationships, bind data to their identity through user profiles, and share trusted content with their peers. Through the Facebook Graph API, peers are able to authenticate themselves, store the X.509 certificate on the Facebook datastore, and they are also able to retrieve X.509 certificates of their social peers as well (see Figure 2-1). As a result, the Facebook Platform can be leveraged to emulate some of the functions of a typical PKI allowing for the secure acquisition of X.509 certificates. The X.509 certificate contains two fields of binding information the peer identifier (i.e. email) and the P2P address for overlay routing. That information serves a crucial role because it provides the identity as well as the endpoint to bootstrap a secure connection to the user.

**Semi-centralized Model.** This hybrid method is similar to the centralized model except for certificate storage and the possibility of supporting multiple social networking infrastructure back-ends. The X.509 certificate's fingerprints are securely stored on the social networking back-end. Peers then share public security credentials (i.e. X.509 certificates) through the distributed-hash-table (DHT) available in P2P overlay (see

Figure 2-1). Using the DHT for storage and only storing the fingerprints in the trusted backend allows for a more scalable design because less storage is required from the backend. Assuming the X.509 certificate fingerprints are obtained through trusted means, the certificate exchange can take place over the DHT as long as the public X.509 certificate's integrity can be confirmed. Once the certificates are safely acquired and trusted on both ends through the verification of their fingerprints, they are utilized to form secure connections between peers.

**Decentralized Model.** In the decentralized model, independent components are utilized together to serve as an online social networking infrastructure. For example, a list of contacts uniquely identified by email addresses; therefore, a trusted address book can provide peer discovery. There is also the concept of an identity provider [44], which may be any infrastructure that provides profile information based on a unique identifier; this serves to meet the second criterion. There are currently many identity providers on the Internet such as WordPress, Paypal, Verisign, just to name a few. Through secured public profiles, users can publish their certificate fingerprints thus allowing other peers to obtain trusted fingerprints for these identity providers. With the trusted fingerprints, the DHT-stored certificates become verifiable allowing for the bootstrap of secure connections.

The web of trust security model can also be integrated in this architecture to provide another example of a decentralized certificate exchange model. In this scenario, a PGP key is used to sign a self-signed X.509 certificate. The reason for the double signing is because the PKI-based and PGP-based models are not compatible with each other. This hybrid approach allows the proposed framework to take advantage of existing PKI-based security infrastructures and to leverage the decentralized benefits of the web of trust PGP design.

To reiterate, the first requirement for creating trusted P2P connections is the trusted exchange of X.509 certificates. Users can easily email each other their X.509

certificates, as long as the email is PGP-signed. If the receiver also has the sender's PGP key as part of their PGP keyring, they can easily verify the X.509 certificate integrity with PGP. Once the email has been verified, the user can input the trusted X.509 certificate to the SocialVPN system manually through the user interface. In this model, the user becomes the source for peer discovery, the trusted peer's PGP key contains the identity binding information, and the public key exchange is done through the email messaging system. As in the other two cases, once public key certificates are exchanged, the process of generating and exchanging symmetric keys for encrypted communication is transparent and is accomplished by P2P messaging among the endpoints, e.g. by following a protocol such as Diffie-Hellman. This same model has been applied to an instant messaging system through the use of the XMPP library, where users can share each other's certificates through an encrypted XMPP chat session. The common theme here is the establishment of trusted, and not necessarily private, out-of-band communication channels for one-time exchange of binding security credentials.

### 2.2.3 Establishing Private Connections

Connection privacy is a necessity for secure wide-area collaboration. While various options are possible to initiate a secure P2P connection, this approach is based on public-key cryptography which supports the public key infrastructure (PKI) model. In doing so, one can reuse existing tools for processing X.509 certificates, and seamlessly integrate with datagram security technology which is widely used such as IPSec or DTLS [45]. This architecture builds upon an IPSec-like datagram security model which works as follows: 1) peers exchange X.509 certificates through a trusted medium, 2) the retrieved X.509 certificates serve as the trust anchors (list of trusted CA certificates) for bootstrapping secure connections, 3) asymmetric public keys help generate symmetric session keys to encrypt the traffic between endpoints, using well-known protocols and implementations such as Diffie-Hellman key exchange. This model thus allows

for the creation of authenticated, private, end-to-end connections which protects from third-party intrusions.

In order to bootstrap a secure P2P connection, self-signed X.509 certificates are exchanged. These X.509 certificates are self-signed and must be acquired through trusted means because each peer ultimately becomes the certificate authority (CA) of their own certificate. In all PKI-based infrastructures, a basic requirement is that the CA certificates serve as the trust anchors and must be acquired securely. On first run, an X.509 certificate is generated containing a peer's security credentials such as name, email, country, organizational unit, organization, and the P2P address (in the SubjectAltName field). The P2P address is a unique 160-bit identifier uniformly assigned to each node on the peer-to-peer overlay; it forms the basis for the peer-to-peer structure and message routing. Social peers are added to the system by retrieving their X.509 certificates from a trusted source. To create a social link, both peers need to add each other's certificate, if *peer1* adds *peer2's* certificate, the secure network link will not form until *peer2* adds *peer1's* certificate. The reciprocity ensures both endpoints have acknowledged the social relationship by explicitly adding each other's certificate; this process is automated when the proposed architecture connects to an online social network. The information in the certificate, specifically email, P2P address and public key, are used by this architecture to create an encrypted P2P link bound to each social peer, meaning the X.509 certificate contains all of the credentials necessary for the secure connection.

### 2.2.4 IP Connectivity through a Structured Overlay

The majority of networking applications are based on the TCP/IP protocol; hence, these applications cannot easily leverage P2P overlay networks for connectivity, since P2P libraries are usually incorporated at the application layer. Allowing unmodified applications network connectivity through a P2P overlay is a valuable feature that can expedite the design and deployment of wide-area social systems. Additionally, social

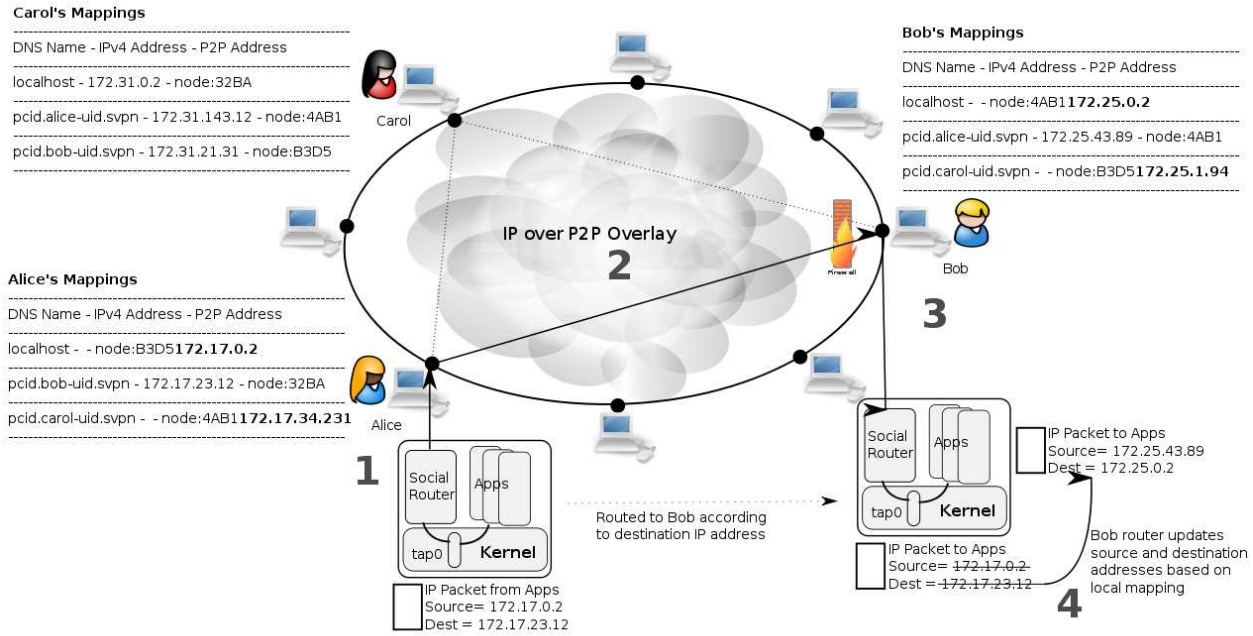



Figure 2-2. SocialVPN Architecture. After the peer discovery and certificate exchange through a social backend, peers form direct, encrypted channels where applications can communicate through TCP/IP. 1) Applications send IP packets to tap0 virtual NIC through the kernel and the user-level social router captures the IP packets. 2) Social router checks the destination IP which maps to friend Bob, encrypts the IP packet with the symmetric key (previously established for the IP-over-P2P tunnel after public key exchange) and sends the encrypted IP packet over the P2P tunnel through Bob's firewall. 3) Bob's social router receives the IP packet. It looks up in its local database information about the source (including Alice's symmetric key and virtual IP address); it then decrypts it, and updates the source and destination IP addresses according the Bob's local mapping. 4) Bob's router sends the translated IP packet to the applications through the kernel-based virtual NIC.

networking application developers can interface with the system through the Berkeley Sockets API instead of some obscure peer-to-peer API. While the IP-over-P2P (IPOP) layer in the SocialVPN architecture can conceivably be designed on top of various P2P substrates, the discussion to follow is based on the Brunet overlay [28]. A capability of Brunet that is key in the SocialVPN context is decentralized NAT traversal; these and other aspects of the Brunet overlay are described in previous works [28, 42].

Virtual network interfaces can be utilized to capture and inject IP packets from and to a host operating system kernel [46]. These captured packets are then tunneled as normal application P2P traffic through an optimized structured overlay [42]. On the sending side, the virtual IP routers are able to retrieve IP packets from legacy applications through the virtual network interface, and send these IP packets to the appropriate P2P node on the overlay. On the receiving end, the router receives an IP packet from the P2P network, and injects it back into the host operating system; thus enabling Layer 3 level communication between applications. As shown in Figure 2-2, the SocialVPN virtual IP routers maintain a mapping of IP-to-P2P addresses where a P2P address is bound to a particular peer based on information obtained from the peers' credentials exchange (i.e. X.509 certificates containing P2P addresses). Therefore, the routers possess a list of P2P addresses representing social peers on the P2P network and will only route IP packets to or accept IP packets mapped to P2P addresses.

Every SocialVPN node joins a structured P2P overlay called Brunet [43], developed at the University of Florida. The structured P2P overlay network manages the connectivity between the peers through self-configuration and self-organization as nodes join and leave the P2P network, and employs decentralized NAT traversal techniques to connect nodes that are not directly addressable over the Internet [42, 43]. When socially connected peers are identified on the overlay network, direct IP tunnels are formed and maintained to allow low latency IP communication amongst peers. In the analysis of the previous chapter, the cost of maintaining the social connections on the P2P overlay is measured as the number of connections increase. The analysis shows the cost to be manageable.

### 2.2.5 Dynamic IP Allocation and Translation

Deployments of SocialVPNs need to accommodate user bases that can be quite large – there are currently hundreds of millions of users registered with WBSNs. This presents a challenging problem because VPN endpoints require unique IP addresses

and is thus subject to several constraints. IPv6 infrastructure and applications are not widespread; public IPv4 address spaces are scarce; private IPv4 addresses do not scale to large numbers, and can collide with local address spaces of users who are increasingly bound to private networks behind NAT devices. In the proposed approach, through address translation, SocialVPN can scale to numbers of users larger than the limit imposed by the IPv4 private address range while avoiding address space conflicts with end user networks.

The key idea behind this approach exploits the fact that, while the total number of participants in an online social network can be very large (hundreds of millions of users), the number of relationships a single user has at any point in time is significantly smaller (typically hundreds to thousands). Nonetheless, while a user's number of relationships is relatively small, it is larger than the number of network interfaces that modern operating systems can typically handle. Therefore, a solution that multiplexes social networking connections into a single virtual network interface with a single virtual IP address is desirable.

SocialVPN accomplishes the goal of presenting a single IPv4 virtual network interface while avoiding address space collisions, as follows. SocialVPN maintains, at each user's endpoint, a private IP address space that is sized to accommodate the expected number of social connections a user may have. For instance, a 16-bit class B private address space supports tens of thousands of connections. This private address space is dynamically assigned locally by the SocialVPN router such that it avoids collision with any existing network interfaces of a user's local machine.

The following example (see Figure 2-2) illustrates the address translation process. For example, if a user has a physical network interface with an IP address 172.16.5.16 with a netmask 255.255.0.0, the virtual network interface used by the SocialVPN router is automatically configured to use a non-conflicting IP address range such as 172.17.0.2 and peers are allocated IP addresses in the 172.17.x.y range. Each peer is

also assigned an IP address in the selected local address space; a mapping between the IP address to peer's P2P address is maintained by the SocialVPN router. The SocialVPN router uses the IP-to-P2P mappings to route IP packets to the appropriate peers based on the destination IP (e.g. destination IP 172.17.34.231 maps to P2P address node:4AB1, so all IP packets with destination 172.17.34.231 go to peer with that node address, see Figure 2-2).

Because of the dynamic IP allocation, IP packets need to be translated by the received SocialVPN router to match the receiver's IP-to-P2P mapping. Consider Alice has a local virtual IP address of 172.17.0.2 and her friend Bob is dynamically assigned the IP address 172.17.23.12 by her local SocialVPN router. On Bob's local SocialVPN router, he has a local virtual IP address of 172.25.0.2 and Alice is dynamically assigned a virtual IP of 172.25.43.89. When Alice communicates with Bob over IP, Alice's SocialVPN router receives IP packets from the host OS with 172.17.0.2 as the source IP and 172.17.23.12 as the destination IP (Bob's address), and tunnels them directly to Bob over the P2P overlay. Bob's SocialVPN router hence receives the IP packets from the overlay and changes the source and destination IP addresses to the appropriate addresses assigned by Bob's SocialVPN router; meaning the source address changes from 172.17.0.2 to 172.25.43.89, and the destination address from 172.17.23.12 to 172.25.0.2 (see figure 2-2 step 4). Since only IP headers are translated, all protocols above Layer 3 such as transport layer UDP and TCP ports information remain unchanged.

Due to this translation, IP addresses are not globally valid in the virtual network. For instance, Alice and Bob both have a friend Carol, Alice assigns Carol an IP address of 172.17.34.231, but Bob gives Carol the 172.25.1.94 IP address. Since only Alice's SocialVPN router has the distinct local IP mapping for Carol, Bob's router could not resolve the 172.17.34.231 IP address to Carol's P2P address; in other words, if Alice tells Bob that he could ping Carol's machine at 172.17.34.231, Bob's ping messages

would never reach Carol's machine. Although source and destination IP addresses are updated, UDP or TCP ports are not changed. This is similar to the network address translation performed by full cone NATs. Most client-server applications (e.g. Web browsing, file sharing, remote desktop) are able to work without changes with such a NAT behavior; however, some protocols which exchange IP addresses in the payload of messages (e.g. FTP, SIP, MDNS) require packet inspection and IP translation in the payload to ensure compatibility.

### 2.2.6 The P2P VPN Social Graph

SocialVPN has an interesting characteristic not found in traditional VPNs in that the encrypted tunnel links of the P2P VPN network represent the edges of a social graph with small world characteristics. For instance, even though both user B and C are part of user A's VPN, it does not imply that users B and C have a VPN link to each other. This is analogous to a social network where Alice can be friends with both Bob and Carol, but it does not mean that Bob and Carol are friends. For scalability and security, it is important to only have VPN connections with trusted peers; there is no point of having links with peers of no common interest. This is a departure from the common concept of private networks such as local area networks (LANs) and traditional VPNs where there is the expectancy of all-to-all connectivity among nodes in the same network.

## 2.3 Experiments

The focus of these experiments is to determine the performance of the system when the DHT is employed as the certificate data store. In addition, a centralized social networking identity/relationship provider was used to facilitate the bootstrapping of a network during the experiments. Some key metrics of the proposed design were analyzed: 1) the time to form the private links with the peers, 2) the bandwidth overhead cost of maintaining these private P2P links, and 3) the server load on a centralized social backend.

In the experimental setup, the Amazon Elastic Cloud (EC2) was utilized to deploy virtual machines with one virtual core and 1.7 GB of RAM each to test social networks of varying sizes. These virtual machines are based on Debian 5 running the 2.6.26 Linux kernel. The software is written in C#; hence, the Mono .NET Runtime version 1.9.1 was used. Each virtual machine ran the P2P architecture which connected to a pre-deployed P2P overlay network of 500 nodes running on PlanetLab [47]. An initial 20 SocialVPN nodes were also deployed in a computing cluster at the University of Florida (UF); these nodes represented peers that are already online when other social nodes join the network and also help against the clustering effect of Amazon EC2 nodes.

All of the nodes connected to the same social networking backend to obtain peer relationships and peer certificate fingerprints. A Django-based backend was implemented which provided the necessary services of a social networking infrastructure which was deployed on the Google App Engine Cloud Infrastructure [48]. The Google App Engine backend provided the tools to measure the number of HTTP requests made by the social endpoints, as well as the storage and CPU requirements on the backend. Also in the experimental setup, the X.509 certificate fingerprints were stored on the online social networking backend, while the actual X.509 certificates were stored on the DHT to minimize the storage requirements on the backend.

### 2.3.1 Link Creation Time

The experiments also helped analyze the time taken to form direct, private connections once peers were discovered through the social networking backend. Two main steps were examined: the X.509 certificate retrieval from the DHT, and the formation of the encrypted connections between peers, which includes the exchange of Diffie-Hellman messages over the P2P overlay to establish a pair of symmetric keys. Understanding the time taken to retrieve an X.509 certificate from the DHT is important to prove that a DHT is capable of supporting such a load with an acceptable retrieval time. The measurements taken were from deployments of 16, 32, 64, and 128 nodes

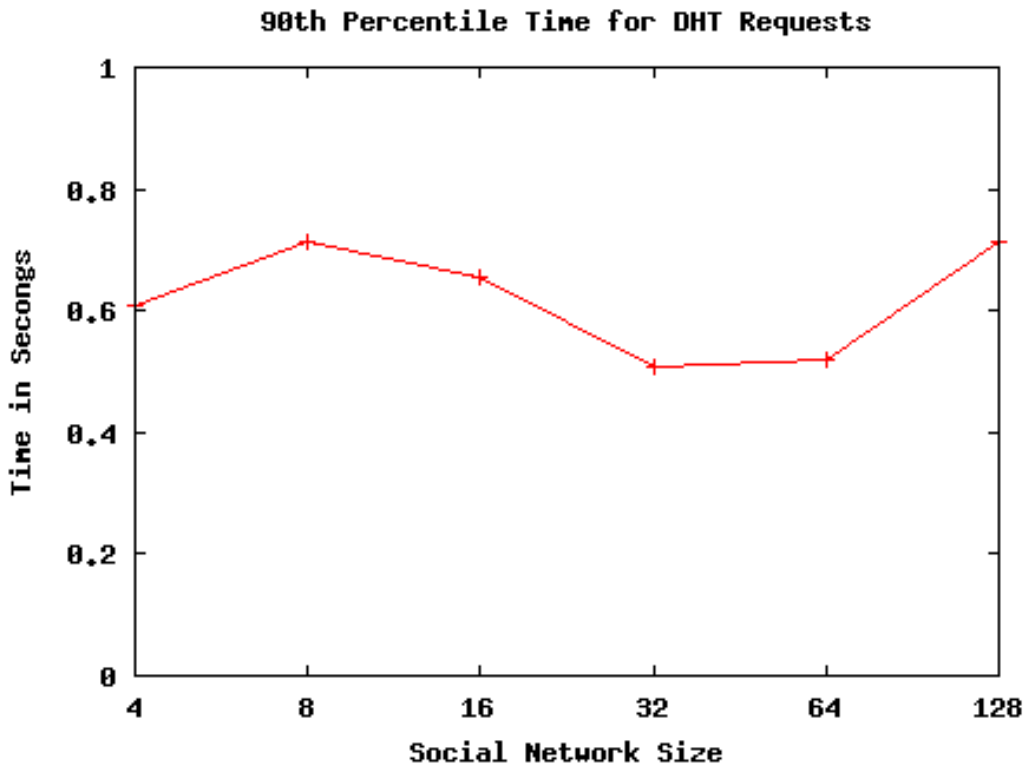



Figure 2-3. 90th Percentile Time for deployments of 4 to 128 nodes. As network size increases, DHT lookup performance remains around 600 milliseconds.

deployed on Amazon EC2. These Amazon nodes were brought up simultaneously and ran for a 50-minute period in each deployment scenario. In all cases, there was all-to-all social connectivity among the nodes; in other words, with 16 Amazon nodes, each node has 15 direct connections with the other Amazon nodes, plus the additional 20 direct connections with the nodes running at the UF cluster.

Figure 2-3 shows that as the number of peers increase it consistently takes less than 800ms for 90% of the DHT requests, which is a reasonable timespan to obtain an X.509 certificate from a distributed datastore. Figure 2-4 provides a histogram demonstrating the distribution of the certificate retrieval times from the DHT with the various network sizes. For network sizes of 16 and 32, all DHT retrieval times were below 1200ms. A DHT request performed on a Chord-like structure overlay has a complexity of $\lg(\mathrm{N})$ hops where $\mathrm{N}$ is the number of nodes in the P2P network. Hence,

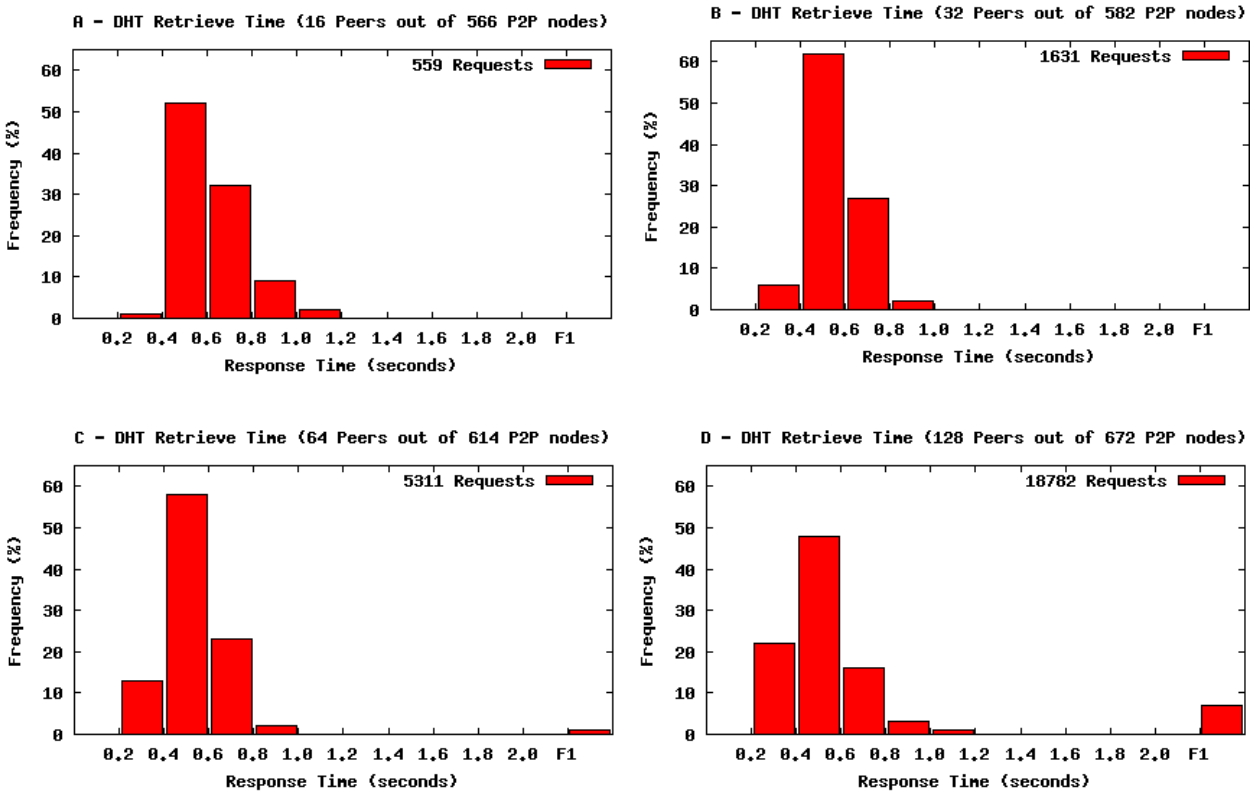



Figure 2-4. DHT Retrieval Times for deployments of 16, 32, 64, and 128 SocialVPN nodes. For 16, 32 node deployments, 100% of DHT queries took less than 2 seconds. For 64-node deployment, 98% of 5311 requests took less than 2 seconds. For 128 node deployment, 94% of 18782 requests took less than 2 seconds. F1 represents percentage which failed on first attempt, attempts succeeded after retries.

with a P2P network size of around 500 to 600 nodes, it takes an average of 9 hops to reach the node responsible for storing the $< \text{key}, \text{value} >$ pair. Also, since the P2P network is deployed over PlanetLab which has globally dispersed nodes, it is not hard to see why a DHT request may take up to 1200 milliseconds to return a value. In the 64 and 128 deployments, there was a 1% and 6% failure rates respectively for first-time DHT requests caused by cases where nodes issue DHT lookups under $< \text{key}, \text{value} >$ pairs that have not been successfully stored in the DHT. This happens because when 64 or 128 nodes join the P2P network simultaneously it takes longer for the P2P network

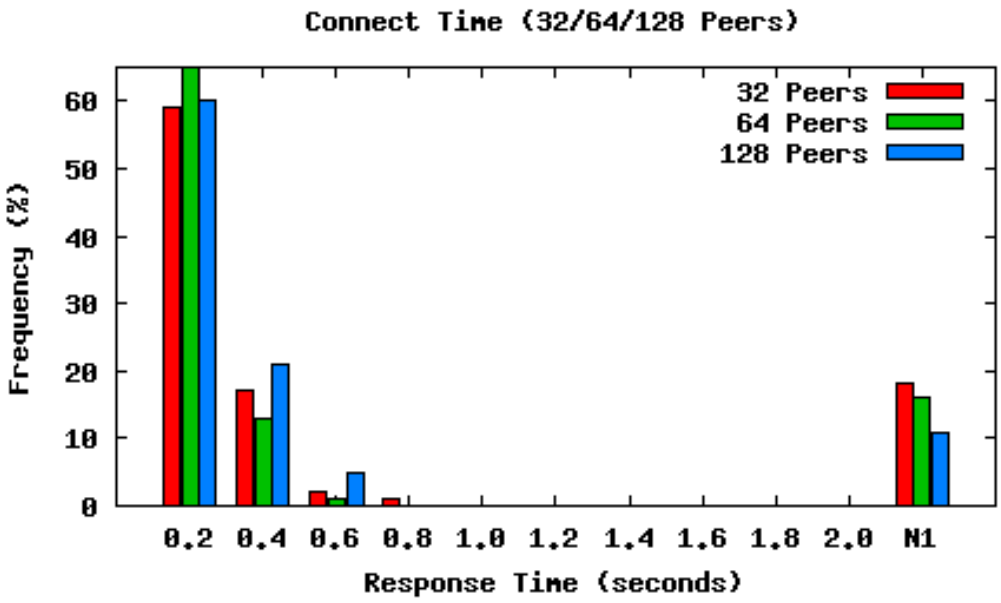



Figure 2-5. Connect Time for deployments of 32, 64, and 128 SocialVPN nodes. Connection times remain stable as the network increases up to 128 nodes. N1 represents percentage with more complicated NAT environments, some nodes took up to several minutes before they could form direct connections.

to stabilize causing some DHT PUT request to initially fail; all DHT lookups eventually succeeded after subsequent retries.

Figure 2-5 shows the connection times for network sizes 32, 64, and 128. In all cases, over 85% of the connections took less than 800ms. The connection time involves peers creating direct paths to each other where NAT traversal is performed when necessary. With Amazon EC2 nodes, NAT traversal is not required between these nodes since they are on the same internal network. NAT traversal is necessary between the UF and Amazon nodes because the UF nodes are located behind a NAT. In each case, between 15% to 20% of the connections took over one second to connect due to the NAT traversal process between the UF and Amazon EC2 nodes.

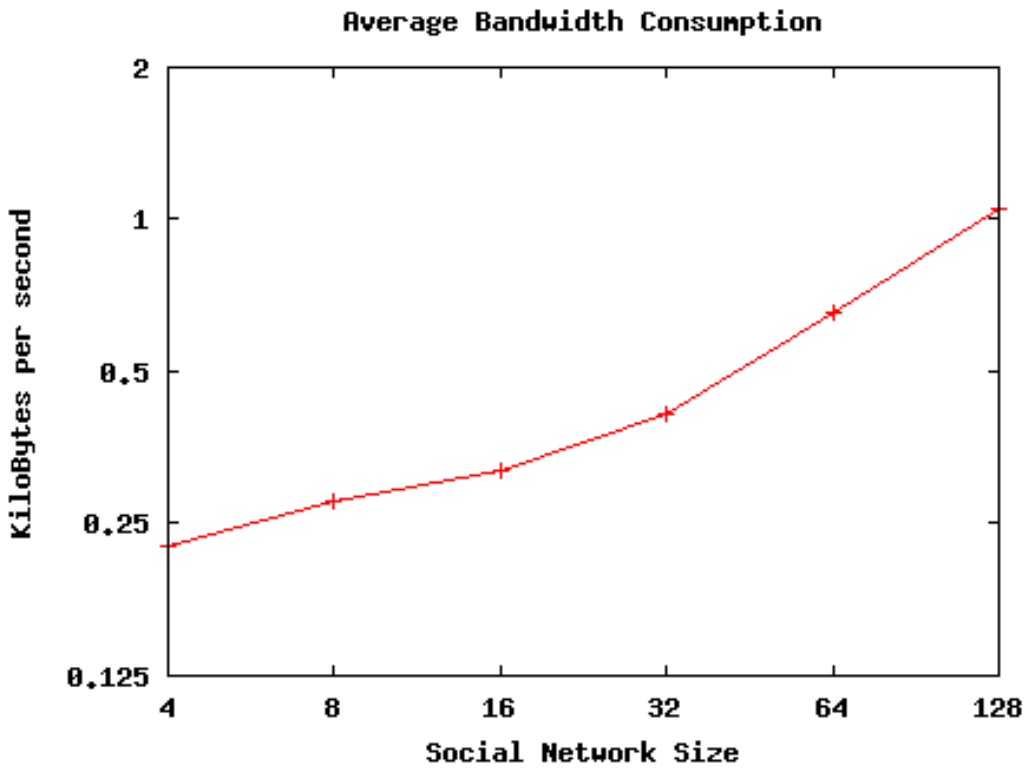



Figure 2-6. Bandwidth cost as social network size increases from 4 nodes to 128 nodes. This is the bandwidth consumed by the SocialVPN router to maintain the structure the P2P network as well as the direct links between peers.

### 2.3.2 Bandwidth Cost

The bandwidth overhead of maintaining the social connections on the P2P overlay was also measured. The calculated bandwidth is the average number of bytes transferred per second across all the Amazon EC2 nodes for each 50-minute deployment. Figure 2-6 shows that the bandwidth cost increases proportionally with the number of peers in the network starting at 0.2 KBytes per second to 1.2 KBytes per seconds from a network size of 4 friends to 128 friends. The P2P overlay uses bandwidth for maintenance such as routing table updates, probing, DHT operations, and nodes joining/leaving the overlay. Peers are proactively probed (every 15 seconds) to determine the status of the node. Hence, there is a small bandwidth overhead for maintaining the structured overlay and the direct social connections — a few percentage points of a typical broadband with access to hundreds of Kbit/s bandwidth.

### 2.3.3 Server Load on Backend

This research argues that the hybrid approach leverages the benefits of P2P communication to alleviate the infrastructure demands of online social networks; therefore, the number of HTTP requests were measured, and bandwidth consumed by the experiments. The current implementation essentially makes three types of HTTP requests to the social networking backend: get friends, get fingerprints, and store fingerprint. These three HTTP requests are performed every five minutes, and when a new social node joins the network, to synchronize the remaining nodes. According to the Google App Engine site monitoring tools taken over a 24-hour period for 38 social nodes, the server received a total of 44121 HTTP requests with a bandwidth cost of 300 MBytes. Each node made an average of 48 HTTP requests per hour, and consumed 293 KBytes of bandwidth per hour for communication with the social network. The cost of aggregating all the work on a single server was compared. The per-node costs are trivial, but even for this small example, the server bandwidth costs would be non-negligible.

### 2.3.4 Prototype Performance

To date, the SocialVPN router has successfully integrated with the Facebook API, with an open-source Web back-end (Drupal), and with support for manual entry of self-signed certificates through the user interface (see Figure 2-7). The latter approach supports users to copy/paste PGP-signed email messages containing SocialVPN certificates as the mechanism for discovery and public key exchange. The prototype router leverages the IPOP overlay, which supports both IP-over-P2P tunneling and a DHT. The user-level router is implemented in C# and works with the Mono and .NET runtimes, using the tap virtual network device. The prototype implements a user-level security stack on the overlay that is inspired by the IPsec protocol. A previous implementation that integrates with a kernel IPsec stack has been demonstrated in prior work. Deployments of the prototype on hundreds of wide-area PlanetLab nodes have

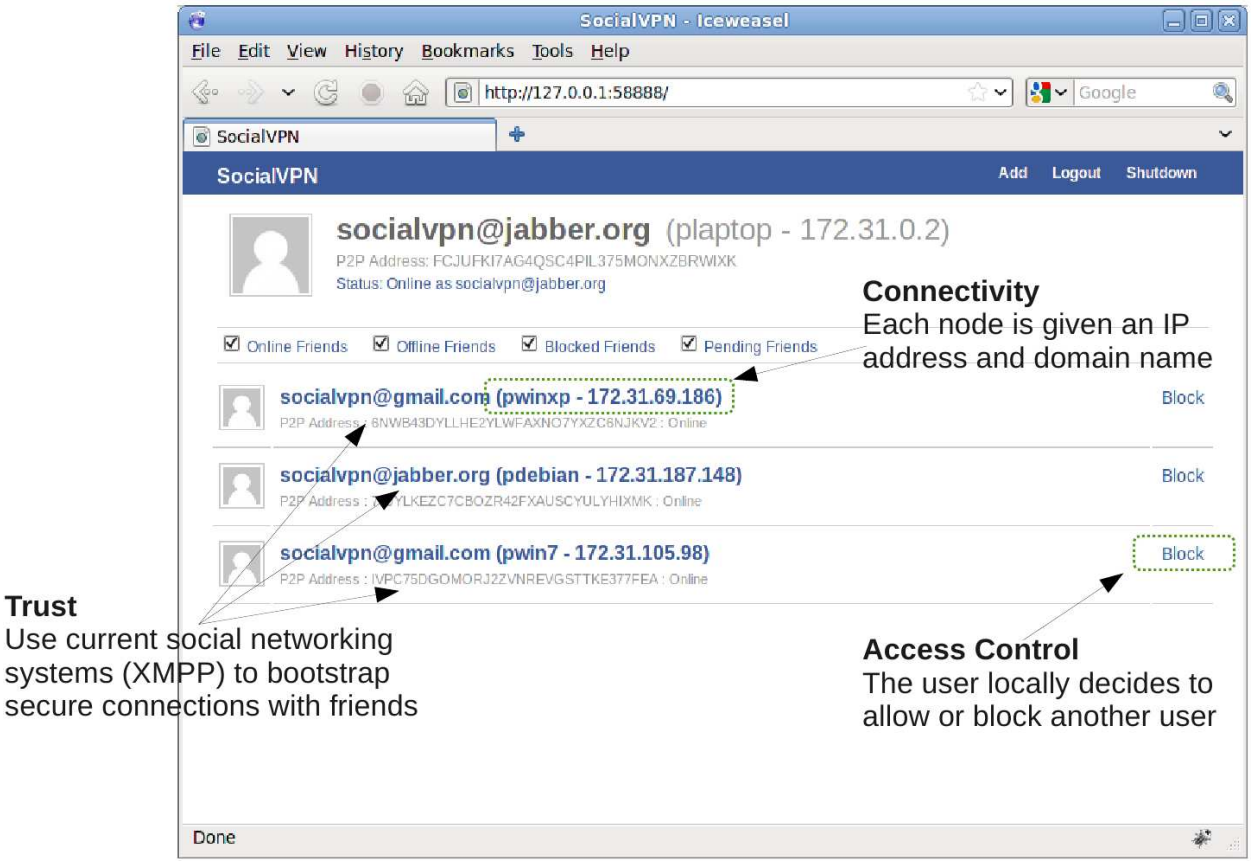



Figure 2-7. SocialVPN User Interface.

been running continuously for months; in the experiments described below, a separate overlay was created on resources distributed across PlanetLab, Amazon EC2 and resources within the lab at the University of Florida to assess the performance of the prototype and establish the feasibility of the proposed approach quantitatively.

Several applications have been qualitatively and quantitatively assessed the latency overhead and bandwidth achieved by the prototype. The following applications were successfully tested between SocialVPN nodes: VNC- and RDP-based shared remote desktop sessions, file sharing through SMB and NFS, multi-player 3D LAN game (Valve's CounterStrike), HTTP server with Apache, music sharing/streaming through iTunes, multicast-based service discovery (MDNS/SD) Bonjour [24], direct P2P chat with Pidgin over Bonjour, VoIP with Ekiga.

The bandwidth and latency of SocialVPN are measured. Two SocialVPN nodes ran on the same cluster connected by a 1GB Ethernet switch. The ping tool was used to measure the round-trip latencies of 100 ICMP request/reply packets and obtained an average latency of 1.1 ms. The bandwidth was also calculated with the Iperf network measurement tool by measuring the TCP throughput of a 30 second-transfer and achieved a bandwidth of 30 Mbps. Overall, the observed performance is acceptable for wide-area environments using a social networking system which is the target of the proposed architecture. Nonetheless, performance improvements are actively being pursued.

## 2.4 Summary

Social networking infrastructures can greatly facilitate the configuration and deployment of systems because they have proven quite effective in enabling users to discover and associate with their peers. This chapter shows an architecture where social connections established through user-friendly Web-based infrastructures can effectively guide the creation of encrypted, authenticated connections at the computer network layer. This is achieved in a user-transparent manner and supporting a wealth of existing TCP/IP applications on top of existing networking infrastructure. To date, this is the first work to integrate social networks and wide-area peer-to-peer overlay networks. A prototype implementation of the proposed approach and experiments with a deployment on PlanetLab and Amazon EC2 infrastructure has shown this approach to be feasible and promising.

In this chapter, three methods of cryptographic key management were discussed. Through the use of trusted third-party systems, X.509 certificates were efficiently exchanged among social peers. These certificates provided the crucial information needed to bootstrap encrypted, peer-to-peer connections to social peers. Through integration with popular social networking systems, users are able to automatically discover friends and manage their security credentials in a seamless way. Additionally.

by adopting the standard PKI model to bootstrap encrypted connections through the use of X.509 certificates, the proposed architecture is able to reuse well-established standards and protocols such as the Diffie-Hellman exchange. Consequently, social networking application developers do not have to readdress the complexities of cryptographic key management, social account creation and relationship management since they are already provided in the proposed architecture. More importantly, developers know that their applications will inherit the intrinsic privacy of the system, since they know that all communication is authenticated, and encrypted end-to-end. With the privacy issues handled by the architecture, designing a social networking application become a much less cumbersome venture.

This chapter also described a decentralized method of implementing a self-configuring virtual private network which tremendously facilitates social networking applications for social peers over the Internet. SocialVPN combines various existing technologies such as social networking APIs, structured peer-to-peer overlays, and PKI certificate models to provide an easy-to-use, scalable, yet secure IP-level communication link among friends. By maintaining a structured peer-to-peer overlay as a messaging substrate, peers can initiate NAT traversal which enables direct IP traffic tunneling amongst two peers. The structured peer-to-peer overlay also provides a distributed datastore through a DHT which can be used for X.509 certificate publishing and retrieval.

# CHAPTER 3
# BOOTSTRAPPING CONNECTIONS WITH UNSTRUCTURED OVERLAYS

The proposed peer-to-peer architecture so far builds upon a structured P2P overlay called Brunet [43]. Brunet is a Chord-like structured P2P network that arranges all of the nodes in the system and it has useful properties such as self-organization in the face of node churn, all-to-all connectivity and efficient routability among all nodes. However, there are some weaknesses to the dependence on the ring-based structure. The first problem is vulnerability to Sybil attacks. A Sybil attack is when a malicious user in a P2P system is able to create multiple identities in the network. Through the use of Sybil nodes, an attacker can control and monitor a large portion of the P2P overlay.

Structured overlays are also susceptible to DHT localization attacks [49] where malicious nodes crowd a particular source node by selecting the appropriate P2P identifiers and intercept all communication to and from the victim node. Structured overlay networks are also vulnerable to unrecoverable network partitions. For instance, consider 1000 nodes located in Egypt in a structured P2P overlay of 100,000 nodes located all around the globe. If the Egyptian government disconnects the country from the rest of the Internet, the 1000 nodes in Egypt are disconnected from the P2P overlay and therefore cannot route any messages even among each other.

The purpose of this chapter is to describe support for an unstructured overlay and provide similar capabilities as the structured overlay, primarily routability, and direct connectivity from behind NATs. The majority of the popular P2P applications deployed on the Internet such as GnuNet, FreeNet, Gnutella and so on are based on unstructured P2P networking. These networks achieve the goals of routability and direct connectivity in a variety of methods, where some use flooding, random-walking, greedy routing, and probabilistic routing [50]. Also unstructured P2P overlays are more resistant to node churn because they do not have to maintain any structure. However, this lack of structure makes it harder to achieve some of the functionality that is easily attained

through a structured overlay. Support for mobile devices is also discussed in this chapter due to the proliferation of mobile devices, it is key that the proposed P2P framework also enables deployments for mobile computing. Addressing new paradigms of computing (i.e. cloud and mobile) requires some fundamental redesign of the previous core components but the improved approach allows for more much flexibility, interoperability, and extensibility.

This chapter presents the TinCan design, a P2PVPN that allows flexible VPN overlays of different topologies that can be instantiated atop Internet infrastructure with low configuration and management overhead [1] . TinCan integrates with existing social networking services for peer discovery and notification, reflection and relaying to allow deployments that securely bootstrap private peer-to-peer tunnels. The overlay implied by private end-to-end TinCan links exposes IP endpoints, allowing existing applications to work unmodified, and providing a basis for overlay peer-to-peer routing for the virtual network. The TinCan design also supports both IPv4 and IPv6 routing within the VPN which is implemented as IPv6 packets encapsulated within UDP packets and sent over IPv4 P2P tunnels, as well as IPv4 packets within IPv6 UDP packets.

A key goal in the design is to minimize the amount of configuration and infrastructure necessary to sustain these virtual private networks. Because TinCan runs on endpoints (e.g. VMs or personal devices), it requires little additional infrastructure for maintaining the network. TinCan links make it possible for VMs and mobile devices to tunnel IP traffic directly to each other — even when constrained by NATs — while simultaneously giving end users the flexibility to define the IP address ranges, subnets, and access control policies for their private network. TinCan integrates with ubiquitous messaging overlays that use the XMPP protocol for signaling, along with well-adopted technologies

[1] The name is inspired by tin can phones that provide private, ad-hoc communication links between friends

for NAT traversal (STUN, TURN, and ICE [51–53]) to bootstrap encrypted TinCan links. In one use case, social peers can run TinCan to deploy VPNs comprised of their personal (mobile) devices and their social peers by leveraging existing Internet services for discovery and reflection (e.g. Google Hangouts, Jabber.org XMPP and STUN servers). The only requirement for deploying a TinCan VPN is an XMPP server; therefore, end users can use any freely available XMPP service on the Internet, or deploy their own private XMPP server such as *ejabberd* [54].

The novel design of TinCan is logically divided in two key layers — reminiscent of the OpenFlow model but applied to tunnels over UDP/TCP links: 1) a datapath packet capture/forwarding layer, responsible for capturing/injecting packets from a virtual NIC, and maintaining TinCan links (over UDP or TCP) to neighboring peers, and 2) a control layer, responsible for implementing policies for the creation and tear-down of TinCan links. Each TinCan peer runs the two layers; communication across layers within a node is achieved through a JSON-UDP RPC interface. The available API allows for the control of TinCan link creation and deletion, mapping IP addresses to identities and TinCan links, and configuring virtual networking interface. Coordination among endpoints and overlay routing is possible through message forwarding along TinCan virtual IPv6 links, supporting user-defined overlay topologies and routing policies implemented as a separate module from the core datapath.

To demonstrate the applicability of TinCan in different use cases, TinCan implements a common datapath based on Google's libjingle P2P library [55], and two different TinCan controllers: a "group" controller that follows a typical VPN model providing a private subnet, and a "social" controller that automatically creates VPN links from social networking relationships established through an external OSN provider. With the group controller, nodes bind to the same subnet in the virtual network and can address each other using unique private IP addresses within the scope of the VPN. In social mode, each user is able to define their own private IP range/subnet and locally map social

peers to IP addresses within that subnet thus forming an unstructured social network graph overlay topology.

The analysis shows that the TinCan design is quite practical and scalable without imposing unrealistic load on the XMPP server. In the experiments, a network of 300 nodes consumes 29 KB/s of bandwidth on the XMPP server. The management of these TinCan links uses about 1 KB/s of bandwidth per connection and there is a 14% networking overhead. This overhead is due to the use of an MTU of 1280 bytes — selected to minimize packet fragmentation — rather than the traditional 1500 byte MTU along with the cost of an additional 40-byte header necessary to encapsulate the virtual IP packets. To measure the maximum throughput, an experiment was conducted between two nodes in a 1 Gbps LAN and ran the iperf networking benchmark to obtain the bandwidth measurements. The results show a latency of less than 1 ms and a TCP bandwidth of 64 Mbps; since the target is to create virtual networks across the Internet, for most applications, the bottleneck will be the bandwidth limit imposed by their local ISPs.

The main contribution of this chapter is a novel VPN design that leverages XMPP servers to bootstrap end-to-end VPN tunnels, supports decoupled controller/datapath model and P2P communication among controllers to implement different VPN membership, address mapping and overlay topology/routing policies, and leverages existing P2P technologies (STUN, TURN, and ICE) for establishing direct and secure P2P tunnels for IP connectivity. This is also the first P2PVPN design that allows mobile devices to maintain their virtual IP address as they migrate across different networks while automatically re-establishing P2P connections with other nodes in the virtual network without the use of a relay.

## 3.1 Background and Related Works

**Overlay Virtual Networking Research.** Academic and industry research have explored applicable solutions for virtual networking that allow geographically-dispersed

nodes to create virtual private networks across the Internet. IBM researchers have developed VirtualWire [56] which implements a layer 2 virtual network tailored to the deployment of legacy applications and VM migration across different physical networks. Virtualwire is a hypervisor-level virtual network integrated with the Xen-Blanket [57] nested virtualization technology, enabling VM migration across public clouds. VIOLIN [39] uses a very similar approach to Virtualwire providing layer 2 networking with components such as switches and routers implemented purely in software. A drawback with these approaches is that users are still required to configure virtual switches, routers, and deploy their own DHCP and DNS servers within the virtual network. The TinCan approach does not necessitate setting up additional DHCP and DNS servers.

VNET [38] provides layer 2 connectivity across different physical networks and it is also implemented at the hypervisor level. This is accomplished through a layer 2 proxy that bridges two different networks across the Internet. All of these previous works do not explicitly deal with NATs and firewalls, and assume the availability of VPN gateways and virtual routers with public IP connectivity. As the pool of IPv4 addresses becomes more scarce — compounded by recursive virtualization and the use of containers — establishing end-to-end virtual network links across NAT-constrained devices becomes increasingly important. VINE [37] is a layer 3 virtual networking alternative which supports NAT/firewall traversal through relaying. However, it requires users to configure the virtual routers and does not provide end-to-end tunnels that bypass a relay/router node. This work enables direct end-to-end IP tunneling without the need for a router middleman because each node runs an IP router locally.

**Host-based and Mobile Virtual Networking.** OpenVPN [58] is a solution that is applicable in mobile virtual networking. However, OpenVPN follows a client/server architecture where all IP traffic is routed through a central gateway. This incurs high latency and creates a resource bottleneck. Many other solutions improve on the OpenVPN model; for instance, Hamachi [59] uses a proprietary central server to

setup P2P connections between hosts, even through NATs and firewalls. IP traffic is tunneled over these encrypted P2P connections. Other approaches such as Tinc [60], Vtun [61], and N2N [62] all create mesh VPNs where nodes create direct connections to each other, but they require nodes to be openly accessible over the Internet. While these solutions can potentially be used to enable wide-area virtual networking, they are not currently supported by mobile platforms, and do not provide a flexible overlay architecture that supports other VPN topologies, such as those implied by friend-to-friend social network graphs. UIA [63] is a closely-related design aimed at providing ad-hoc virtual networking for mobile devices. One key difference in TinCan is the use of existing infrastructure, including OSN providers, to mediate peer discovery and bootstrapping. Previous work, IPOP [64], is a peer-to-peer VPN based on a structured P2P overlay for bootstrapping direct connection between nodes. While sharing similar goals, TinCan addresses several limitations of the IPOP design: in IPOP, peer discovery, bootstrapping, reflection, and relaying are provided by an overlay where peer-to-peer communication is layered atop a common structured P2P library (Brunet). TinCan decouples discovery, reflection, relaying and bootstrapping, decouples datapath from control modules, and exposes P2P communication through virtual IP links, allowing multiple overlay topologies. The TinCan design does not depend on a structured P2P overlay, it uses publicly available STUN servers and XMPP servers to bootstrap P2P connections.

## 3.2 Design

This section describes the core components of the TinCan design which include a packet capture/forwarding datapath module, a network controller module, a discovery/notification overlay, reflection and relay servers. TinCan primarily enables an extensible framework for building P2PVPNs for various types of deployments. While the TinCan design supports other implementations, currently TinCan uses XMPP for discovery/notification, STUN for reflection, and TURN for relaying, and leverages the libjingle library (developed

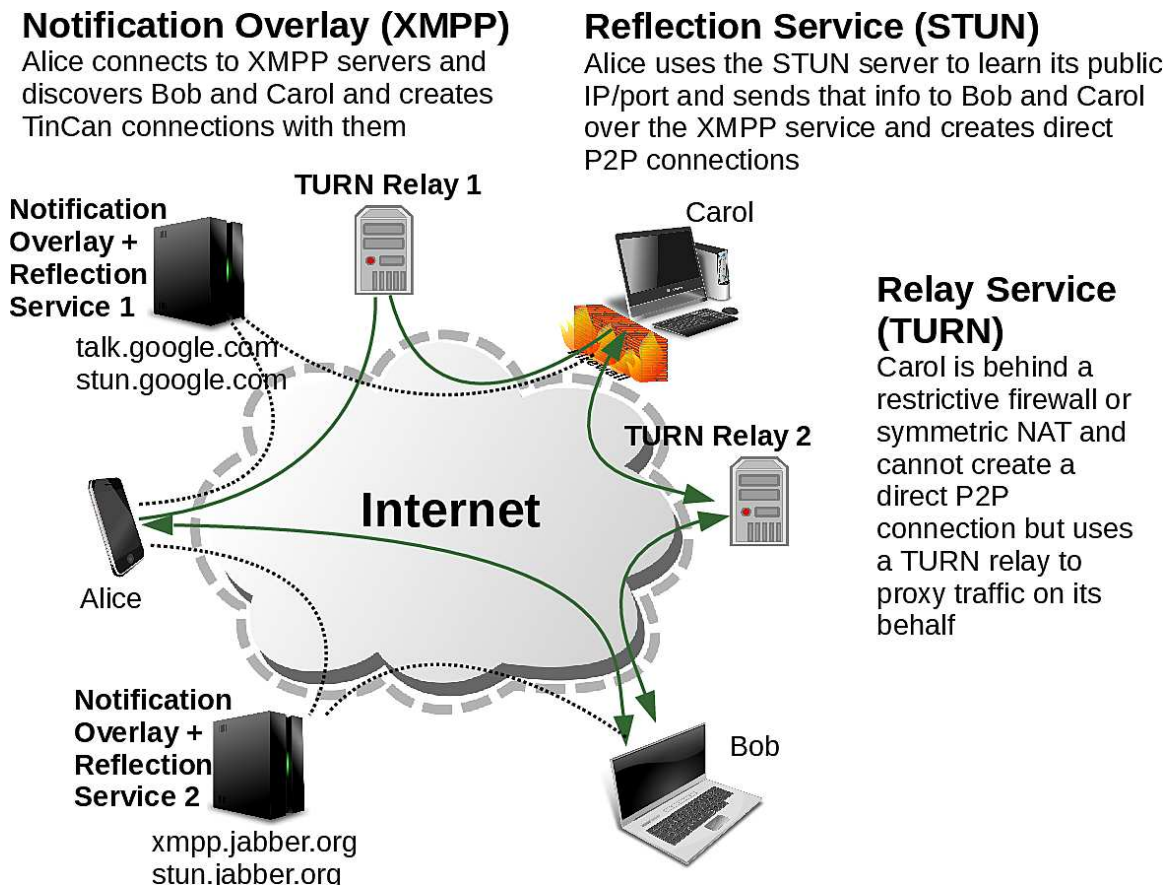



Figure 3-1. TinCan Components and Overview

by Google) to establish and maintain P2P TinCan links using the aforementioned services. Figure 3-1 gives a general overview of the services involved in deploying the system.

### 3.2.1 Endpoint-Hosted Components

**Datapath packet capture/forwarding module.** This component is a user-level module that runs on the end user device. It creates a virtual networking interface (vNIC) on the local operating system to capture and inject IP packets to/from local applications. It also possesses the mechanics of creating, maintaining, and tearing down encrypted TinCan links to peers, and manages a local routing table that maps a virtual IP address of an appropriate TinCan P2P link. In a typical packet flow scenario, this module reads an IP packet from the vNIC on the local OS, uses the destination IP address to lookup

whether a mapping to a TinCan P2P link exists. If a TinCan link exists, the IP packet is encapsulated and sent directly over this link to the receiving data path module at the other endpoint node. Upon receiving the IP packet, the receiver decapsulates and injects it in the local vNIC (see figure 3-2).

The datapath module tracks the state of the local P2P links, and maintains a connection to one or more notification overlays (e.g. XMPP servers). TinCan links are typically tunneled over UDP — as it is most amenable to NAT traversal — and use DTLS for privacy, authentication, and integrity. The design also uses keep-alive messages to determine the state of links, and uses the notification overlay to verify that online peers that are available to accept TinCan connections requests. This module is responsible for implementing the mechanisms to maintain TinCan links; however, it does not prescribe the policies associated with link creation and tear-down. To this end, it exposes an RPC interface to the controller module, decoupling mechanism from policy. The RPC API exposes the following functionality: 1) configuration of the virtual network interface, 2) creation and deletion of TinCan links, 3) registration into the notification overlay, and 4) adding a mapping for a destination virtual IP address (see figure 3-2).

**Network Controller.** The controller module implements different policies for managing TinCan links and the overlay topology. Through the API exposed by the datapath module, the controller determines the criteria for TinCan link creation, deletion, and the mapping of IP addresses. For example, a controller may implement a policy to create P2P connections when a node joins the network for a small-scale VPN with a proactive link creation policy, or only create connections on demand when virtual IP traffic is detected between endpoints. The controller also manages the configuration of the vNIC, including the IP address and network mask. Moreover, it maintains the credentials for connecting to the discovery overlay for certificate exchanges with peers for setting up private TinCan links. Figure 3-4 shows an example configuration for a controller.

In addition to programming local forwarding tables, the controller is also responsible for routing virtual IP packets not mapped to local TinCan links through one or more hops. This mechanism is used to route packets when a direct TinCan link is not available, for instance while a link is being initialized. Controllers bind to an IPv6 vNIC that allows it to communicate to neighboring controllers over TinCan links; this private IPv6 address is configured with a unique node ID which can be used for identifier-based routing. In doing so, the controllers can use this mechanism to implement different overlay topologies and routing algorithms without requiring changes to the core datapath (see figure 3-3). Finally, the controller also determines the policies for various network events such as node arrival and departures, TinCan connection requests, and link failures.

### 3.2.2 Internet-Hosted Components

**Notification/Discovery Overlay.** As stated above, the datapath module maintains a communication link with a notification overlay (e.g. XMPP server) that allows for the advertisement of network-wide events such as node arrivals and departures. The notification overlay plays the role of the trusted out-of-band channel for bootstrapping encrypted TinCan connections (see figure 3-4). When two nodes decide to create a TinCan connection, they exchange a list of candidate endpoints (i.e. public and private IP addresses and ports) and security credentials (i.e. X.509 certificate fingerprints). The notification overlay provides the following primitives: 1) multicast notification to peers connected to the overlay (e.g. XMPP buddies), 2) unicast message delivery to a specific node, and 3) node authentication and message integrity guaranteeing trusted node identity and message delivery.

**Reflection and Relay Servers.** There are two services needed in the public network to enable the bootstrapping of TinCan links through NATs and firewalls. First, reflection servers are used to inform nodes of their public-facing IP addresses and ports. Nodes are then able to exchange their public IP information with other nodes through

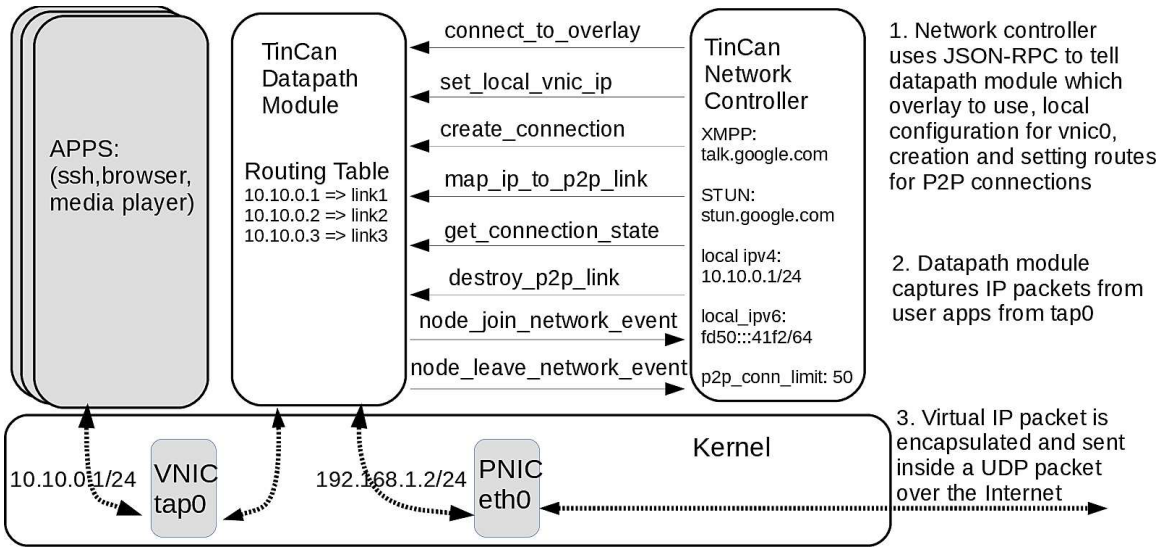



Figure 3-2. Interaction between TinCan Modules

the notification overlay to bootstrap TinCan connections. While most NATs are amenable to UDP NAT traversal, around 8% of the time [55], nodes behind symmetric NATs or some restrictive firewalls cannot create direct TinCan connections (see figure 3-1). In those cases, they require the assistance of a relay server with a public IP address. A relay service serves as an indirect communication path when a direct TinCan link cannot be established.

### 3.2.3 Controller Policies

A key aspect of the TinCan design is extensibility, accomplished through decoupling of the controller and data path. This approach is inspired by OpenFlow [65], but applies at the IP layer over tunneled links, rather than at layer 2 flows over physical links. To illustrate the extensibility of the design, this section describes two different controller models: a “group” VPN for virtual private clusters, and a “social” VPN connecting mobile devices of social peers (see figure 3-3). For the former use case, the controller creates a VPN where nodes join the same virtual subnet (e.g. 10.10.0.0/16) and IP addresses are assigned by the VPN network creator. Virtual IP addresses are bound to node identifiers within the scope of this VPN by configuring the node ID to be a cryptographic hash function of the virtual IP address. TinCan links are created on-demand in response

to IP packets being captured by the datapath module; while links are setup, packets may be dropped by a controller, or routed through overlay hops. In the “social” VPN model, the controller creates VPNs where per-endpoint virtual network address spaces are created at each node, peers are mapped dynamically to IP addresses within this namespace, and address translation is handled transparently. For instance, Alice has friends Bob and Carol; her VPN binds virtual IP addresses of Bob and Carol to a local private subnet (e.g 172.31.x.y). Bob and Carol have their own mappings of friends to virtual IP addresses within the local IP address space (e.g. Bob uses 10.15.x.y, Carol uses 192.168.5.y). Alice may link to Bob and Carol, while Bob and Carol may not have a direct link to each other if they are not friends. So far, two different controllers have been implemented but other controllers can be designed with various IP allocation and management policies.

**Network Admission.** Nodes joining the network advertise themselves and exchange connection information for bootstrapping through a trusted notification overlay. Hence, admission to the network is controlled by establishing identities and membership (e.g. friend-to-friend, or groups) in the notification overlay. In the group VPN scenario, each node in the network is given the following network settings: a private IP address and netmask, the network ID, the address of the notification overlay service, and a username/password for accessing the group through the overlay. For example, suppose Trent is a trusted user responsible for creating a VPN. Trent creates a personal VPN and distributes credentials for authentication and network access for each endpoint to join the notification overlay. Alternatively, Trent may establish relationships (e.g. XMPP buddies) with other users who are authorized to join the VPN. Trent would then determine the IP address range and netmask for the network and distribute these settings to each VM that joins the virtual network. In the social mobile VPN case, users have their own customized view of the network and thus define their own peer-to-peer trust relationships in the notification overlay, and select their own local private IP address

and netmask. In social mode, users are not required to share XMPP credentials to other members of the VPN because TinCan leverages existing social relationships to determine admission into the P2PVPN; therefore, only a user's XMPP buddies will be part of a user's social VPN.

**Proactive Link establishment.** If the controller implements a "proactive" link policy, it triggers a connection request as soon as a node joins the notification overlay and proactively creates TinCan P2P links to peers even if no IP packets are flowing. This policy has the benefit of reducing latency for packets, but comes at the cost of increased resource utilization (ports and bandwidth). The proactive link policy implied by this approach may be applicable for small overlays [66], but is not scalable (see figure 3-3).

**On-Demand Links.** An alternative controller policy is "on-demand connections" where TinCan links are formed when the controller receives a packet with a destination IP address that is not currently mapped to a TinCan link. This event triggers a connection request through the notification overlay, which results in a new TinCan link being mapped to the destination IP address. Such a policy causes a delay when connecting to new IP addresses. The controller with this policy also limits the number of connections, and can expire links with inactive IP flows.

**Social Profile Links.** Another connection policy deals is one where nodes are only interested in connecting with social peers rather than every node in a particular group. In this model, peers create proactive links with friends that they have frequently communicated with in past sessions, and on-demand or multi-hop routing through common friends for nodes for which communication is infrequent. Other policies may include a combination of on-demand and proactive connections, and create overlay topologies that attempt to match communication patterns expected (or observed) by applications (see figure 3-3).

**IP Addressing and Translation** The controller has the flexibility to assign an IP address to the device and map friends to IP addresses in a subnet range that does not

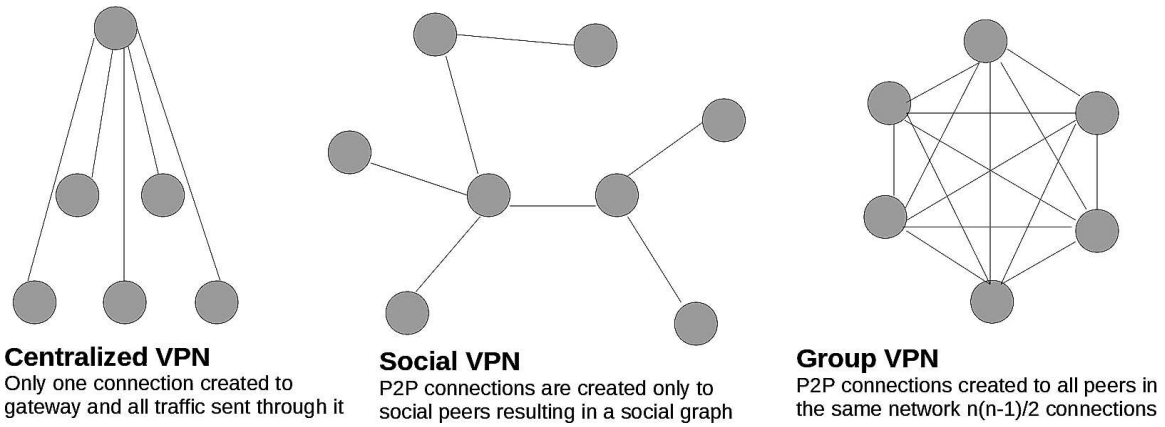



Figure 3-3. VPN Topologies

conflict with the local network. Each controller is able to select its own subnet range without coordinating with a centralized entity or controllers. Therefore, in “social vpn” mode, each controller can select a different subnet for their network; meaning that IP addresses are only valid locally. This is of crucial importance for IPv4 addresses where the virtual address space is limited and can lead to IP conflicts. Since each user defines their own network, they can freely select IP addresses without fear of network subnet collisions. The datapath module performs IP packet translation on incoming packets which ensures no IP conflict following the approach described in previous work [67]. For example, Alice maps her mobile phone to 172.31.0.1 and maps Bob to 172.31.0.2. On his mobile device, Bob’s controller maps his mobile device to 192.168.0.1 and maps Alice to 192.168.0.2. Hence, Alice is able to reach Bob’s mobile phone using the 172.31.0.2 IP address and the IP translation performed by the datapath module on Bob’s phone would make it seem as if the request came from 192.168.0.2. The proposed design also readily creates a private IPv6 address space and pseudo-randomly assigns IPv6 addresses to nodes in the network. Since the IPv6 address space is so vast, no IP translation is necessary due to much lower probabilities of collisions in the virtual IP space.

## 3.3 Implementation

The current TinCan implementation reuses existing technologies and infrastructures that enable P2P connections for both SIP and WebRTC standards by leveraging Google's libjingle [55] P2P library to create private TinCan links. XMPP servers (possibly federated) serve as the discovery/notification overlay. By using STUN and TURN servers for reflection and relaying which are Internet services already freely accessible, users can deploy their own VPNs without any additional infrastructure.

### 3.3.1 Endpoint-Hosted Components

**Packet capture/forwarding.** The datapath packet capture/forwarding module is written in C/C++ and currently runs on Linux, Android, Windows and OpenWRT. Through the TUN/TAP kernel driver, TinCan is able to receive and send Ethernet frames to the vNIC. TinCan uses libjingle [55], leveraging its adoption in existing software (e.g. the Chrome browser). While the typical use of libjingle is for audio/video streaming in WebRTC, TinCan uses it to tunnel virtual IP packets.

**Controllers.** The controllers are written in Python and run as a separate process on the local machine. Controllers access the datapath module's API through a JSON-RPC interface over a UDP socket. The controller uses the API exposed by the datapath module to:

- Register with credentials to the XMPP overlay
- Setup the local vNIC with IPv4/IPv6 addresses, and netmask
- Create/Delete a TinCan link over jingle
- Map an IP address to a TinCan link
- Query the state of a TinCan link
- Handle notifications received through the XMPP overlay (e.g. new node presence, request to connect)

- Handle notifications received from the data path module (e.g. to forward virtual IP packets to other controllers when the destination is not mapped to a local TinCan link)

Through the API, one can extend TinCan to support various combinations of policies and deployments based on anticipated use cases.

### 3.3.2 Internet-Hosted Components

**XMPP Notification Overlays.** The messaging overlays play a crucial role in providing access to the network and as well as serving as a trust anchor for signaling and bootstrapping private TinCan links. The XMPP protocol accomplishes this role by securely routing XML messages through user authenticated TLS connections (see figure 3-4). Hence, TinCan-based VPNs are able to utilize public XMPP providers (such as Google Hangouts or Jabber.org), as well as use their own XMPP service (e.g. an ejabberd server) if they desire that level of control.

**STUN and TURN Servers.** For the reflection and relay servers, the STUN and TURN protocols are used, respectively. These technologies are used in the SIP/WebRTC communities to enable P2P connections for audio and video conferencing; as a result, there are many publicly available STUN servers that TinCan can utilize when creating P2P connections. For some nodes behind symmetric NATs or restrictive firewalls, an XMPP server and STUN server may not be enough to bootstrap a TinCan link; therefore, less than 10% of the time [68], these nodes require the assistance of a relay server to help proxy their TinCan connections. The TURN standards [52] provide such a relaying capability. Google Hangouts is an example of an existing service that already provides such a capability for its users; therefore TinCan links can leverage that for connection relaying through libjingle. There are also many open-source implementations of TURN relays. This work uses one of those implementations [69] for experimentation.

### 3.3.3 Bootstrapping Private TinCan Links

**Rendezvous through XMPP.** TinCan assumes that the XMPP server is a third party trusted for peer discovery, notification, and exchange of PKI certificates. Users can connect to servers they trust, or deploy their own private XMPP server. All communication from TinCan modules to the XMPP server is encrypted at the socket layer using transport layer security (TLS). A user authenticates herself with the XMPP server and broadcasts a presence probe to all peers (or buddies in XMPP terminology) that are part of their group. Therefore, all of the nodes within the group that are connected to the XMPP server receive the presence probe. Each node in the network periodically broadcasts a ping message to all other nodes in the network every two minutes. The datapath module maintains a list of online peers along with the timestamp of their last XMPP broadcast message. Once peers are able to discover and notify each other through the XMPP server, they can proceed to create trusted TinCan links.

To this end, a connection request is created containing the requester's X.509 fingerprint, a list of endpoints containing private/public IP addresses with port numbers, and security credentials to ensure access control for the connection. The request is then sent to the peer over the XMPP overlay. The recipient replies to the request with a query response mirroring the contents of the request: X.509 fingerprint, list of endpoints, and security credentials. Once both sides have the necessary information, they initiate a TinCan link with each other by sending packets directly to these public IP addresses until a response is received (see figure 3-4). This process follows the Interactive Connectivity Establishment (ICE) RFC [53].

As mentioned earlier, nodes exchange their X.509 certificate fingerprint as part of the connection request/reply messages. To encrypt the link, the libjingle library uses the OpenSSL Datagram TLS (DTLS) protocol with peer certificate verification. Once the certificates have been successfully verified and a symmetric key is derived from

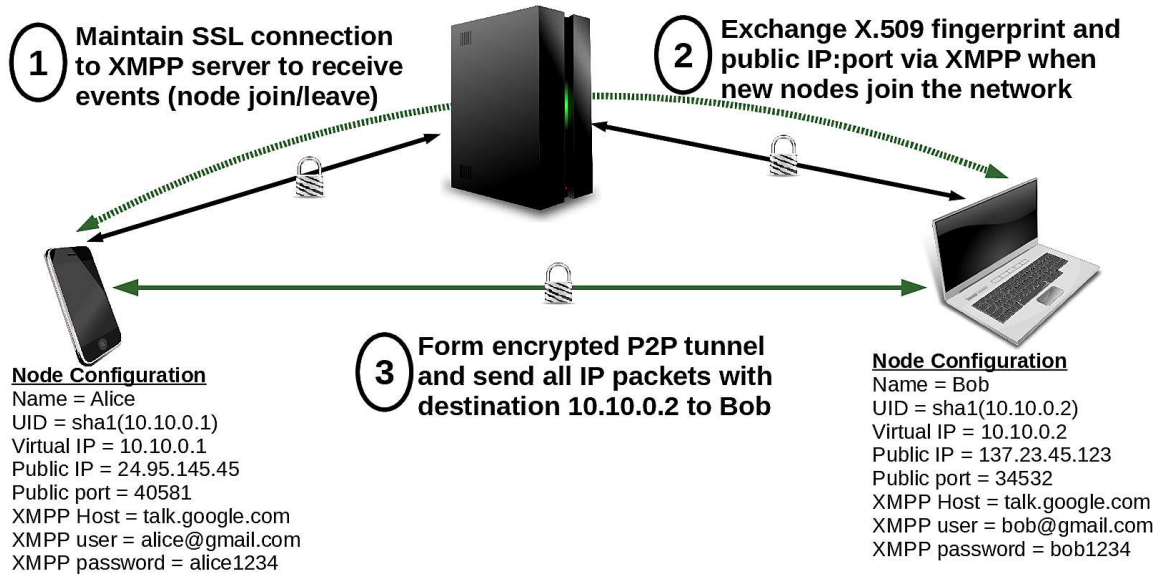



Figure 3-4. Bootstrapping Connections through XMPP

the Diffie-Hellman exchange, the DTLS protocol can proceed to encrypt data flowing through the P2P channel between the peers. IP packets picked by the vNIC interface are encapsulated into data packets sent over the link, and thus protected by DTLS. It is possible to apply IPsec-layer end-to-end security atop of the virtual network overlay as well.

**Rendezvous through Social Graph.** TinCan also supports the bootstrapping of connections through a trusted peer in cases where an XMPP server is not available. In this scenario, well-known trusted peers with public IP addresses can be used as rendezvous points for nodes to initiate TinCan P2P connections (see figure 3-5). Therefore, TinCan nodes can use the social graph for the same functionality as the XMPP servers. Both the XMPP servers and the social graph created by TinCan links can serve the role of a social overlay where users exchange X.509 certificate fingerprints along with additional endpoint information. The public bootstrap nodes can also run additional STUN and TURN servers to help nodes in the connection creation process. By democratizing the STUN and TURN services, the TinCan deployment can become self-sustaining without depending on any external service providers.

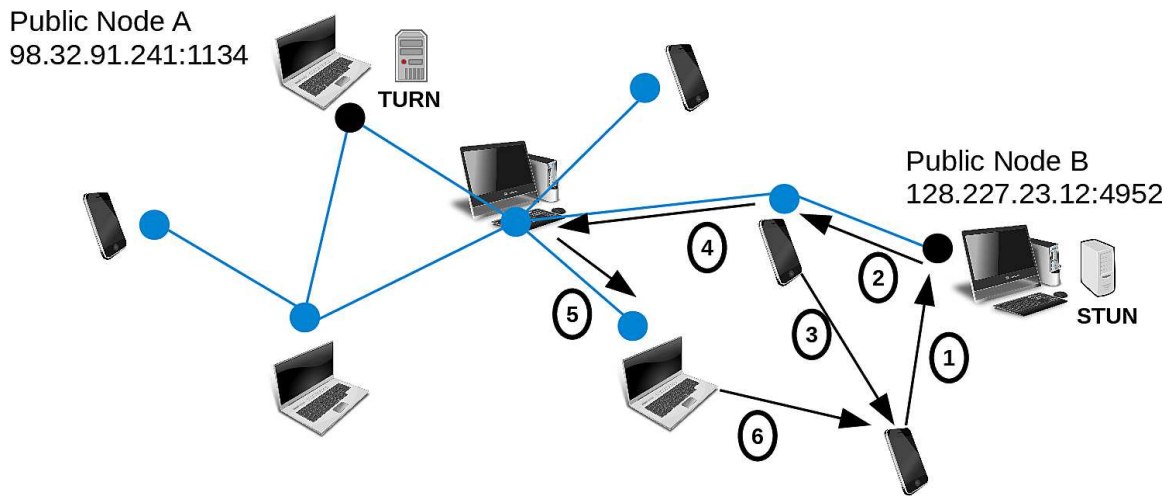



Figure 3-5. Bootstrapping Connections through Social Graph

### 3.3.4 Multi-hop Routing

TinCan's modular design also enables it to support multi-hop routing through controller-to-controller packet forwarding. As mentioned earlier, the TinCan design consists of a datapath module and a controller module. The controller module uses the datapath module's JSON RPC interface to assign P2P links to IP addresses in the network. The controller module also indicates a forwarder for IP addresses that are not mapped to a P2P link. In the current implementation, the controller module tells the datapath module to forward unmapped IP packets to itself. Hence, the controller module can deal with these IP packets according to its policies. For multi-hop routing, the controller communicates with neighboring controllers in the social P2P graph to determine a routing path for IP packets without direct P2P links. If such a routing path exists, the controller forwards the IP packet to the appropriate next hop controller until it reaches its destination. Upon reaching the destination controller, it is injected back to the datapath module allowing it to reach its destined application. Therefore, in this example, the controllers serve as forwarding agents and unmapped IP packets are routed through this forwarding plane until they are correctly delivered. It is important to note that IP packets going through the forwarding plane will have worse performance with respect

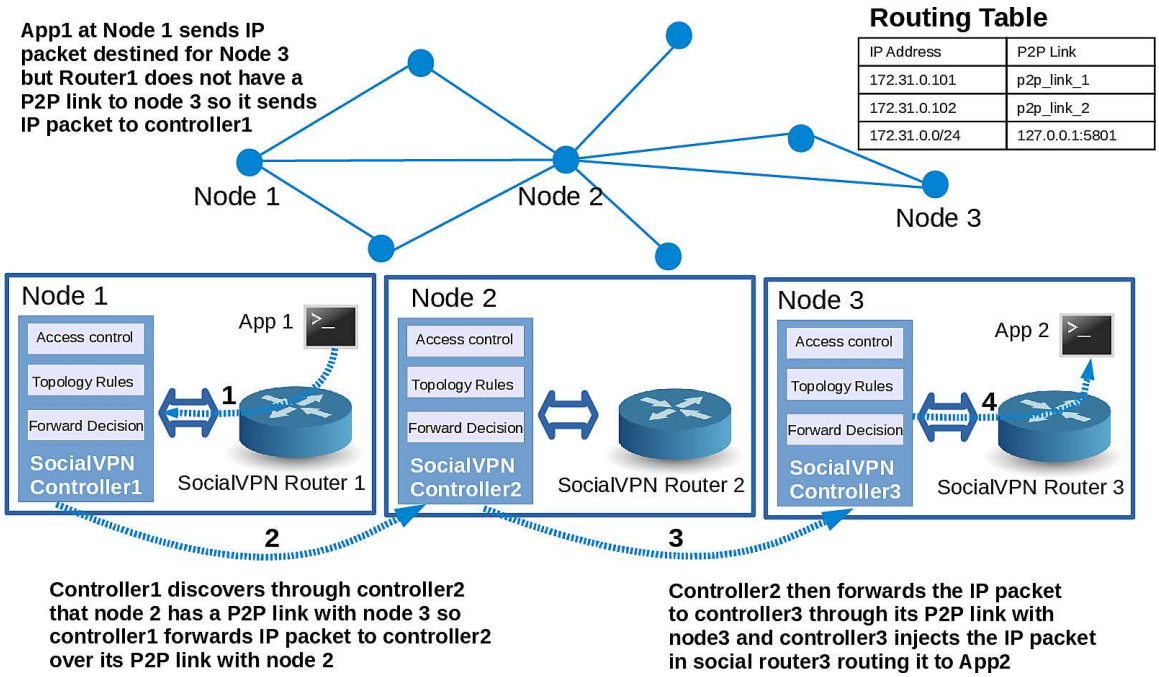



Figure 3-6. Enabling Multi-hop Social Routing

to latency and bandwidth due to the extra processing. If continuous and low latency communication is required, it is more efficient to create direct TinCan connections between the communicating nodes.

## 3.4 Analysis

Various experiments were conducted to understand the resource requirements of the TinCan design. This analysis also focuses on measuring the overhead of maintaining the VPN and packet processing, and the power consumption on mobile devices. In order to make these experiments reproducible, all of the source code is open on Github at http://github.com/ipop-project. To test scalability, a 300-node deployment was setup on FutureGrid [70] using a mix of virtual machines and Linux containers (LXC). FutureGrid is an experimental Infrastructure-as-a-Service (IaaS) cloud environment that is available for academic research.

By running a 300-node experiment, it is possible to analyze the bandwidth usage on the XMPP server, as well as the maintenance cost of managing TinCan P2P links. Rather than reuse existing infrastructure such as Google XMPP and STUN servers, for

these experiments, independent *ejabberd* XMPP and STUN servers were deployed in order to have greater control over the testing environment. This experiment consisted of 8 virtual machines (VMs) running Ubuntu 13.10. One VM ran the notification overlay service, the *ejabberd* [54] open-source XMPP server was used. Another VM hosted the reflection and relay servers, another open source implementation of the TURN protocol was utilized to enable these services [69]. For these two deployments, the bandwidth load on the XMPP, STUN and TURN servers is summarized in Table 3.4.

Each of the remaining 6 VMs ran 50 instances of the TinCan implementation through the use of Linux containers (LXC [71]) which is a lightweight virtualization technology. Using LXC allows for more efficient utilization of resources because it makes it possible to simulate a 300-node network without needing to use 300 VMs or personal devices. The LXC environment is configured to create an isolated virtual network for the containers residing in the same VM; these containers are then able to connect to the outside world through the IPtables symmetric NAT. Therefore, the nodes running on different VMs have to rely on the relaying service (TURN) because the symmetric NATs do not allow for UDP hole-punching thus precluding direct TinCan P2P connections. In practice, typical usage scenarios are unlikely to be as constrained by symmetric NATs, nor is the use of a proactive all-to-all policy recommended for all but small-scale VPNs, since it does not scale well.

For this experiment, the “social vpn” controller was utilized to represent the use case where end users would like their personal devices (e.g. desktops, laptops, tablets, smartphones) along with their friends’ devices to belong to the same SocialVPN and thereby having secure network access to each other. In this model, social relationships are mapped to TinCan VPN connections; for example, if Alice has a friend Bob, then if they run TinCan in SocialVPN mode on their devices, these devices will automatically join each other’s social virtual private network. Hence, the TinCan P2P links in the SocialVPN mode will resemble the edges of a social graph because each VPN link

Table 3-1. Experimental Setup Summary

| Parameter | Value |
|---|---|
| Number of VMs | 6 |
| Number of Containers per node | 50 |
| Number of nodes | 300 |
| Number of connections | 1475 |
| Bandwidth Cost for TinCan connection | 1 KB/s |
| Average Traffic at XMPP server | 19 KB/s |
| Average Traffic at STUN server | 27 KB/s |

represents a social link (see figure 3-3). To simulate this social graph environment, the Barabasi-Albert model from the NetworkX graph library [72] generated a 300-node graph with 1475 edges (or TinCan links).

### 3.4.1 Bandwidth Costs

During this deployment, the average bandwidth consumption at the XMPP server is about 19 KB/s; this shows that the Tincan protocol incurs very little traffic on the XMPP server. This traffic is primarily the periodic ping messages that each node in the network send to each other to indicate that they are still alive. In the case of the reflection (STUN) server, the TinCan implementation running on the end-nodes sends a 64-byte STUN *binding request* and receives a 72-byte STUN *binding response* every 15 seconds per connection. Therefore, the bandwidth cost on the STUN server for supporting the deployment of 1475 TinCan connections is about 27 KB/s (or 0.18 KB/s per connection). Also there are numerous freely accessible STUN servers on the web hosted by Google and others meaning that these resources can also be leveraged for the reflection service. Table 3.4 summarizes the bandwidth costs of the deployment.

In order to calculate the bandwidth cost on the TURN relay server, it is important to understand the maintenance cost of each TinCan connection. Libjingle sends a STUN *user request* every 500 ms and expects a *Success Response*; the average size of these packets is 130 bytes. These ping packets help libjingle keep track of the state of the TinCan link in terms of latency, jitter, and link failure. Therefore, each TinCan connection consumes about 1040 bytes per second as connection maintenance overhead. When

Table 3-2. VPN Network Performance

| | Latency | TCP | UDP |
|---|---|---|---|
| LAN | 0.5 ms | 325 Mbps | 320 Mbps |
| TinCan DTLS | 1.07 ms | 64 Mbps | 47 Mbps |
| TinCan no DTLS | 1.07 ms | 84 Mbps | 128 Mbps |

a TURN server is used for relaying, these ping messages are routed through the relay. According to Google research, about 10% of P2P connections require a TURN relay and therefore, supporting a 300-node network with 1450 edges would necessitate a relay service for about 145 connections costing about 145 KB/s for connection maintenance. It is important to note that the TURN service would also have to relay IP traffic between the nodes that it supports and therefore deploying a TURN service requires thoughtful planning and proper access control. TURN implementations provide user authentication making it possible to identify different connections and apply bandwidth limitations per user. For instance, it is possible to configure a TURN server to only allow a maximum of 50 KB/s throughput per connection and limit the number of connections. However, since the relay service is required in the face of symmetric NATs (i.e. less 10% of the time) it is possible to support up to 1000-node network on a single TURN server. There are also commercial offerings such as turnservers.com that provide this relay service for a fee if users do not want to deploy their own TURN service.

### 3.4.2 Network Performance

One of the drawbacks of the TinCan design is that, instead of dedicated virtual switches and routers, each node runs their own virtual router that tunnels IP packets to the appropriate TinCan links. Therefore, every IP packet has to be encrypted, decrypted, and translated. This user-level packet processing can greatly constrain network performance. In this experiment, the iperf network benchmarking suite measured the maximum bandwidth achievable by TinCan between two nodes in the same gigabit LAN. As shown in table 3.4.1, TinCan achieves 64 Mbps for TCP and 47 Mbps for UDP with DTLS encryption, but without encryption the bandwidth increases to 84 Mbps for TCP

and 128 Mbps for UDP. A possible optimization for LAN environments is to bypass the overlay and allow TinCan nodes in the same LAN to directly route packets to each other without encryption, as described in [73]. TinCan supports this router-mode of operation; in this mode, containers or VMs on the same host can all share a single instance of the TinCan router. Local nodes can therefore communicate directly with each other and only use the TinCan pathway to private connect with remote nodes outside of their LAN. It is also possible to run TinCan in OpenWRT-enabled routers, this approach would also make it possible for local nodes to communicate directly without the overhead of the local processing and they would also have connectivity to nodes in the P2PVPN since the OpenWRT router would now have a connection into the network.

Understanding the time it takes to create a TinCan connection is crucial in designing a controller when considering a proactive connection policy versus an on-demand connection policy. As shown in figure 3-7, the median connection setup time is about 6.3 seconds, with the 75% percentile at 7.8 seconds but in the worst case, it may take up to a few minutes to bootstrap a connection due to dropped connection requests. Therefore, it may not be ideal to use an on-demand connection policy if an application generates bursty traffic that is sensitive to high latency start-up times.

### 3.4.3 Encapsulation Overhead

The proposed design incurs packet overhead due to the additional headers necessary for IP encapsulation. Another source of overhead is the selection of a relatively small MTU for the vNIC. Ethernet devices typically have an MTU of 1500 bytes, but using an MTU of 1280 bytes minimizes the probability of UDP packet fragmentation. Moreover, the TinCan implementation uses a 40-byte header for each packet consisting of a 20-byte source unique identifier (UID) and another 20-byte destination UID. The 160-bit UIDs creates an extra level of indirection which facilitates packet routing in the network. Consequently, a small MTU and the extra header has an adverse impact on network performance.

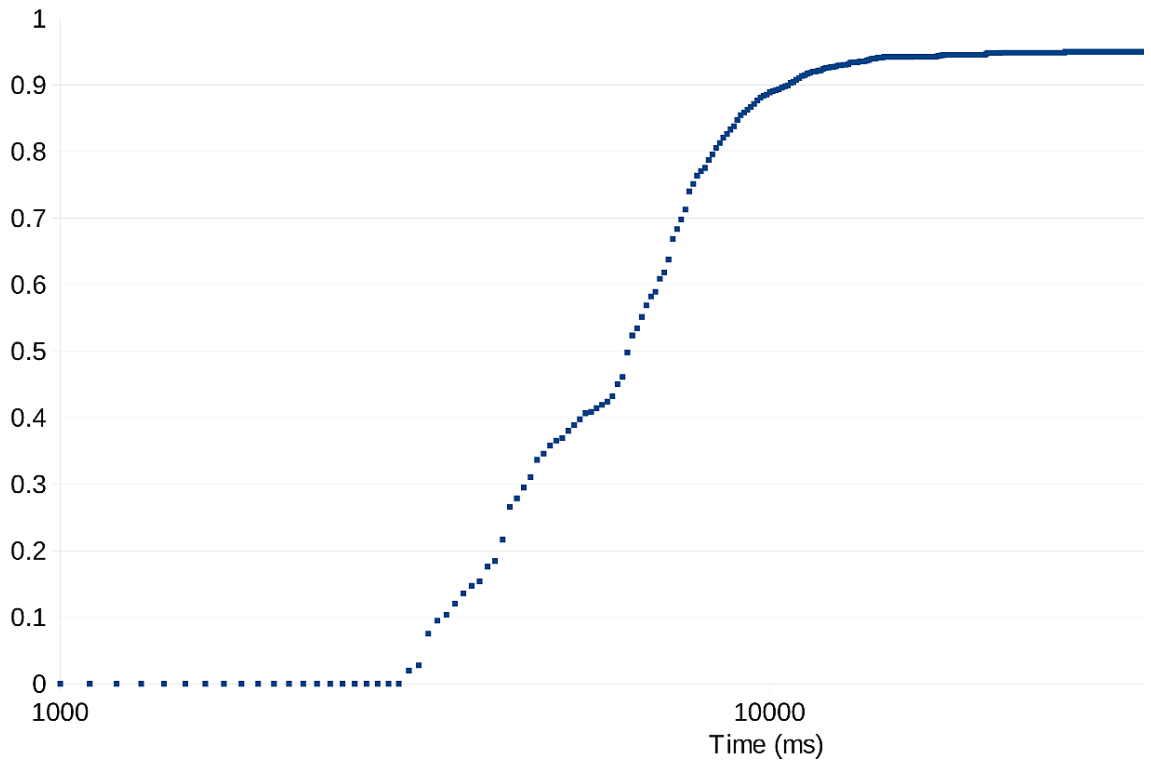



Figure 3-7. CDF of 1450 Connection Times. 75% of connections take less than 8 seconds.

The following experiment quantifies the network overhead. For this setup, there is a TinCan network of just two nodes, a Samsung Galaxy Tab 10.1 and an Ubuntu 12.04 workstation. The tablet has a 1GHz dual-core Nvidia Tegra 2 processor and 1GB of RAM and the workstation is a 3.0GHz Intel Core 2 Duo with 8GB of RAM. By performing file transfers of different sizes as shown in figure 3-8 over both WiFi and TinCan, the results show an average network overhead of 14%. This overhead can be reduced by choosing a higher MTU closer to 1500 bytes and by using a smaller header size (e.g. 128-bit UIDs instead of 160-bit).

### 3.4.4 Mobile Power Consumption

Mobile computing support is an important aspect of the TinCan design; hence, an experiment on the Android tablet, with the number of connections scaled up from 1 to 23, provides insight in the power costs of TinCan P2P connections. PowerTutor [74], a software-based power measuring app available for Android, calculated both the

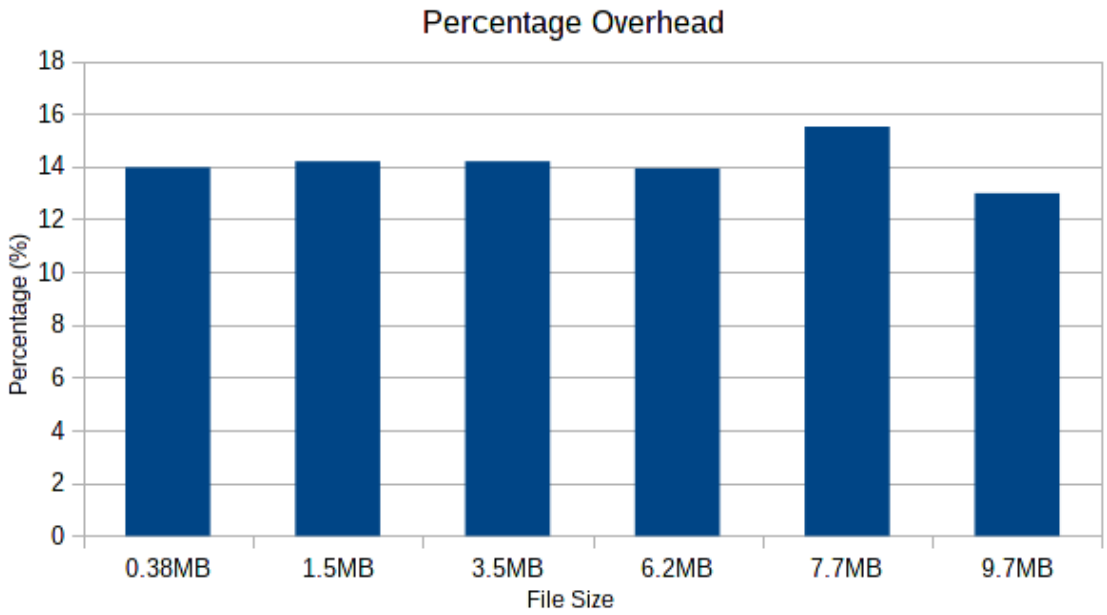



Figure 3-8. File Transfer Percentage Overhead. Due to MTU of 1280 and extra 40-byte header for IP encapsulation, there is a 14% overhead in the extra number of bytes sent over the network for the same file size when compared to WiFi.

WiFi and CPU energy consumption. In terms of CPU energy consumption, figure 3-9 shows a steady increase in energy cost which averages about 0.13 Joules (J) per 5-minute interval ranging from 4.3 J for one connection to 7.2 J for 23 connections. For comparison, the LinPack for Android benchmark on the same tablet consumes 64.7 J for the same time interval. The WiFi energy consumption in figure 3-10 shows a different pattern where there is a sharp energy increase from 1 to 3 connections followed by a steady state. As mentioned earlier, a TinCan connection generates about 8 network packets with sizes around 130 bytes per second consuming about 1 KB/s of bandwidth. The mobile WiFi card is able to handle the bandwidth requirements of one connection in low-power mode. However, once there is more than a single P2P connection, the WiFi enters high-power mode which increases the energy consumption by 1.5x from 144 J to 220 J (for a 5-minute period). Since the WiFi card remains in high-power starting with two connections, there is no significant change in energy consumption as the number of connections increases.

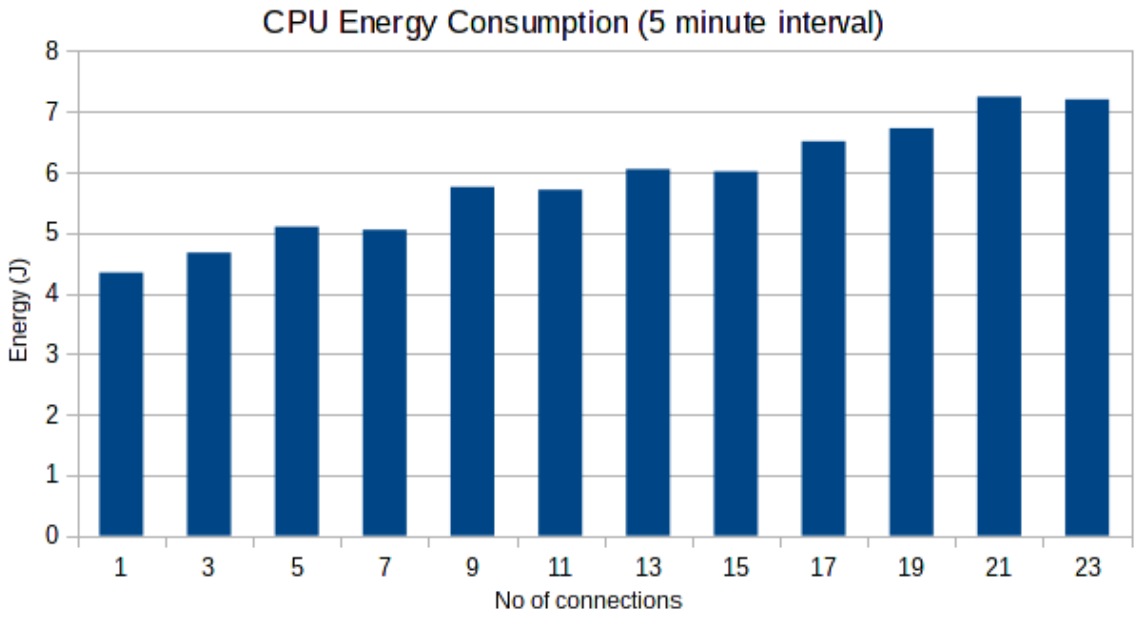



Figure 3-9. CPU Engergy Consumption on Mobile

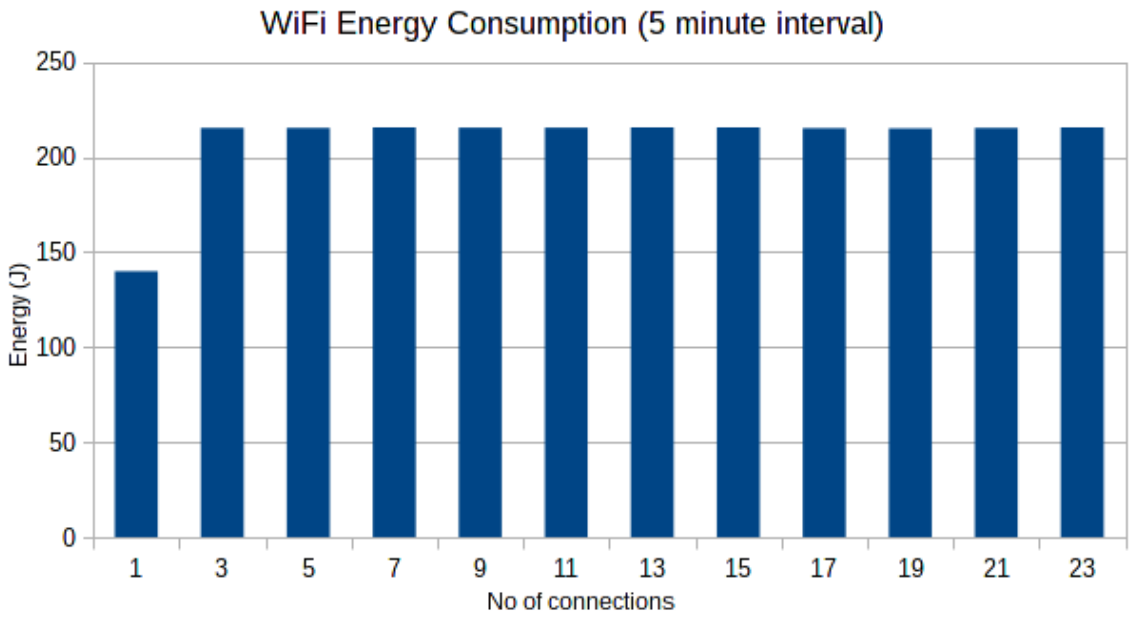



Figure 3-10. WiFi Energy Consumption on Mobile

### 3.4.5 Zero Infrastructure Experiments

The key advantage of the TinCan design is the ability for a user to create their own virtual private network by simply running the software on their end devices or cloud instances and configuring it to use existing XMPP services (e.g. Google.com or Jabber.org). To demonstrate this, a virtual network consisting of two Android devices was created: a Motorola Photon Q smartphone and a Samgsung Galaxy Tab 7. The smartphone was connected via the Sprint 4G network while the tablet connected via WiFi network. Using the Google XMPP servers, the two devices created a SocialVPN and therefore had private IP access to each other as if they are connected on the same LAN. Using the CSipSimple Android app, the two devices could perform SIP calls between each other. The call was performed by simply using the devices' virtual IP addresses as the SIP address (i.e. sip@172.31.0.101). Consequently, secure SIP calls were conducted over TinCan IP links through both the 4G ISP firewall and WiFi NAT without any registration or signaling through a SIP server. By leveraging Google's XMPP service along with the dozens of publicly available STUN servers on the web, users can easily get private IP connectivity to each other at no cost.

## 3.5 Summary

Cloud and mobile computing have created a need for more user-defined overlay virtual networks which enable both node mobility and information security. The proposed TinCan design leverages existing overlay and P2P technologies such as XMPP, STUN, and TURN thereby creating a solution where users can define and deploy their virtual networks without needing additional infrastructure. Additionally, unlike other existing solutions, this approach does not require special access to the hypervisor, nor do users have to configure virtual switches and routers. To provide layer 3 connectivity, each node in the network runs their own virtual router which maps IP addresses to TinCan connections. Analysis of the TinCan design shows that a network of 300 nodes incur acceptable bandwidth loads on the XMPP, STUN, and TURN servers. The

experiments also show that it takes less than 10 seconds to create 75% of TinCan P2P connections. The additional headers for IP encapsulation and smaller vNIC MTU cause a 14% network overhead. In terms of mobile power consumption, it seems ideal to only maintain one TinCan connection at a time to avoid running the WiFi card in high-power mode.

The evaluations consider the overheads associated with a single link, and for small-scale VPNs that could be deployed with a very simple topology and connection policy. These are feasible for small-scale VPNs, e.g. for small virtual clusters. For VPNs scaling to larger number of nodes (100s to 1000s), it is clearly a requirement to reduce the number of links in order to reduce traffic at the notification/discovery and reflection services. One approach that scales well and has been used in previous work is to use a structured P2P routing overlay with on-demand shortcut connections [64]. The choice of a scalable overlay approach can be encoded in the logic embedded in the controller. Future work will consider different topology options (e.g. different structured P2P approaches, as well as social and random graphs) and different policies for on-demand link establishment/tear-down.

# CHAPTER 4
# ENABLING USER-DEFINED DOMAIN NAMES

In the previous chapters, this dissertation advocated for an environment where social peers have private IP access to each other. Social networking application developers are thus able to leverage that connectivity to create more secure social applications. However, a mechanism to efficiently discover these private IP addresses is still necessary especially since these IP address are dynamic. This chapter explains a solution that solves this problem through the use of the domain naming system (DNS). By adding a DNS service to the architecture, developers and users are able to use predictable canonical domain names instead of dynamic IP addresses to initiate connections between social peers. Domain names provide location transparency and serve as more stable endpoints than private IP addresses.

This chapter presents SocialDNS, a decentralized, naming service for P2P VPNs (e.g. SocialVPN). P2PVPNs provide users IP access to each in a decentralized fashion; however, there does not exist a decentralized solution which allows end users the freedom to choose fully qualified domain names for their services. Decentralized solutions such as multicastDNS (MDNS) [24] and WINS [25] exist for private networks and LAN environments; however, these solutions cannot be applied to the P2PVPN environment unmodified. This need is addressed in SocialDNS by providing an alternative comparable to decentralized solutions such as multicastDNS but better suited for P2P VPNs.

With SocialDNS, P2PVPN users are able to select short-names among themselves for their resources through social scope uniqueness instead of the global uniqueness enforced by the normal DNS system. Name conflicts can arise in the SocialDNS system if two peers decide to choose the same domain name for a resource. In such cases, SocialDNS use a simple rank-based method to select the mapping with the highest popularity in the social circle.

A social graph analysis of the design is provided with estimates for the expected bandwidth costs, and latency. The analysis is based on the assumption that P2PVPNs form a social graph with small world characteristics since each VPN link represents a social relationship. Although not all P2PVPNs possess this property, the focus is only on P2PVPNs that only created VPN links based on social relationships. Based on this assumption, the analysis is based on a 100,000 social networking graph from Orkut to validate various design choices.

## 4.1 Background and Related Works

The main motivation for SocialDNS is to provide end-users with the freedom to set their own domain names in P2PVPN environments. Domain names serve an important role in the user-friendliness of the Internet and are used in most TCP/IP connections. P2PVPNs make it possible for end users to host services on their personal resources and provide network level access to peers of interest. For example, an end-user, Alice, can host a blog from her laptop that only her P2PVPN friends can access, or share her desktop through a VNC session to do a PowerPoint presentation. An important requirement for hosting services is user-friendly domain names to these services so that Alice's colleagues can connect to her blog by typing *aliceblog.sdns* or view her presentation by typing *alicepc.sdns* in their remote desktop clients. Domain names also provide location transparency in dynamic IP environments such as private networks, and P2PVPNs; thus end-users are not required to re-discover the dynamic IP address to a service every time there is a change in the host's IP address. In the previous example, without a domain naming service, Alice's friends would have to discover Alice's IP address every time they want to access her blog or view her presentation. Hence, IP connectivity is not the only requirement for enabling users the freedom to host their own content, a domain naming system is also necessary so that end users can select the names used to refer to these services.

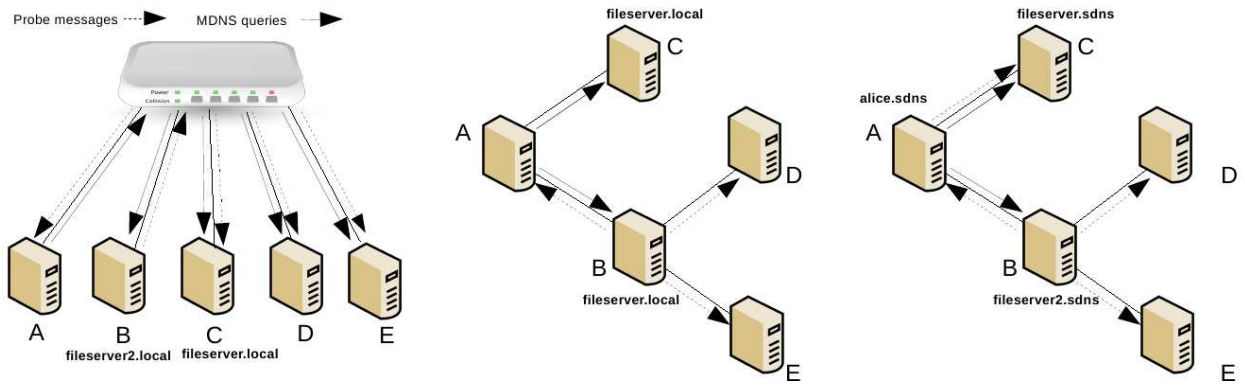



Figure 4-1. **On left: MulticastDNS in LAN environment.** All-to-all connectivity among nodes allows multicast DNS to successfully detect duplicate names, hence host B chooses domain name *fileserver2.local* because host C's mapping of *fileserver.local* is discovered in through a probing phase. **In middle: MulticastDNS in P2PVPN environment.** Lack of all-to-all connectivity causes host B to be unaware of host C's mapping of *fileserver.local* and thus claims the same mapping. This creates a conflict for host A who now has two peer in her network with the domain name of *fileserver.local*. **On right: SocialDNS in P2PVPN environment.** SocialDNS uses a two-hop broadcast to search for duplicate names in the social circle. With the two-hop broadcast, host B discovers host C's mapping for *fileserver.local* and chooses *fileserver2.local* instead. If host B still decides to pick the same domain name as host C, then the conflict resolution mechanism picks the most popular name in the social circle.

Decentralized naming services are extremely useful and common in private networks (e.g. LANs) because they provide a zero-configuration solution to mapping user-friendly names to resources. For example, in a typical home network, a decentralized naming service makes it possible to access a file on a Windows machine using the following url `smb:\\mom-pc\SharedDocs\familypic.jpg`. The two commonly available decentralized naming solutions for private networks are the Windows Internet Name Service (WINS) [25] by Microsoft and Apple's multicastDNS [24] system called Bonjour. One approach would be to run one of these naming solutions on the P2PVPN unmodified. However, as shown in Figure 4-1, these approaches were designed with the assumption of all-to-all connectivity amongst all node within the same network through a common networking backbone (e.g. routers and switches). The lack of

all-to-all connectivity in the P2PVPNs makes it impossible for WINS and Bonjour to properly detect name collisions in their probing phase. SocialDNS aims to address these limitations for the P2PVPN environment.

There have been various previous works aimed at improving the resource naming experience. One of the first is by Cox et. al. [75] who discussed the implementation of a DNS system on top of the Chord DHT. Although such an approach was feasible, the latencies of a DHT-based DNS were much higher than conventional DNS. SocialDNS is not dependent on a structured DHT and is deployed on top of an unstructured social peer-to-peer network through the use of P2PVPNs.

Walfish et. al. [76] suggested using semantic free references (SFR) as a replacement for DNS-based URLs on the web. They also suggested using a structured DHT to store self-certifying o-records containing pointers to resources. Their focus was on decoupling the mnemonic names from the actual references which were 160-bit hash tags, and a whole separate mechanism, maybe social, to map names to the complex tags. The SFR design therefore suggested a total redesign of referencing on the Internet. CoDNS [77] is a system which automatically forwards pending DNS requests to another local DNS server to a remote administrative domain. The rationale is that most DNS failures are associated with poorly configured local DNS server, by sending long pending requests to different local DNS server in another domain, it provides some redundancy to local DNS failures. The SocialDNS system reuses the current DNS protocol without requiring any re-architecting and simply supplements current DNS systems. SocialDNS.net [78] is a project similar to dynamic DNS which allows users to manage their own domains through the use of a browser-plugin that can redirect urls prefixed with go:// to a service that translate to a proper HTTP url. This approach is based on a client-server architecture while SocialDNS uses a peer-to-peer design.

Allman [79] introduced the concept of personal namespaces (*pnames*) to provide easier references to their email, blogs, links, and so on. Every user is given an NID

which is a cryptographic hash of the public key. These NIDs are shared among friends through a one-time exchange and are used to retrieve the mappings of a user's namespace from a DHT. The *pnames* system share many similarities with SocialDNS, but it is proposed as second level of indirection to name resolution and the resolution are not restricted to DNS records. For example, *alice:email* would resolve to *alice@mailserver.com*, a second resolution would then be required to resolve *mailserver.com*. Since NIDs are always unique, and the local user defines the mapping of names to NIDs, name conflicts are not an issue. SocialDNS has to deal with name conflicts and uses a social conflict resolution to help rank various names.

## 4.2 Design

The main design goals of SocialDNS are short-names through social scope, decentralization, simple user management, and name conflict resolution through social popularity. The SocialDNS design is also based on the following assumptions: 1) a P2PVPN creates a social graph where links represent relationships between ends users, 2) each P2PVPN tunnel is an authenticated and encrypted end-to-end link similar to an IPSec connection, and 3) the previous two assumptions makes it harder for a malicious user to mount a Sybil attack. The third assumption is also corroborated by previous work [80] which have shown that Sybil attacks can be mitigated using social links instead of anonymous links. The authentication and encryption of the IP tunnels are discussed in previous chapters.

### 4.2.1 Enabling Short Domain Names

One of SocialDNS's key roles is to enable short domain names to resources in the P2PVPN, for example, Alice can set *alicepc.sdns* as the domain name for her personal computer. The first step in enabling short domain names is to choose an unallocated root-level domain zone to avoid conflicts with the global DNS. SocialDNS chooses an unassigned root-level domain zone; therefore preventing phishing attacks based on addresses used on the public Internet i.e. by creating a fake DNS mapping for

*www.bankofamerica.com*. A second requirement is to limit the scope for uniqueness. The global DNS system provides naming for the whole world, potentially billions of hosts thus making short-names scare. In the P2PVPN setting, uniqueness is only required within a peer's social circle which is usually composed of a few hundred or at most a few thousand hosts. The reduced scope lessens the probability of collision for short-names such as *alicepc.sdns*.

### 4.2.2 Decentralization and Broadcasting

Decentralization in SocialDNS is achieved by running a local DNS service on each peer's machine; hence there is no head node or central point of failure. SocialDNS leverages the unstructured peer-to-peer social graph created by P2PVPNs as the messaging substrate connecting these local DNS servers. The local DNS service provides users with an interface where they can easily add and remove DNS mappings. The SocialDNS nodes use one or two-hop broadcast messages to communicate amongst each other thus allowing users to perform arbitrary searches on each other's SocialDNS caches. Using broadcast greatly simplifies the design due to its stateless nature and topology independence. As users create new DNS mappings, they are able to freely share these mappings with social peers over the P2PVPNs. Therefore, a typical DNS query can search the local DNS cache as well as the DNS caches of friends in the social circle. This is analogous to the common Gnutella P2P file sharing paradigm where each peer runs a local file server, and other peers are able to broadcast queries to search for files. In the SocialDNS case, each peer runs a DNS server instead of a file server, broadcast queries are sent to social peers instead of random peers, and the results are DNS mappings instead of files.

### 4.2.3 Simple User Interface and Management

Managing a typical DNS server such as BIND is a serious undertaking, a task suited mainly for network administrators [81]. The focus is to provide simple management to the user with minimal configuration and an intuitive Web interface for creating,

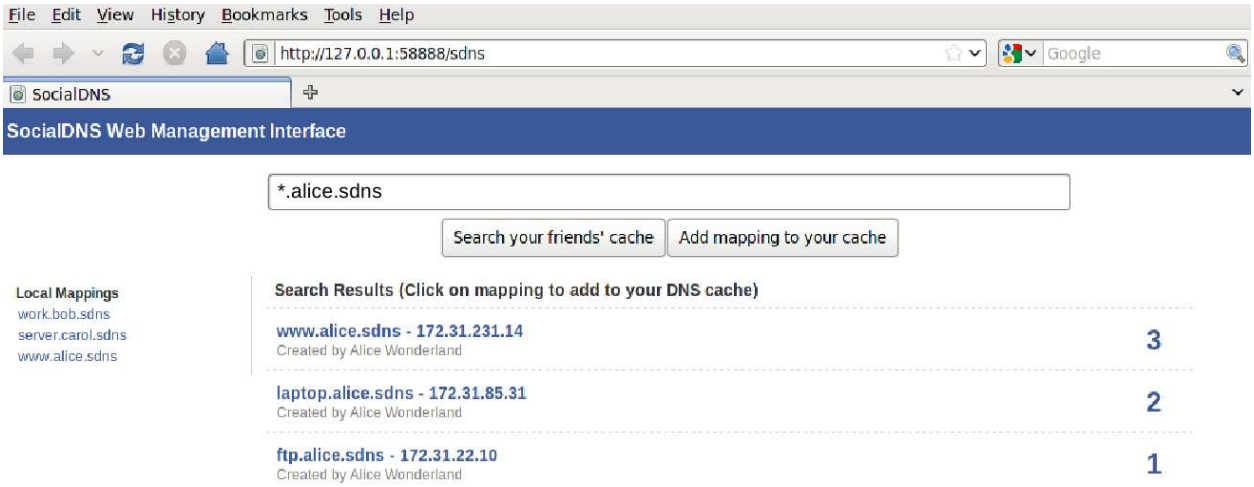



Figure 4-2. SocialDNS AJAX Web Interface. In this screenshot, Bob does a wildcard search for all DNS mappings with the **.alice.sdns* search string. By clicking on the *Search your friends' cache* button, the search is sent to all friends through the P2PVPN. As the responses arrive from friends, the Web interface is updated along with a ranking for each mapping on the right. The ranking information is used to sort the mappings, the end user makes the final selection on the mapping to add to the local cache. Although IP addresses are shown in the interface, the user is not involved in the actual manipulation, when IP addresses are changed, mappings are updated automatically. Existing local cache mappings are shown on the left hand side.

deleting, searching, and sharing DNS mappings. The user experience for the SocialDNS system involves just a few steps. First, by running the SocialDNS service on the local machine, startup scripts configure the operating system's settings and point to the local server as one of the DNS servers. The local DNS server only resolves requests under the *.sdns* root domain zone to avoid common DNS phishing attacks and to allow co-existence with the global DNS. The user then uses their web browser to access the SocialDNS interface (see Figure 5-3).

**Creating User-Defined Mappings.** Users can add new mappings to their local SocialDNS cache in one of two ways. The first method is for the user to manually create a SocialDNS mapping through the Web interface: for example, Alice uses the input box to map the name *alicepc.sdns* to her local PC. The newly created mapping will only be accepted if it follows these criteria: 1) the mapping ends with *.sdns*, and 2) it points to a

virtual IP address in the P2PVPN address range. SocialDNS gives the user the freedom to pick any DNS mapping under the *.sdns* root zone.

**Importing Mappings through Search.** The second method is through a SocialDNS search which allows users to query each other's SocialDNS caches. The searching process resembles a typical Web search, and the results are presented to the user sorted by the ranking method described below. A user can choose to import a mapping to his/her SocialDNS cache by simply clicking on the mapping.

Figure 5-3 shows a search for **.alice.sdns* and the results of that search and the user can choose to import these mapping locally. Since this search is done as a typical flood-based broadcast in an unstructured P2P network, SocialDNS therefore supports various types of queries such as exact, nearest, and regular expression matching. The search is done by broadcasting the DNS query to all friends and perhaps friends of friends depending on user settings. Using broadcast queries with greater hop counts gives the local user a more accurate view of the popularity of a DNS mapping at the cost of generating more traffic and higher latencies.

**Freedom to Redefine Domains.** SocialDNS also gives users the option of redefining the domain name that point to a friend's resource in his/her local DNS cache. This is similar to creating a bookmark to a web page, but instead of using the page title provided by the website, the user can create his/her own reference to that web link. SocialDNS operates in a similar fashion because the user can import the existing mapping defined by a friend or create another mapping at his/her discretion. For example, let's say Bob chooses *bobpc.sdns* as the SocialDNS name for his machine, Alice can either import that domain mapping or define her own mapping such as *bobby-pc.sdns* for Bob's machine. Consequently, SocialDNS allows a user to create map multiple domain names to the same IP address.

Creating a different mapping for a peer's resource only makes that mapping available to the local SocialDNS cache; however, peers have the option of importing

that mapping in their own SocialDNS cache through the search interface. Extending the previous example, assuming Carol has friendships with both Alice and Bob, through a SocialDNS search, Carol noticed that Bob the following mapping *bobpc.sdns = 172.31.231.23* while Alice has *bobby-pc.sdns = 172.31.231.23* and both mappings point to the same IP address, Carol will have the option of choosing either Bob's mapping, or Alice's mapping, or both. Once again, DNS mappings are analogous to bookmarks in webpages, and SocialDNS makes it easy to create any mappings and share it with friends. As a mapping is replicated across peers' SocialDNS caches, its popularity in the social circle increases and that creates the basis for the rank system described below.

### 4.2.4 Name Resolution

SocialDNS performs two types of name resolution: user-verified and automatic. Providing an automatic mode gives the end user the option of not having to manually import each mapping created by social peers, but supporting a user-verified mode is crucial for some secure services. The mode of operation is a configuration the user is able to set at startup or runtime.

**User-Verified Name Resolution.** In user-verified mode, SocialDNS only resolves mappings that have been explicitly created or imported by the user through the web interface. This is important because changing a DNS mapping from $\mathrm{IP1}$ to $\mathrm{IP2}$, which can happen in automatic mode without user intervention, may cause information leakage due to web browser cookies. Hence, users should use the SocialDNS in user-verified resolution mode for information sensitive services.

For a user-verified name resolution, the local DNS server utilizes its SocialDNS cache only to resolve incoming DNS queries from the operating system. In other words, when Bob types *www.alice.sdns* in his browser, the browser asks the operating system to resolve the DNS name, the request is forwarded the local SocialDNS service, the mapping is looked up in the local cache; if found, the IP address is returned, if not found an NXDOMAIN response is sent back to the application. In this mode, there are no

name conflicts because they are resolved by the user through the web interface. For example, if Alice tries to create or import a mapping such as *www.alice.sdns* which already exists in her local SocialDNS cache, she will be informed of the collision and required to choose a different domain name for the resource. Also, in the web interface, if there can be multiple search results with the same ranking; in this case, the user makes a selection on the mapping they would like to import thus resolving the name conflict.

**Automatic Name Resolution.** In automatic mode, SocialDNS automatically searches the social circle for mappings that are not present in the local cache and picks the highest ranked mapping. This mode of operation is not as secure as the previous mode because it can lead to frequent changes in IP address that is transparent to the user because the resolution is based on the most popular mapping at the time of the query. To perform this resolution, the SocialDNS system sends a search to all friends, waits for a predefined time interval to gather results, and the highest ranked mapping is chosen. In the case of a tie, one of the SocialDNS mapping is randomly selected.

**Ranking Domain Names.** SocialDNS does not enforce uniqueness during the creation of the DNS mappings due to the use of social context to help with the DNS resolution; therefore, *www.alice.sdns* can map to one IP address in one social circle, and to a totally different IP address in another social circle. In the case where both mappings exist in the same social context, a ranking algorithm is used to provide preference to one mapping over another. So *www.alice.sdns* will map to the IP address with the highest ranking, and the local user will have to specify an alternate name for the conflicting mapping.

The current ranking algorithm is simple; when a DNS query is sent to all friends, the friends search their DNS cache for matching results and send the responses back to the requester. Mappings are ranked based on an aggregation of the responses. For instance, if five friends return responses saying *www.alice.sdns* maps to 172.32.122.31,

while two friends says that it maps to 172.15.223.112, then the first mapping will get a ranking of five, and the second a ranking of two. Once ranked, the mappings are presented to the local user for selection (see Figure 5-3). With this simple scheme, the framework uses the presence of a SocialDNS mapping in a peer's local cache as a vote for that mapping, the more peers that have a mapping in their cache, the more votes that mapping gets. This is similar to the ranking system used by Delicious.com where a bookmark's popularity increases as more people add that bookmark to their accounts.

### 4.2.5 Protection against DNS attacks

Attacks such as phishing, session hijacking, cache poisoning have plagued DNS and it requires careful administration and robust software to protect against the DNS security flaws. Hence, it is important for the SocialDNS design to avoid these same flaws currently plaguing the current DNS system. SocialDNS, however, benefits from inherent network level security provider by P2PVPNs. In a P2PVPN each tunnel is encrypted and authenticated either through the use of IPSec, or a TCP/IP level encryption such as TLS or DTLS. Therefore, every network packet can be bound to a peer's identity making it very difficult to spoof an IP packet or a DNS response. This security primitive hence makes attacks such as phishing and cache poisoning futile because the culprit can be detected and banned for the P2PVPN.

## 4.3 Analysis

The focus of the analysis is the exploration of the following aspects of the design: number of peers in the social scope, bandwidth cost, and latency. Due to the assumption that the P2PVPN create a social graph, the analysis is based on a 100,000 node social graph dataset captured from Orkut. The dataset was provided by the authors of [35]. NetworkX [82], a Python package for complex network analysis, was used to study the different aspects of the design through the social graph. The social graph contains the following small-world characteristics: 1) an average clustering coefficient of 0.27, 2) and a powerlaw degree distribution.

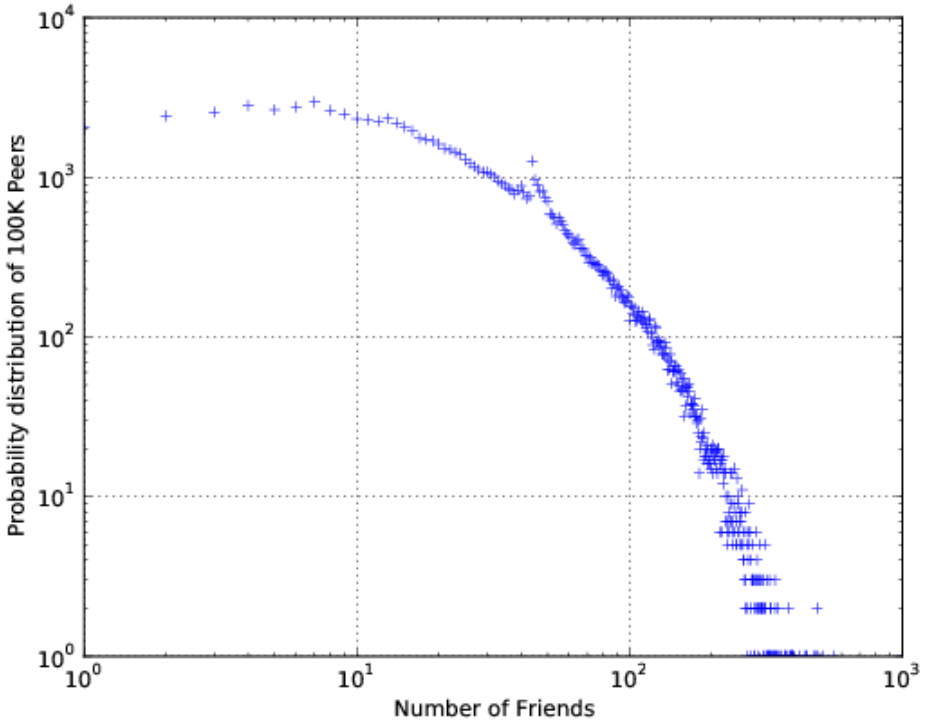



Figure 4-3. Distribution of Number of Friends in 2-hop Radius of Social Circle. As shown in this distribution, a user has about $2000$ friends in a two-hop radius on average.

### 4.3.1 Reduced Conflicts in the Social Scope

A major strength of SocialDNS is the increased availability of short names through social scoping, meaning because there is no guarantee of global uniqueness as in the case of regular DNS, uniqueness is only guaranteed within the social circle. Since the recommended social scope for uniqueness is two hops, this analysis examines the distribution of the number of friends in a two-hop radius to get an idea for the number of peers that may be competing for the same domain names. The observation is that, in a two-hop radius of the social circle, the average number of friends are 1,832 and a median of 1,150. Therefore, Alice would only have to compete with less than 2,000 peers on average to claim the name *alice.sdns* instead of the billion of users on the Internet to guarantee uniqueness in a two-hop radius of the social circle. The reduced

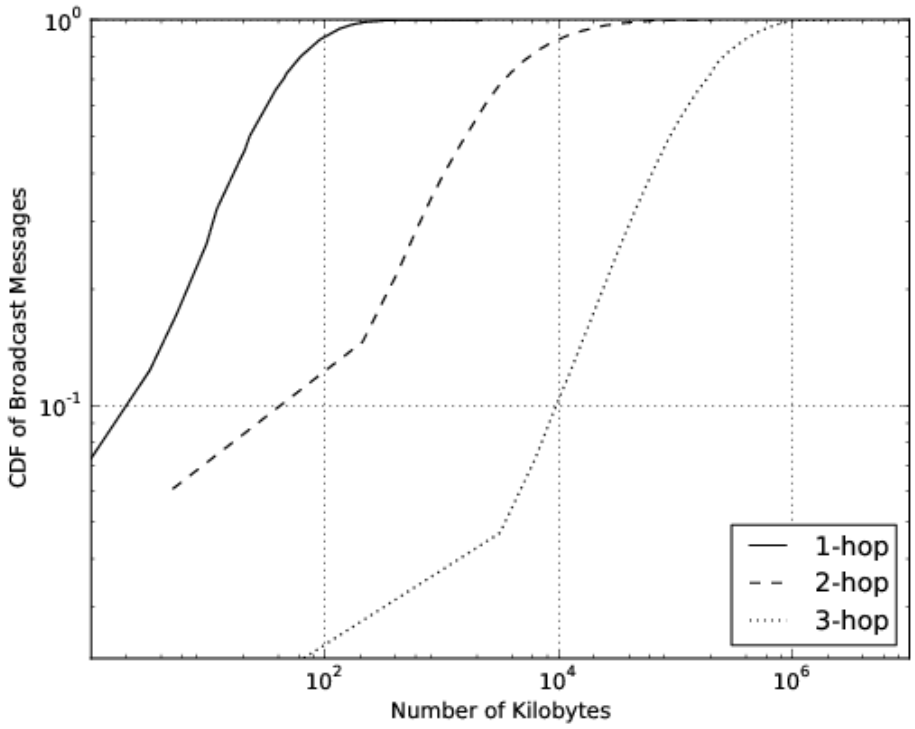



Figure 4-4. CDF of One, Two, and Three hop Queries. As the graph shows, increasing the hop count for SocialDNS Queries increases the bandwidth consumption by two orders of magnitude.

number of peers makes finding a short SocialDNS domain name more probable. Figure 4-3 shows the actual distribution of the number of host within a two-hop social circle.

### 4.3.2 Expected Bandwidth Cost

SocialDNS uses broadcasting as the primary method of communication; therefore, understanding the expected bandwidth costs helps the prediction of the traffic generated by SocialDNS queries. Figure 4-4 shows the cumulative distribution function (CDF) of the expected number packets generated when a user does a one, two, or three hop search for a SocialDNS mapping. The maximum packet size allowed in the DNS RFCs [83] is 512 bytes. Hence, a one, two, and three hop SocialDNS search generates about 42 Kbytes, 4.5 Mbytes, and 161 Mbytes of traffic on average, with a medians of

25 Kbytes, 1.7 Mbytes, and 90 Mbytes respectively. Therefore, a user is only allowed to perform a one or two hop broadcast in SocialDNS to avoid consuming to much bandwidth per DNS queries. Future work will look at efficient ways of caching the DNS mappings to minimize bandwidth consumption.

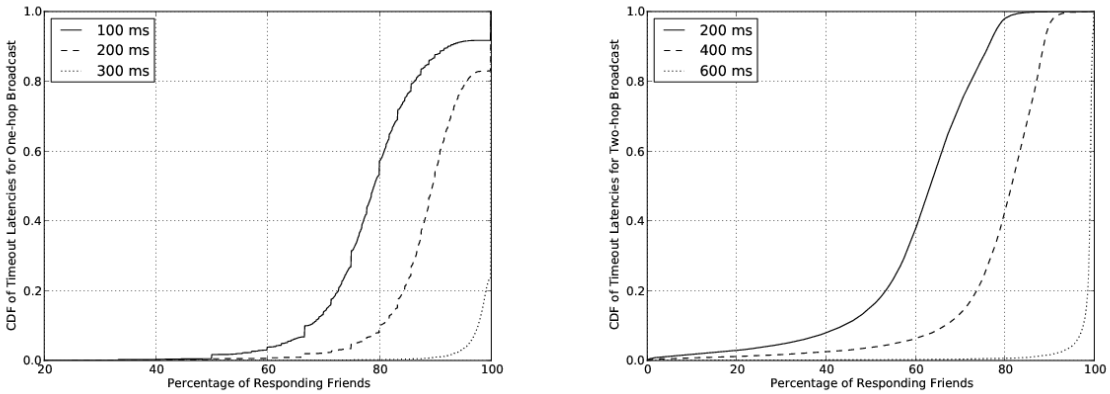
### 4.3.3 Anticipated Latency

Analyzing the latency helps the determination of the proper timeout to set per SocialDNS query to gather adequate responses from social peers. The first step in the latency analysis is to examine the relationship between friendships and geography. According to Liben-Nowell et al. [84], friendships in a social network are based on a geographic preference given by the formula $\mathrm{P(d)} = \frac{1}{\mathrm{d}^{1.2}} + 5.0 \times 10^{-6}$, where $\mathrm{P(d)}$ is the probability of friendship between two peers that are located at distance $\mathrm{d}$ kilometers away. Based on the works of Bassett et al. [85], one can derive latency from distance by approximating latency as $\frac{4}{9}$ the speed of light. Another work by Dischinger et al. [86] also has shown that in residential ISPs, a packet may take up to 2.5 milliseconds from the host machine to the ISP's router for a total round-trip time of 5 milliseconds. Using these previous works, the latency in milliseconds is approximated as:

$$\mathrm{latency} = \frac{\mathrm{d} \times 10^3}{\frac{4}{9}\mathrm{c}} + 5 \times 10^{-3} \tag{4–1}$$

where $\mathrm{c}$ is the speed of light at 299,792,458 m/s. Hence, SocialDNS uses the probability function $\mathrm{P(d)}$ to assign distances between friends and the equation (1) to approximate the latencies of friendship links in the social graph created by the P2PVPN.

Based on the aforementioned latency distribution, the measured percentage of received responses are based on timeouts of 100 ms, 200 ms, and 300 ms for one-hop queries, and 200 ms, 400 ms, and 600 ms for two-hop queries. For a timeout of 100 ms for a one-hop broadcast, the received responses are from 79% of friends on average, 89% for 200 ms, and 99% for 300 ms. In the two-hop broadcast scenario with timeout of 200 ms, 400 ms, and 600 ms and received responses from 61%, 78%, and 98% of




A Timeout Latencies for One-hop Broadcasts B Timeout Latencies for Two-hop Broadcasts

Figure 4-5. Impact of Query Timeouts. A) Timeout Latencies for One-hop Broadcasts. At 100 ms median is at about 75% and we hear back from all friends 10% of the time; the median at 200 ms is about 90% and we hit 100% results about 16% of the time; at 300 ms, our median is at 100%, and we get all results 75% of the time. B) Timeout Latencies for Two-hop Broadcast. At 200 ms, we obtain a median of 65%, but we obtain 100% less than 1% of the time; at we achieve a better median of 80%, with barely any improvement at the 100% mark; at 300 ms, we get a much better median of 99%, and we hit 100% about 3% of the time.

friends on average, respectively. Figure 4-5 plots CDF showing the distribution of the percentage of responding friends with the various timeouts for one and two hop queries.

### 4.3.4 Experimental Latency

There is currently a prototype of the SocialDNS system implemented as part of the proposed P2P architecture. SocialVPN is a P2PVPN which uses social networking backends such as Facebook, or Google Chat to automatically create P2PVPN links with friends. The whole software suite is written in C# and is available on `http://socialvpn.wordpress.com`. The Web-based interface is implemented as a search engine-like interface written in AJAX (see Figure 5-3).

The following experiments assess the functionality and performance of the prototype. The experiments involved both PlanetLab [47] and Amazon Elastic Cloud [87] (EC2) infrastructures. PlanetLab, a global research testbed with nodes located around the world, was used to deploy 600 P2P nodes which formed the bootstrap P2P overlay.

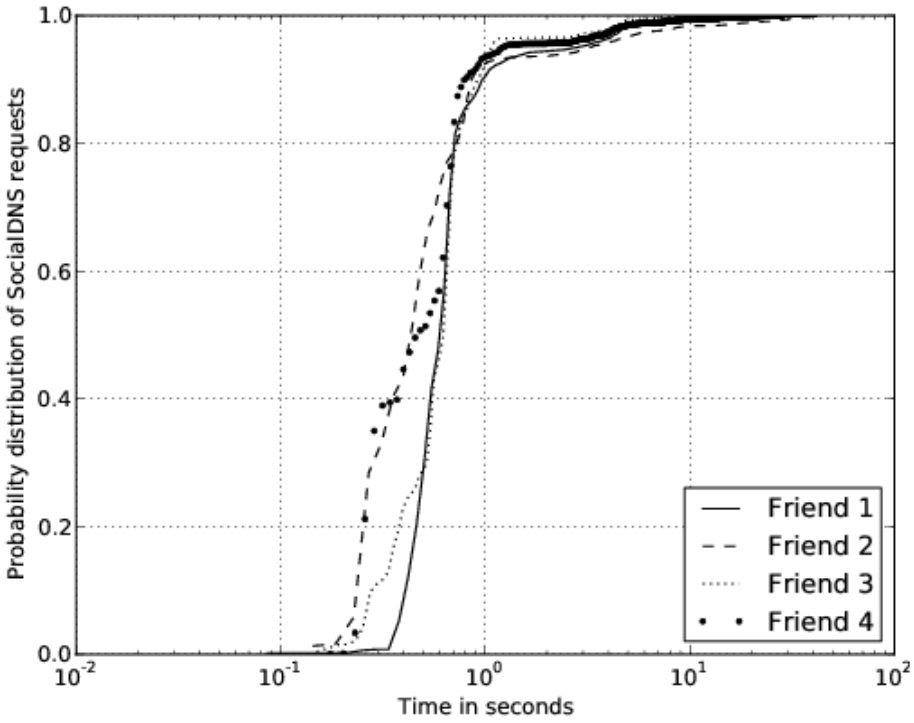



Figure 4-6. CDF of User Request Times. Each of the four user sent 3000 DNS requests (1000 to each friend). The four nodes were located in Virginia, California, Florida, and Ireland. We measured the round-trip latency of each DNS query. In each case, over 90% of requests takes less than one second.

The P2P nodes on PlanetLab did not include the SocialDNS software stack. SocialDNS nodes were deployed on Amazon EC2 because the system requires adding the local SocialDNS server to the operating system's DNS configurations. Modifying the DNS settings is not possible on PlanetLab nodes, but, this is possible on Amazon EC2 because of the necessary root access needed on the virtual machine. On the Amazon EC2 nodes, the system added DNS mappings to the local caches, searched and imported DNS mappings from social peers, and resolved DNS mappings created by the local user and friends. Hence, the design proved feasible and all components integrated successfully.

It is also important to explore the latencies of the DNS queries through the peer-to-peer overlay running on PlanetLab with nodes located around the globe.

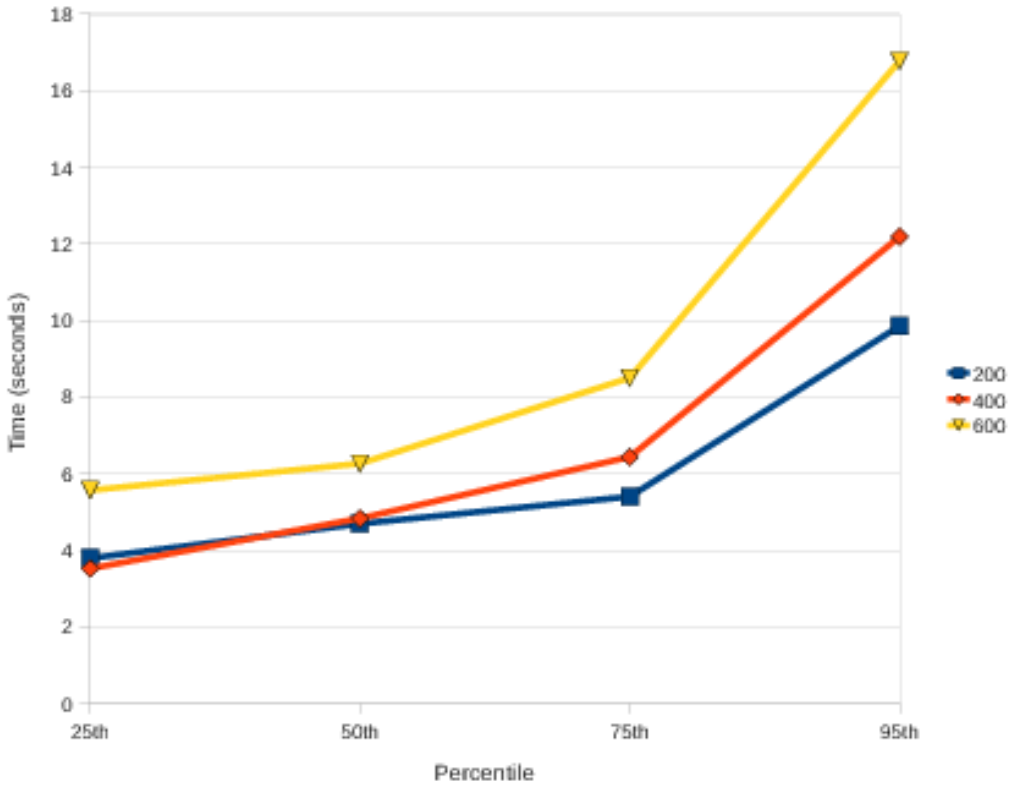



Figure 4-7. Time Taken to Broadcast to Various Size Networks. 100 broadcast queries were sent to each network size. 95% of the queries took less than 16 seconds for a network size of 600 nodes.

Three SocialDNS nodes were deployed on the various regions of the Amazon EC2 infrastructure located in Virgina, California, and Ireland; a fourth node was located in Florida from a residential ISP. Figure 4-6 shows the cumulative distribution function (CDF) of 3000 DNS queries conducted at each of the four nodes. The results show that more than 90% of the DNS requests take less than one second, irrespective of the geographic locations.

It is also desirable to measure the time taken to broadcast DNS queries simultaneously to all friends through the overlay. Assuming a private peer-to-peer overlay [88] consisting only of social peers, the broadcasting time was measured for queries to all peers with network sizes of 200, 400, and 600 nodes. Using only the PlanetLab nodes,

100 broadcast queries were sent to each of the different sized networks. Figure 4-7 shows the 25th, 50th, 75th, and 95th percentile of time taken to broadcast to the whole network. The 95th percentile are 8, 12, and 16 seconds for network sizes of 200, 400, and 600 nodes, respectively.

## 4.4 Summary

This chapter presents a decentralized domain naming solution suited for P2PVPNs. This work makes the case that providing IP access is not enough to enable simple implementation and deployment of social networking applications; the ability to assign create DNS mapping to their machines and friends is also useful. SocialDNS provides that freedom and reuses existing concepts from decentralized naming solutions from local area networks. It also improves upon the current limitations of LAN-based decentralized naming systems that makes them unsuited for a P2PVPN environment.

The SocialDNS design makes it trivial to assign short domain names to resources in a P2PVPN through social scoping; meaning, a user only has to compete with less than 2,000 other users on average for a domain name. For simple management, this design provides a minimal configuration, easy-to-use AJAX Web interface where a user can search his/her friends' SocialDNS cache and import mappings of interest. For design simplicity, broadcasting is used as the main method of communication among SocialDNS nodes in the P2PVPN. In case of name collisions due to the lack of a central authority, a popularity-based ranking mechanism is employed to pick the mapping with the highest presence in the SocialDNS caches of the social circle. By using the social graph properties, the analysis predicted the bandwidth cost and responsiveness by assuming a P2PVPN will form a social graph. The results validated the SocialDNS design choices because the expected performance is practical.

# CHAPTER 5
# CASE STUDY: DESIGNING A DISTRIBUTED MICROBLOGGING SERVICE

A distributed microblogging service was designed to more concretely examine the implications and limitations of the proposed architecture. Since microblogging services are a popular type of social networking services, it seemed a good candidate application to prototype in the SocialVPN environment. The proposed architecture is essentially a peer-to-peer virtual private network; the remainder of this chapter uses the term P2PVPN to refer to systems similar to the architecture. This chapter details the design of this distributed microblogging service along with analysis of various aspects of its performance.

This dissertation presents Litter, a decentralized microblogging service that leverages currently available peer-to-peer virtual private networking (P2PVPN) technologies. P2PVPNs provide privacy and low-latency communication in the common case of P2P messaging among social peers. With private connections and multicast message delivery already enabled by P2PVPNs, this work focuses on some of the other key hurdles of building a private and decentralized microblogging service, such as controlling the scope of a message and ensuring data availability. The Litter design utilizes both IP multicasting and data replication through high degree nodes to ensure that peers are able to publish messages with varying scopes (i.e. friends, friends of friends, and/or the public). The analysis studies the implications of the Litter data dissemination mechanism for a decentralized microblogging service through a simulation using a 3-million user social graph from Orkut. The prototype implementation demonstrates the feasibility of the design choices along with an experimental deployment on FutureGrid to test its performance on the wide area. Overall, the experimental results show that peers can effectively follow each other's updates with acceptable overhead.

The primary goal of this design is to enable fast, private updates among friends, rather than the Twitter model which focuses more on posting messages publicly on the Web. By utilizing SocialVPN [67], Litter builds upon a layer where users have private IP connectivity to each other along with IP multicast support thus enabling trusted social communication on the Internet (even through NATs and firewalls). P2PVPNs naturally map to social graphs and therefore serve as a strong foundation for building decentralized social services because they specialize in low latency, private communication by creating direct, encrypted paths among friends.

This chapter focuses on understanding the characteristics of the social overlay formed by P2PVPNs – for example, the ability to reach friends with a one-hop multicast IP packet, and the impact of the high clustering coefficient on replication and data availability. With this underlying social overlay, Litter lets the user control the propagation of a post in the social overlay through the use of IP multicast packets. Litter also leverages the properties of the social graph in its data replication heuristics in order to efficiently disseminate public posts throughout the social overlay. By designing with privacy as a starting point, the proposed microblogging service is a system that makes it more difficult to censor, disrupt, and infiltrate which ultimately creates a more robust social networking application.

The main contributions of this chapter are:

- A novel decentralized microblogging service with privacy as the primary focus and which also uses IP multicasting, and random walks to propagate updates through a social overlay constructed by a P2PVPN.
- A simulation-based analysis which predicts bandwidth costs and compares data dissemination strategies using a 3-million node social graph from a real social networking website.
- A prototype implementation of the Litter microblogging service which validates its functionality and design choices.

## 5.1 Motivation

In recent years, social networking services such as Twitter and Facebook have played a major role as a vital communication tool around the world. For example, the Arab Spring is at times dubbed the “Twitter Revolution” because of the microblogging service’s key role in connecting protesters. Another instance is during the catastrophic earthquake in Haiti, where Twitter updates posted through mobile devices became a primary source of real-time information. Microblogging services provide users with an intuitive method for communicating ideas, discontent, and are now an invaluable means of organizing revolts. As a result, a common first step of governments seeking to restore order during a revolution is to block access to these Internet-based social services. In some extreme cases, whole nations have been disconnected from the global Internet in order to deny the dissidents access to any service that may help amplify the awareness of their grassroot revolution [5]. One lesson that became evident to the research community is the ease with which authoritarian governments can instantly block all IP traffic flowing in and out of a country. For example, in the case of Egypt, all of the country’s BGP prefixes disappeared from global routers making it impossible for the outside world to route any IP packets to Egypt [5]. This was easily accomplished once the government took control of the BGP routers in Egypt and stopped sending routing updates to the rest of BGP routers in the world.

Although it is fairly easy to disconnect a nation from the Internet by disabling a few key connections to the outside world, in most cases, it requires more effort to shutdown a country’s entire internal Internet apparatus. This was the case in Egypt, after disconnection from the Internet; services running internally where still accessible by the people. Hence, this occurrence motivated the need for a microblogging service, that would continue to function despite a disconnection from the global Internet. A decentralized approach also provides an opportunity to address the issue of privacy in

microblogging especially after the many public instances of Twitter handing over private user data to authorities.

The goal is to create a service which makes it possible for end-users to send messages to their followers directly in a peer-to-peer manner without having to depend on a centralized service. Hence, disabling such a service would require governments to shut down their internal Internet infrastructure, which may be a more costly and complex endeavor. By leveraging existing P2PVPN technologies, Litter is based on a service that allows communication among peers through direct, encrypted IP tunnels without the need of a middleman. With private messaging to trusted peers as the primary feature of a P2PVPN, this eliminates many of the shortcomings of a centralized approach by making it harder to aggregate all messages in a centralized database. Moreover, by only disseminating messages through trusted friends, it makes it harder for governments to monitor user activity and limit their ability to freely communicate.

## 5.2 Related Works

In this section, the focus is on recent approaches for decentralized microblogging services along with some recent alternatives for building decentralized social networks and publish-subscribe systems in unstructured peer-to-peer networks.

### 5.2.1 Decentralized Microblogging

FeedTree [89], and Megaphone [90] are peer-to-peer micronews/RSS services which are built on top of Scribe [91], where followers join a multicast tree through the use of the Pastry [18] structured overlay. By sending updates to the root node of the tree, the messages are propagated through all members of the tree. These are elegant approaches because they have very little overhead besides overlay maintenance, and guarantee that all followers receive updates within a logarithmic number of hops with respect to the number of peers in the system. The major difference between these approaches is that FeedTree uses the RSS feed model to disseminate messages while Megaphone uses the microblogging model of delivering short 160-character messages.

Also Megaphone uses encryption to guarantee the delivery of trusted messages but has to deal with handling key distribution and session keys. For the Litter design, this work explores the feasibility of designing a microblogging service without depending on a structured DHT.

FETHR [92] is a more recent approach aimed at delivering a fully decentralized HTTP-based microblogging service. Peers subscribe to each other by exchanging canonical URLs and use the HTTP GET and POST methods to pull or push updates to each other. This approach also recommends using gossip-based dissemination for users with a high number of followers. The use of the ubiquitous HTTP protocol and the simplicity of the gossip protocol make this a very practical solution; however there is one key weakness: the authors do not mention if users will have to run the HTTP service locally or if they will depend on a service provider. If users have to depend on a service provider, then this approach becomes much more similar to a federated system such as email or XMPP. If users run this service locally, then issues such as connectivity through NATs and firewalls become major impediments which are not addressed in the design. The use of a P2PVPN with NAT traversal allows Litter to bypass such hindrances.

In Cuckoo [26], peers use a DHT to discover each other's endpoints and send follow requests directly to each other. Publishers are then able to send updates directly to a subset of followers, depending on bandwidth availability, while the remaining followers use a gossip protocol to propagate updates among themselves. To handle churn, the Cuckoo approach relies on a centralized backend which stores all updates and follow requests from all users in case a publisher is not online. The use of a DHT and a centralized backend separates the Litter design from the Cuckoo approach.

### 5.2.2 Decentralized OSNs

A decentralized microblogging service can also be considered to be a decentralized online social network (OSN). The main difference is that rather than publishing short 140-character messages, a user can also publish pictures, videos, and other objects

of interest. PeerSoN [11] is a fully decentralized OSN. Similar to Cuckoo, peers use a DHT as a lookup service to locate each other's endpoints, then exchange encrypted information directly with each other. Safebook [10] is another DHT-dependent OSN which focuses on anonymity by adding an additional layer of indirection to message requests by routing them through multiple layers of friends and friends of friends. Neither of these approaches mentions a replication mechanism to ensure content availability when users are offline. Vis-a-vis [93] is yet another decentralized OSN proposal which relies on DHT-based multicast for message propagation. In this system, a user runs a virtual individual server (Vis) which stores and serves all of the content on behalf of the user. To ensure content availability, the user can run a Vis on a cloud provider such as Amazon EC2. LifeSocial [94] is a totally DHT-dependent OSN; everything a user publishes is stored in the DHT which ensures data availability even in the case of churn.

### 5.2.3 Publish-subscribe Systems

Structured DHTs are well-known for supporting publish-subscribe systems (e.g. Scribe [91]), but there are also unstructured publish-subscribe systems which can be applied on a social overlay. In Quasar [95], group members advertise their memberships to all nodes within a K-radius which are aggregated through attenuated Bloom filters. Publishing messages to group members involves doing a k-parallel random-walk with the group id in the message. As the message travels through the network, each node checks its Bloom-filter for group members in its vicinity; if a member is found, the message is routed towards that node, if not, the message continues to the next random node. Vitis [12] is a hybrid system which uses both a DHT and clusters for propagating updates by broadcasting each other's messages. Litter differs by focusing on privacy and performance and by pushing updates directly to friends over encrypted links since that is believed to be the common case.

## 5.3 Background

The proposed microblogging service depends on two basic networking services: IP multicasting for pushing posts to friends (and friends of friends), and UDP unicasting for data replication throughout the social overlay. Therefore it is important to provide some background on the IP networking capabilities of P2PVPNs and explain the assumption that P2PVPNs form social overlays with characteristics similar to social graphs. Finally, this chapter argues that a microblogging service should be built around the concept of private messaging, which is the primary motivation for starting the Litter design with a P2PVPN.

### 5.3.1 P2PVPNs and IP Multicasting

Although it is common knowledge that the Internet does not support IP multicasting and UDP traffic is firewalled in many cases, there still exist methods that currently allow peers to have direct, unrestricted IP connectivity to one another, including IP multicasting support. Virtual private networking is the typical solution to providing unblocked IP connections to nodes that are geographically dispersed. In this scenario, virtual IP traffic is tunneled over a TCP or UDP connection to a VPN gateway which routes the traffic accordingly. Hence, peers can get unrestricted IP connectivity to each other by connecting to a common VPN gateway (e.g. OpenVPN). The gateway can also provide IP multicasting support by tunneling multicast IP packets to all peers connected to the same gateway. However, this is a highly centralized approach which is not scalable and suffers from a single point of failure.

A peer-to-peer virtual private network (P2PVPN), such as [20, 67], is a practical decentralized alternative to centralized VPNs. In a P2PVPN, peers leverage P2P technologies to tunnel IP traffic directly to each other instead of relying on a centralized VPN gateway. Virtualizing a P2P connection into a virtual IP link is a powerful concept because applications can be developed independently, without any knowledge of the P2P virtual networking infrastructure. This is a major departure from common P2P

application development where the application is intimately coupled with an application layer P2P library. Such a strong coupling limits the portability and modularity of the system.

P2PVPNs also provide IP multicast support by tunneling the multicast packets to each friend over an encrypted P2P connection. There has been much research in application-level, P2P multicast [96, 97], each with their own set of assumptions and requirements, thus necessitating developers to design their software around their technical specifications. In a P2PVPN case, applications join a multicast group through the well-adopted Berkeley sockets API, and they are able to send data to group members by simply sending a UDP datagram to a multicast IP address. IP multicast support through P2PVPNs over the Internet plays a key role in the Litter design choices because it does not require reinventing the wheel by having to design yet another multicasting strategy for a P2P application.

### 5.3.2 The P2PVPN Social Overlay

A recurring theme of this chapter is the assumption that P2PVPNs form social overlays with small-world characteristics. This is primarily the case because P2PVPNs such as Hamachi [20] or SocialVPN [67] provide a means for trusted and social peers to communicate with each other at the IP layer in order to collaborate (e.g. multiplayer gaming, screensharing, media sharing, among others). Also, in order to establish these encrypted peer-to-peer connections in a P2PVPN, users are required to exchange security credentials either through a shared password, or the exchange of cryptographic public keys. Since connections are created with diligence and mainly with trusted, social peers, the resulting peer-to-peer network is a social overlay. This assumption has very important implications because it is the basis for later analysis and design decisions. For example, in the simulations below, a well-established social graph generation models is used to represent this P2PVPN social overlay. Moreover, the data dissemination model is based on the assumption that peers primarily want to send updates to friends

and friends of friends, and followers tend to be socially close of their publishers. Thus, Litter optimizes for the common case of disseminating messages to social peers. Its replication strategy leverages the clustering of social graphs to increase the probability that social distant peers are still able to obtain posts from a publisher. Finally, this social overlay is independent from the directed social graph created by subscription-based social networks. The proposed design ensures that a method of message dissemination exists among all peers, but it gives priority to friends (and friends of friends).

### 5.3.3 Microblogging Privacy by Default in P2PVPNs

The motivation for designing a private, decentralized microblogging service is not to provide a peer-to-peer alternative to Twitter. The primary focus is to build a framework where peers can privately send status updates to their friends (and friends of friends) without fear that their activities are being indefinitely recorded and monitored by a third party. A P2PVPN allows developers to leverage existing private IP tunnels that can be used by a variety of different applications, not just a microblogging service. With a private microblogging service, peers can confidently send updates to friends knowing that these updates will only be saved by trusted peers instead of a centralized database. This assumption implies that different types of usage is expected between Litter and popular microblogging sites such as Twitter where the focus is much more on publishing information publicly. The Litter approach starts with the messaging capabilities of the P2PVPN, but it also provides mechanisms to make it possible for peers to push updates to everyone publicly as well.

## 5.4 Design

The Litter microblogging service is built on two basic IP layer mechanisms: 1) IP multicasting to propagate messages to two-hops neighbors in the social graph, 2) and UDP datagrams for traversing the social graph to disseminate updates to social distant peers. In order to enable this service, the assumption is that users run a P2PVPN service that provides peer-to-peer, encrypted IP tunnels to friends, even those behind

Table 5-1. The four types of messages in the system

| Type | UID | Dest | TTL | ID | Timestamp | Perm | Payload | Signature |
|---|---|---|---|---|---|---|---|---|
| Multicast Push | UID of post creator | All | 1 or 2 | Uid + post | Yes | F or P | Message content (140-char max) | Hash (uid, id, timestamp, permissions, payload) then signed with private key |
| Multicast Pull | UID of post requester | All | 1 or 2 | Random number | Yes | None | List of (followers, highest post ID) | Hash (uid, id, timestamp, permissions, payload) then signed with private key |
| Random Push | UID of post creator | Any | 100, 200, 400 | Uid + post | Yes | F or P | Message content (140-char max) | Hash (uid, id, timestamp, permissions, payload) then signed with private key |
| Random Pull | UID of post requester | Any | 100, 200, 400 | Random number | Yes | None | List of (followers, highest post ID) | Hash (uid, id, timestamp, permissions, payload) then signed with private key |

NATs and firewalls; and the P2PVPN service supports IP multicast. The previous chapters describe the SocialVPN system which meets these criteria.

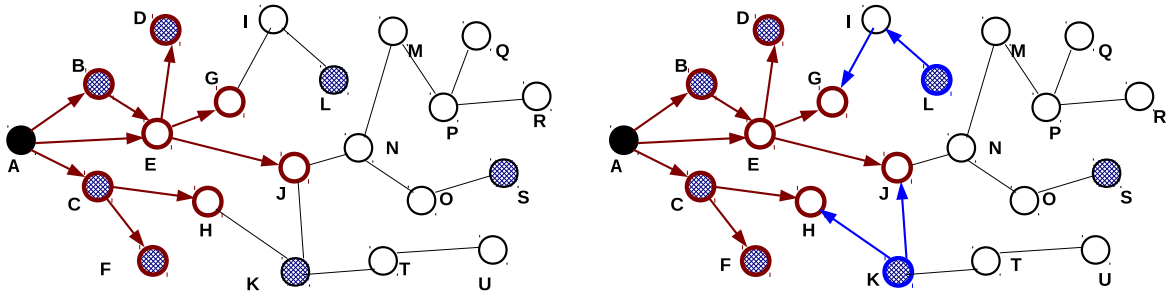
### 5.4.1 Multicast Push to Followers

**Message Format.** Every post generated in the system contains the following information: a creator UID, a destination, a TTL, a timestamp, a post ID, a permission flag, a message, and a signature (see Table 5.3.3). The creator UID helps the system keep track of the source of the posts. The destination field is set to *all* meaning that the message is broadcasted to all friends through IP multicast. The TTL field lets the system control the scope of the message, if a user would like to post messages to only friends, then the TTL is set to 1. If a user would like to reach both friends and friends

of friends then the TTL is set to 2. The timestamp helps keep track of the creation time of the message. The post id is a number which increments by one for each post a user generates. The permission flag help control the privacy of the message; peers are only allowed to share updates from pull requests if the message is set to P (or public). The message is the actual content of the post; currently this is limited to a maximum of 140 characters. The signature field is created by hashing the creator UID, the timestamp, the post ID, and the message and signed with the creator's private key. Therefore the signature ensures the integrity of the message and verifies its creator, assuming recipients have access to the creator's public key.

A multicast push is the most basic message type in the system. Since a P2PVPN enables IP tunnels to friends, and multicast support which makes it very easy to contact all friends privately, implementing this feature simply involved sending a UDP packet to an IP multicast address. However, it is important to note that despite its simplicity, it is a vital first step in allowing peers to send updates privately. This multicast push, which is viewed as the common case, is efficient because messages are sent directly to friends, and they are private by default because they go through encrypted IP tunnels. For example, Alice wants to go bowling with her friends, so she uses the Litter service to quickly broadcast that message to her one-hop friends who are currently online. In this example, Alice limits the scope of her message to only friends, which means that the TTL will be set to 1.

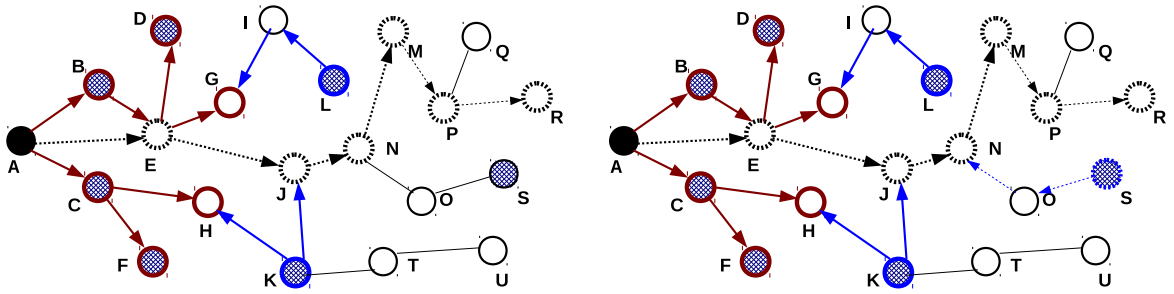
Suppose now that Alice is visiting Portland and would like to know if any of her friends or friends of friends know of any good restaurants in the area. By setting the TTL to 2, she is able to propagate that message using the same multicast push mechanism (see figure 5-1A). By starting with the social overlay formed by P2PVPNs, sending messages to friends and friends of friends is translated to a UDP datagram sent to an IP multicast address with a TTL value controlling the scope. Unlike many previous works, pushing messages to close friends is a better alternative to polling for new messages




A Multicast Push B Multicast Pull

Figure 5-1. Multicast push and pull mechanism. A) Multicast Push. Publisher (filled in black): A, Followers (partial filled in blue): B, C, D, F, K, L, S. Publisher node A pushes posts to nodes B, C, D, E, F, G, H, J (since they are two-hops away). Some nodes see duplicate posts (e.g. node E from both nodes A and B). B) Multicast Pull. Nodes K and L are three and four hops away and they are able to receive updates from publisher node A through nodes G, H, and J. When node L does a two-hop multicast request, it reaches node G which sends the posts back to node L.

because it improves the interactivity of the system by delivering messages with low latency. Moreover, these messages are encrypted end-to-end by the P2PVPN which ensures a third-party is not able to intercept or spoof the data. Thus, it is used as the primary mechanism to publish posts privately to connected friends in a timely fashion.




A Random Push B Random Pull

Figure 5-2. Random push and pull mechanism for distant followers. A) Random Push to Distant Followers. Publisher (filled in black): A, Followers (partial filled in blue): B, C, D, F, K, L, S. Node A pushes posts through a random path to extend the reach of the posts beyond four-hops. As a result, node A's post is stored at nodes E, J, N, M, P, and R. B) Random Pull by Distant Followers. Node S does a random-walk and gets node A's posts through node N which has cached it due to node A's random walk push.

### 5.4.2 Multicast Pull by Followers

**Message Format.** Users do not only rely on the publishers to push messages to them, they can also proactively pull for new updates as well. A pull request contains the following information: a requester UID, a destination, a TTL, a request ID, a timestamp, the list of followers, and a signature. The requester UID helps the system identify who should receive responses for a particular request. The destination field is set to *all* to indicate the user of multicasting. The TTL can be set to 1 or 2 depending if the follower would like to pull updates from friends and/or friends of friends as well. Each request has a unique ID which allows the system to keep track of where a request originated, so that replies can be sent along the right path. The timestamp helps determine the freshness of a request; requests beyond a certain time window can be ignored. The list of followers includes the publishers that the requester is following along with the last post ID for each follower. That information is used by each node that receives the request to determine if a requester is missing the latest posts from a particular publisher. The signature in this case is the signed hash of the requester UID, request ID, the timestamp, and the list of followers. The signature makes it possible to verify the legitimacy of the request and helps guard against spoofed requests.

Multicast pulls are used by followers because sometimes they do not receive pushed posts due to packet drops in the P2PVPN, or due to the follower being offline. Currently in the Litter prototype, the pull requests are generated every five minutes, but this period is configurable by the user. The multicast pulls also serve another important task: it makes it possible for followers who are four hops away to receive public updates from a publisher. As shown in Figure 5-1B, if node A pushes a public post to friends and friends of friends, node L is able to retrieve those updates through node G, even though nodes A and L are separated by four social hops. Therefore, when node A pushes a public post to all friends and friends of friends, the two-hop requests make it possible for

followers who are four hops away to receive updates through common subset of friends of friends.

Returning to the privacy argument, when a user creates a post, it can be set for friends only, friends of friends, and everyone. Friends and friends of friends will only share public updates with social distant peers (users that are farther than two hops away in social overlay) if they are public. Also, even if the two-hop push and two-hop pull mechanisms provide four-hop coverage, there are still cases where followers are more than four hops away. This limitation is addressed with the following message propagation mechanism.

### 5.4.3 Random-walk Push to Distant Followers

**Message Format.** The message format for the random-walk push is very similar to the multicast push with two main differences. First the destination field is set to *any* instead of *all*. When set to any, the system selects a random peer to forward the message. This is the most basic form of a blind random-walk. Second, the TTL value is set to a large value (e.g. 100, 200, or 400) depending on the user's preference. The TTL serves as the replication factor because the message is stored at each node it reaches. Selecting a higher TTL increases the chances that distant followers will be able to receive the updates of the publisher.

The need for a random-walk push is necessary to ensure that any peer in the network could follow each other. Although this is not the common case, it is important to provide a controllable mechanism in which a publisher could expand the reach of their posts. By replicating posts throughout the social overlay, a publisher can control the availability of his/her updates. However, randomly replicating the messages does not ensure that it is stored at follower nodes; followers also need to request updates from peers in the network to increase their chances of obtaining messages. For example in Figure 5-2A, the random push method replicates posts at nodes E, J, N, M, P, and

R. The nodes are not follower nodes, but the hope is that they will make it easier for followers to retrieve updates from node A.

### 5.4.4 Random-walk Pull by Followers

**Message Format.** The message format for the random-pull is quite similar to the multicast pull requests. However, the destination field is changed to *any* in order to cause a random-walk in the social overlay. Also, the TTL is set to a large TTL (e.g. 100, 200, or 400) to increase the search path for updates.

The random pull requests are handled in a similar manner as the multicast pull request; if a peer notices that the requester does not contain the latest updates, it replies with these updates. As shown in Figure 5-2B, node S is able to pull node A's updates through node N that holds a replica of node A's updates from the random walk. Node N replies to node S and forwards the random pull request to the next node in the system. The replies take the reverse path of the random-walk. The reverse path is known because each node builds a routing table that maps request IDs to IP addresses. That information makes it possible to send a reply along its reverse path in the system as long as the reply has the same ID as the request. Also, a random pull request continues through the network until the TTL reaches zero. This ensures that the requester can control the number of nodes to sample for new messages from his/her followers.

Overall, the combination of the random push and pull mechanism makes it possible for any peer in the system to follow another peer as long as both the publisher and the follower use the appropriate TTL values in their request. By providing a configurable TTL in the system, publishers are able to control how much publicity they want for their posts; therefore, if users feel they have something important to say to the whole network, they can set a high TTL such as 400, if they only care about reaching close friends, they can set a lower TTL of 1 or 2. On the pulling side, a follower is also able to use the TTL to increase his/her ability to retrieve updates from social distant peers in the network.

### 5.4.5 Message Privacy and Verification

Message privacy is accomplished through a permission flag. If a message is set to public (or P), then friends are to allowed to share the posts when they receive a pull request. With the permission flag and the TTL, the user is able to control the scope of their updates. For followers who are multiple hops away from the publisher, a message has to go through multiple intermediate nodes in the social graph. To ensure message verification and integrity, each message contains a signature formed by hashing the contents of the post and signing it with a private key. A follower is responsible for acquiring a publisher's public key through a trusted out-of-band medium. Dealing with cryptographic key distribution in peer-to-peer system is beyond the scope of this research.

## 5.5 Implementation

Another key contribution of this work is a prototype implementation that users can download and provide feedback. The code is open source (http://github.com/ptony82/litter). Coding simplicity is one of the strengths of the proposed approach: by leveraging P2PVPNs and reusing Berkeley sockets API and Web/database modules for user interface and data storage, the code is only 1500 lines. Litter also differentiates itself from other previous attempts at a microblogging service by ensuring ease of deployment and usability. In this section, the technical details of the Litter approach are shown to be much more practical than previous solutions.

The first requirement for decentralized microblogging service is a P2PVPN of which there are various commercial and open-source solutions that a user can install. The implementation is written in Python and uses its corresponding standard modules to enable the following capabilities: network communication, object serialization, data encryption, data storage, and HTTP interface. Network communication is the first crucial component and it is based on the Berkeley sockets API available in Python. To communicate, the service first joins an IP multicast group and then listens on the

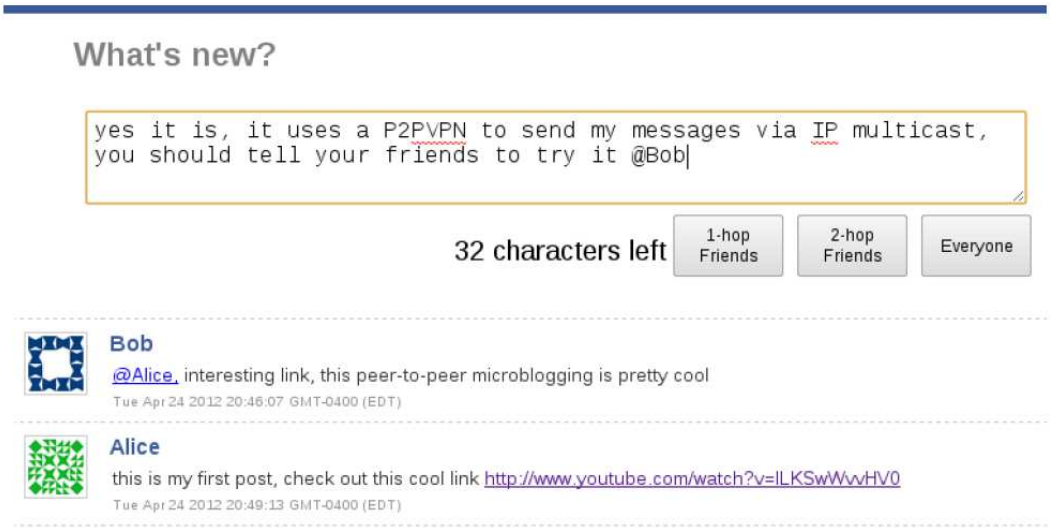



Figure 5-3. Screenshot of the microblogging service

appropriate UDP port for incoming posts and pull requests. To send a multicast post or a pull request, the payload is encapsulated in a UDP datagram that is sent to the multicast IP address through the virtual networking interface (NIC) of the P2PVPN. The P2PVPN then sends the multicast IP packet to all connected friends, so that the listening nodes can receive the multicast UDP message.

The fields of each message type are serialized through the use of the Javascript Object Notation (JSON) protocol, which converts the contents of these message types into a normalized string which becomes the payload of the UDP datagram. An important field of all the different message types is the signature. For generating the hash,the SHA1 hash functions are used which are part of the standard Python library, and for signing the hash, the RSA private key is used. For data storage, Litter utilizes the SQLite database module in the Python standard library.

The final component is the HTTP interface which plays an important role in the design in terms of interfacing with the local microblogging service. Litter provides an HTTP interface primarily because it makes it possible to interact with the microblogging service through a Web browser instead of a custom user interface. Therefore, Litter has an AJAX Web interface to interact with the local microblogging service (see figure 5-3).

The first experiment tests a 100 node FutureGrid deployment to study the IP multicast performance on an open-source P2PVPN on the Internet. FutureGrid is an academic testbed for cloud research which lets users instantiate virtual machines at geographically dispersed locations. As shown in figure 5-1, all of the latencies are below 100 ms which demonstrates that a user can push updates to social peers that are hundreds of miles away fairly quickly.

## 5.6 Analysis

This section discusses the bandwidth costs of the multicast messages, and also studies the impact of various connection limits and TTL values on the propagation of updates through the social overlay. This analysis is based on the assumption that P2PVPNs form social overlays; therefore, by analyzing the Litter design a representative 3-million user social graph from Orkut, it is possible to gain some insight on the performance of the proposed implementation.

### 5.6.1 Graph Characteristics

The social networking graph utilized in this work was obtained from the authors of [98] and this is a summary of its characteristics according to the authors. It was collected during October 3rd to November 11th 2006 using a partial breadth-first search. It contains 3 million nodes and 223 million edges which represents about 11% of the social network at the time. The node degree distribution is best estimated as a power-law distribution with a coefficient of 1.50 with an average degree of 106. However, since partial BFS crawls tend to oversample high-degree nodes, this power-law coefficient is probably higher than the true coefficient of the whole graph. Finally, the social graph has an average path length of 4.25, with a radius of 6, and a diameter of 9. The clustering coefficiency is 0.171 which is in the range of typical social networking graphs.

One of the novelties of this approach is the use of such a massive social graph for conducting the analysis. Most peer-to-peer simulation packages do not easily scale to 3

million nodes due to both time and resource constraints. For analysis, the Lemon Graph Library is used which is a lightweight and efficient software package for modeling and optimizing networks. It is written in C++ and has a small footprint for mapping social graphs in memory.

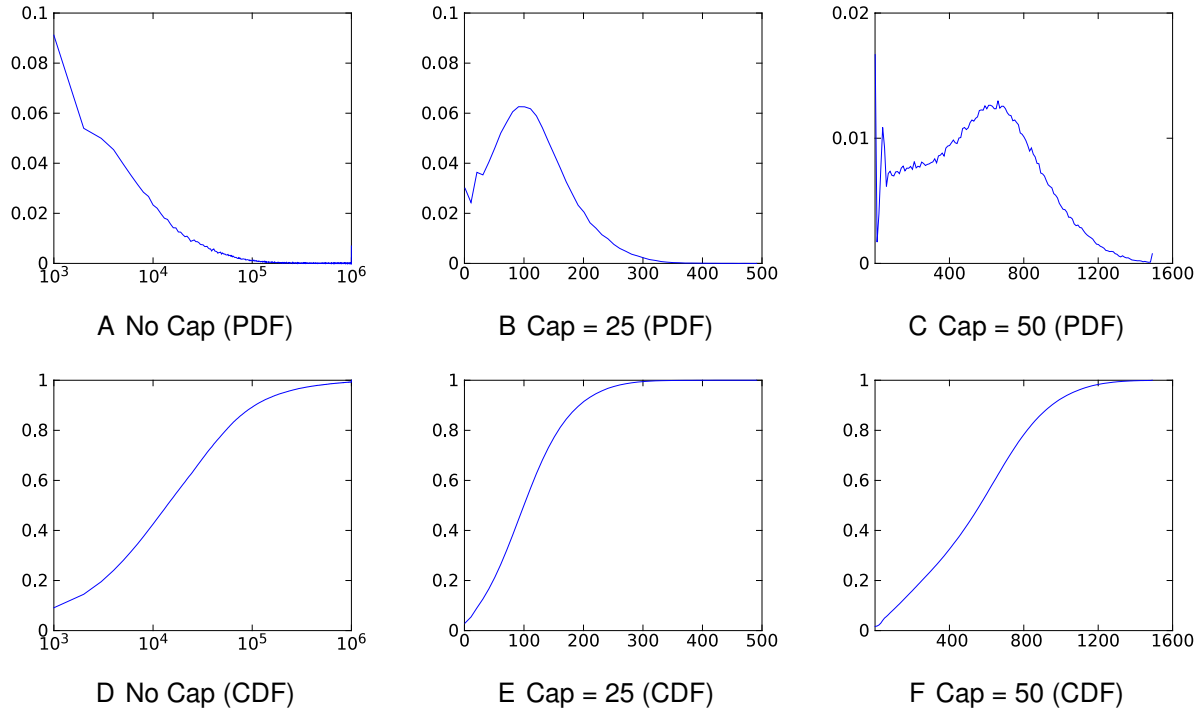



Figure 5-4. Bandwidth cost of two-hop broadcasts for 3M user Orkut social graph. The x-axis is the number of KB for push/pull messages at a two hop radius, the y-axis is its probability. A) No Cap (PDF). B) Cap = 25 (PDF). C) Cap = 50 (PDF). D) Cap = 50 (PDF). E) No Cap (CDF). F) Cap = 25 (CDF). G) Cap = 50 (CDF)

### 5.6.2 Bandwidth Costs of Two-hop Push/Pull Publishing

Litter relies heavily on the use of a two-hop multicast to push messages to friends and friends of friends. It also utilizes a two-hop multicast pull to guarantee that followers who are three and four hops away in the social graph can also retrieve messages upon request. Therefore it is imperative to understand the bandwidth usage of such a mechanism in the context of a social graph. In the analysis, it is assumed that each post is 1KB for simplicity. Since posts are only 140 characters with some additional fields

for routing, authentication, and integrity verification, it is expected that the majority of messages to fall well under this 1024-byte assumption.

Figures 5-4A and 5-4D demonstrate the probability distribution function (PDF) and cumulative distribution function (CDF) of using a two-hop multicast mechanism to publish messages to friends. According to these distributions, the first quartile is 4.61 MB, the median is 13.99 MB, and the third quartile is 41.53 MB. The average bandwidth cost is 54.91 MB. Such a high bandwidth cost per message is due to the presence of many high degree nodes in the social graph. It is important to note that this is not the bandwidth cost per node, it is the total bandwidth utilized by the publisher and his/her first hop friends. For example, assuming Alice has 2 friends, Bob and Carol, who each has 3 friends. The total cost of a two-hop multicast from Alice would be 8 KB (assuming 1 KB posts).

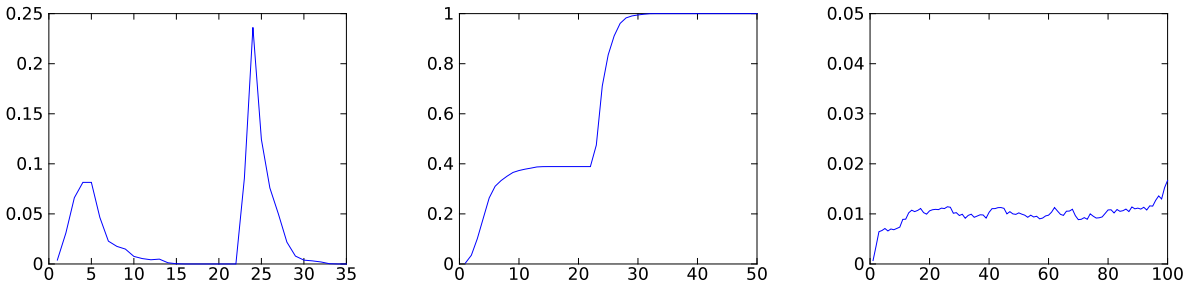
As mentioned in the design section, the system leverages existing P2PVPN technologies which allows for the development of this decentralized service using simple UDP datagrams and IP-multicasting for message dissemination. However, P2PVPNs creates an overlay network which requires constant monitoring and maintenance of each P2P connection. In a representative social graph, an average user will possess over 100 social connections. Maintaining such a large number of connections over time can be costly in terms of both bandwidth and power consumption. Therefore, this analysis explores the impact of imposing a limit on the number of social connections that a peer can maintain. These limits are implemented randomly where there is no preferential treatment given to any connection. This chapter also analyzes limits of 25 and 50 connections per user. As seen on figures 5-4B, 5-4C, 5-4E, and 5-4F, the implementation of the connection limits greatly reduces the bandwidth cost of the two-hop propagation. With a limit of 50 connections, the first quartile is 320 KB, the median is 569 KB, and the third quartile is 774 KB. Although these are more acceptable bandwidth usage numbers, the tradeoff is the fact that one-hop friends can now be

two, three, or four hops away in the social graph. The experiments show that with a limit of 50 connections, over 25% of friends are more than four-hops away. Limiting the connections therefore increase the diameter of the social graph then decreasing the number of followers that can be reached at a four-hop distance and thereby increasing the reliance of other message replication mechanisms to ensure that socially distant followers can still retrieve messages from anywhere in the social graph.

### 5.6.3 Message Replication Towards High Degree Nodes

The distributed microblogging system was designed for message distribution for close friends and friends of friends. The assumption is that by using a two-hop push/pull mechanism, Litter guarantees four-hop coverage which should cover the majority of social interested peers. However, there are some cases where a user might want anyone in the social graph to have access to a subset of their posts. In such a scenario, Litter possesses a mechanism which allows posts to propagate beyond the four-hop social radius. The first naive attempt is to use a simple random-walk-based replication. In this method, a user posts a message with a high TTL value (i.e. 50, 100, 200) representing the replication factor of the message. By increasing the TTL value based on the size of the network, a user can increase the probability that followers that are more than 4 hops away can successfully retrieve a message. In the simulations, the random-walk approach was almost a complete failure with a success rate of less than 1% when the TTL = 100.

Hence, Litter adopts a different strategy of replicating at the highest degree nodes rather than random nodes. This modification guaranteed that nodes always retrieve updates with a maximum TTL of 35 (see figure 5-5A). The algorithm is simple, at each hop of the message-walk, the current node selects the neighbor with the highest degree that has not yet been visited. There has been much research about the effectiveness of publishing and doing lookups through high degree nodes in power-law (or social) networks. The consensus is that in certain well-defined power-law networks, the




A BW Cost - Cap = 0 (PDF) B BW Cost - Cap = 25 (PDF) C BW Cost - Cap = 50 (PDF)

Figure 5-5. Probability distribution of walks. The x-axis is the number of hops it took to find the post, the y-axis is its probability. A) BW Cost - Cap = 0 (PDF). B) BW Cost - Cap = 25 (PDF). C) BW Cost - Cap = 50 (PDF).

average number of hops per lookup is on the logarithmic scale of the size of the network (i.e. $\mathrm{O(klogN)}$ where $\mathrm{N}$ is the size of the network).

By following this simple rule, figure 5-5A and 5-5B shows that about 95% of the time, a follow can retrieve a message replicated at high degree nodes with a TTL of 30. However, this is only effective when the social graph is not constrained by a limit on the number of connections. By imposing a limit on the number of connections, the success rate drops down to below 1% and therefore this mechanism is not very effective.

Overall, the analysis section provides two key insights. First, using a two-hop multicast push mechanism is costly in a 3 million user social graph which follows a power-law distribution with an average cost over 50 MB. Therefore, it is recommended to limit the number of connections per user in order to use less bandwidth. Second, by replicating messages using a propagation technique that selects the highest degree neighbor as the candidate for message replication, Litter guarantees that any user in the social graph can retrieve the message with a fairly low TTL. But this approach only works when there is not a connection limit, since the limit can distort the power-law property of the social graph.

## 5.7 Summary

This chapter demonstrates the possibility of leveraging private IP links and the social overlay formed by P2PVPNs to create a decentralized microblogging service. The Litter design starts with private message at the core of the protocol which differentiates it from other approaches. This solution also minimizes coding complexity by allowing the use of the Berkeley sockets API for communication instead of a customized P2P library interface. Therefore, with IP multicasting and UDP-based random-walks, it is possible to deliver a system which optimizes for fast, private updates to social peers. Litter also provides mechanisms for peers to publish updates beyond just friends (and friends of friends) through the use of a high-degree node replication scheme throughout the social overlay. The analysis also shows that it is important to impose limits on the number connections that a peer can maintain in order to minimize bandwidth consumption. However, the connection limits reduce the effectiveness of our replication strategy.

# CHAPTER 6
# CONCLUSION AND FUTURE WORK

This dissertation describes a peer-to-peer architecture aimed at facilitating the development and deployment of social networking applications. The goal of this architecture is to provide the core functionalities that most decentralized social networking applications depend on such as: social peer discovery and key management, direct IP connectivity and translation among social peers even through NATs and firewall, user-friendly references to user endpoints, and social identity management.

This research thus describes SocialVPN, a novel peer-to-peer virtual private network (P2PVPN) where social relationships are mapped to virtual IP connections. Instead of an unfamiliar P2P application programming interface (API), SocialVPN provides a virtual networking layer which enables private IP connectivity among social peers through encrypted P2P tunnels. This lets programmers develop social P2P applications using the well-known, and well-supported Berkeley sockets API. Consequently, legacy client-server applications can run with minimal modifications while adopting a more robust P2P model for social applications.

More specifically, the SocialVPN approach leverages existing social infrastructures to enable ad-hoc VPNs which are self-configuring, self-managing, yet maintain security amongst trusted and untrusted third parties. The key principles of the SocialVPN design are: (1) self-configuring virtual network overlays enable seamless bi-directional IP-layer connectivity to socially connected parties; (2) online social networking relationships facilitate the establishment of trust relationships among peers; and (3) both centralized and decentralized databases of social network relationships can be securely integrated into existing public-key cryptography (PKI) implementations to authenticate and encrypt end-to-end traffic flows. SocialVPN, therefore, solves the key challenge of bootstrapping trusted P2P communication links among friends and achieves this through a novel

cryptographic key exchange and verification mechanism which utilizes existing social networking infrastructures such as Facebook, or XMPP.

A natural addition to private IP channels is a domain naming system (DNS) which allows for a consistent endpoint to network services. SocialDNS enables such a system in a decentralized P2P fashion. Users are able to select domain names among themselves for their resources within their social circle for uniqueness instead of the global uniqueness enforced by the normal DNS system. When name conflicts arise in the SocialDNS system — two peers decide to choose the same domain name for a resource — a social rank-based method is used to select the mapping with the highest popularity in the social circle. This is the first decentralized domain service which introduces a social aspect to domain naming in a secure fashion. Hence, with the combination of SocialVPN, which provides IP connectivity, and SocialDNS, which allows domain name mappings to private IP addresses, this architecture greatly simplifies the development of social applications.

To demonstrate SocialVPN's ability to enable social networking applications, a decentralized microblogging service was developed, dubbed Litter. The primary goal of the Litter design is to enable fast, private updates among friends. By utilizing SocialVPN, Litter builds upon a layer where users have private IP connectivity to each other along with IP multicast support thus enabling trusted social communication on the Internet (even through NATs and firewalls). SocialVPN naturally maps to social graphs and therefore serves as a strong foundation for building decentralized social services because they specialize in low latency, private communication by creating direct, encrypted paths among friends. With this underlying social overlay, Litter lets the user control the propagation of a post in the social overlay through the use of IP multicast packets. It also leverages the properties of the social graph for data replication heuristics in order to efficiently disseminate public posts throughout the social overlay.

This application serves to demonstrate the practical benefits of building social P2P applications on top of SocialVPN.

The overall peer-to-peer architecture therefore contains the following features: 1) locally assigned private IP addresses to social peers which does not require global uniqueness, 2) the ability to map domain names to IP endpoints and share those mapping with friends without dependence on a third party (i.e. network administrator), and 3) support for both structured and unstructured P2P networks which adds robustness and versatility to the bootstrap of social connections. These features make deploying social networking applications more flexible, and independent of expensive centralized infrastructure investments which benefit developers and end users in the long run. This dissertation contributes the design of a peer-to-peer architecture for social networking applications along with a prototype implementation that works in realistic environments, accompanied with quantitative evaluation of its performance.

For future work, it is imperative to support routing over other types of networking technologies beyond the typical Ethernet or WiFi. The ability to leverage any type of connectivity among users will reduce the barriers to deployment. The challenge is to provide support for the new types of networking technologies without disrupting either the developers or the users. The transition and adaptability should be transparent in most cases. The use of networking virtualization helps with programming transparency, but the user experience should not be affected. Therefore, it will be important to create an intelligent mechanism that will decide when to select the appropriate network connection without causing any major disruption to application performance or quality of service.

A common type of network connectivity that exists among most user is Bluetooth networking. Bluetooth networking is ubiquitous ranging from cell phones to laptops. It is also a lower power method of communication than WiFi and it does not require any infrastructure which makes it ideal in ad-hoc situations. Supporting Bluetooth

connectivity in the architecture will make it a more robust solution because it adds more connection options to help users share information. From the developer standpoint, no new code will be required to take advantage of this capability.

One of the features of the architecture is message routing. With the addition of unstructured P2P networking support and the ability to route over other networking technologies such as Bluetooth, the task of selecting the most optimal route becomes more complicated due to the added dimensions. For example, the optimal route may be one which puts more weight on power consumption than on hop count. In the current architecture, the optimal route is the shortest path. In future design, each route may have a set of parameters such as the power usage or social trust. Understanding the impact of these new dimensions and integrating them into the routing algorithm is a crucial addition to this architecture.

Social routing through the unstructured P2P social graph formed by SocialVPN links is another potential avenue for future work. Unstructured P2P networks are common in popular P2P applications such as Skype, Bittorent, and Freenet due to their simplicity and robustness to P2P attacks. However, the use of an unstructured P2P network requires a probabilistic social routing mechanism for peer discovery instead of the greedy routing algorithm available in structured P2P systems. By leveraging the properties of the unstructured graph, the appropriate routing algorithm makes it possible to efficiently discover peers in the social P2P network.

The unstructured SocialVPN P2P network can be abstracted as a social graph with the small-world, scale-free characteristics of a low diameter and high clustering. Also, the node degree distribution tends to have a long tail representing the supernodes which serve as hubs ensuring a low-diameter in the social graph. By leveraging the natural characteristics of the P2P network, algorithms and heuristics can be developed to optimize the performance of social networking applications. By integrating social routing

directly into the SocialVPN architecture, any social networking application deployed on top of this framework inherits this enhancement without any code modification.

## REFERENCES


[1] I. Paul, "Microsoft bing social vs. google search plus your world: Showdown," http://www.pcworld.com/article/255476/microsoft_bing_social_vs_google_search_plus_your_world_showdown.html, May 2012.

[2] K. Hill, "Facebook privacy policy change paves way for off-facebook advertising," http://www.forbes.com/sites/kashmirhill/2012/05/11/facebook-privacy-policy-change-paves-way-for-off-facebook-advertising/, May 2012.

[3] J. Cheng, "Over 3 years later, "deleted" facebook photos are still online," http://arstechnica.com/business/2012/02/nearly-3-years-later-deleted-facebook-photos-are-still-online/, February 2012.

[4] E. V. Buskirk, "Denial-of-service attack knocks twitter offline," http://www.wired.com/business/2009/08/twitter-apparently-down/, August 2009.

[5] I. van Beijnum, "How egypt did (and your government could) shut down the internet," http://arstechnica.com/tech-policy/news/2011/01/how-egypt-or-how-your-government-could-shut-down-the-internet.ars, January 2011.

[6] J. P. Mello, "Anonymous claims attack on facebook," http://www.pcworld.com/article/256646/anonymous_claims_attack_on_facebook.html, June 2012.

[7] D. Tynan, "Facebook's 'man in the middle' attack on our data," http://www.itworld.com/it-managementstrategy/247344/facebooks-man-middle-attack-our-data, February 2012.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 407–418.

[9] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 299–314, Dec. 2002.

[10] L. Cutillo, R. Molva, and T. Strufe, "Privacy preserving social networking through decentralization," in *Wireless On-Demand Network Systems and Services, 2009. WONS 2009. Sixth International Conference on*, feb. 2009, pp. 145 –152.

[11] S. Buchegger, D. Schiöberg, L. H. Vu, and A. Datta, "Peerson: P2p social networking: early experiences and insights," in *SNS '09: Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*. New York, NY, USA: ACM, 2009, pp. 46–52.

[12] F. Rahimian, S. Girdzijauskas, A. Payberah, and S. Haridi, "Vitis: A gossip-based hybrid overlay for internet-scale publish/subscribe enabling rendezvous routing in unstructured overlay networks," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, may 2011, pp. 746 –757.

[13] S. Abbas, J. Pouwelse, D. Epema, and H. Sips, "A gossip-based distributed social networking system," in *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE '09. 18th IEEE International Workshops on*, 29 2009-july 1 2009, pp. 93 –98.

[14] "Facebook platform," http://developers.facebook.com/, 2008.

[15] C. Cooper, "Linkedin bows to twitter over tweet traffic direction," http://news.cnet.com/8301-1023_3-57464054-93/linkedin-bows-to-twitter-over-tweet-traffic-direction/, June 2012.

[16] Z. Whittaker, "Facebook applications leak users' personal data to third parties," http://www.zdnet.com/blog/igeneration/facebook-applications-leak-users-personal-data-to-third-parties/9906, May 2011.

[17] P. Saint-Andre, "Xep-0277: Microblogging over xmpp," http://xmpp.org/extensions/xep-0277.html, 2012.

[18] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, ser. Middleware '01. London, UK, UK: Springer-Verlag, 2001, pp. 329–350.

[19] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*. New York, NY, USA: Springer-Verlag New York, Inc., 2001, pp. 46–66.

[20] "Hamachi - instant, zero configuration vpn." http://https://secure.logmein.com/products/hamachi/vpn.asp?lang=en, May 2009.

[21] "Socialvpn," http://socialvpn.org/, 2010.

[22] "The p2pvpn project," http://p2pvpn.org/, 2010.

[23] "The xmpp foundation," https://xmpp.org/, 2009.

[24] M. Krochmal, "Multicast dns internet draft," http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt/, September 2008.

[25] "Multicast dns internet draft," http://technet.microsoft.com/en-us/library/cc784180(WS.10).aspx, 2010.

[26] T. Xu, Y. Chen, J. Zhao, and X. Fu, “Cuckoo: towards decentralized, socio-aware online microblogging services and data measurements,” in *Proceedings of the 2nd ACM International Workshop on Hot Topics in Planet-scale Measurement*, ser. HotPlanet ’10. New York, NY, USA: ACM, 2010, pp. 4:1–4:6.

[27] B. Ford, P. Srisuresh, and D. Kegel, “Peer-to-peer communication across network address translators,” *CoRR*, vol. abs/cs/0603074, 2006.

[28] A. Ganguly, A. Agrawal, P. Boykin, and R. Figueiredo, “Ip over p2p: enabling self-configuring virtual ip networks for grid computing,” *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pp. 10 pp.–, April 2006.

[29] “Facebook statistics,” http://www.facebook.com/press/info.php?statistics, May 2009.

[30] F. Qiu, C. Lin, and H. Yin, “Ekm: An efficient key management scheme for large-scale peer-to-peer media streaming,” vol. 4261. Berlin / Heidelberg: Springer, 2006, pp. 395–404.

[31] S. Rafaeli and D. Hutchison, “A survey of key management for secure group communication,” *ACM Comput. Surv.*, vol. 35, no. 3, pp. 309–329, 2003.

[32] X. Liu, H. Yin, C. Lin, and Y. Deng, “Efficient key management and distribution for peer-to-peer live streaming system,” 28 2007-Dec. 1 2007, pp. 638–641.

[33] X. Zhang, M. Nakae, M. J. Covington, and R. Sandhu, “Toward a usage-based security framework for collaborative computing systems,” *ACM Trans. Inf. Syst. Secur.*, vol. 11, no. 1, pp. 1–36, 2008.

[34] J. Domingo-Ferrer, “A public-key protocol for social networks with private relationships,” in *Lecture Notes in Computer Science*, vol. 4617 (Modeling Decisions for Artificial Intelligence-MDAI’2007), August 2007, pp. 373–379.

[35] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, “Measurement and analysis of online social networks,” in *Proceedings of the 5th ACM/USENIX Internet Measurement Conference (IMC’07)*, October 2007.

[36] “Openvpn - the open source vpn.” http://openvpn.net/, 2009.

[37] M. Tsugawa, A. Matsunaga, and J. A. B. Fortes, “Virtualization technologies in transnational dg,” in *dg.o ’06: Proceedings of the 2006 international conference on Digital government research*. New York, NY, USA: ACM Press, 2006, pp. 456–457.

[38] A. Sundararaj and P. Dinda, “Towards virtual networks for virtual machine grid computing,” http://citeseer.ist.psu.edu/645578.html, 2004.

[39] I. X. Jiang, “Violin: Virtual internetworking on overlay,” http://citeseer.ist.psu.edu/714412.html, 2009.

[40] L. Deri and R. Andrews, "N2n: A layer two peer-to-peer vpn." http://luca.ntop.org/n2n.pdf/, 2008.

[41] S. Aoyagi, M. Takizawa, M. Saito, H. Aida, and H. Tokuda, "Ela: a fully distributed vpn system over peer-to-peer network," Jan.-4 Feb. 2005, pp. 89–92.

[42] A. Ganguly, P. O. Boykin, D. I. Wolinsky, and R. J. Figueiredo, "Improving peer connectivity in wide-area overlays of virtual workstations," in *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2008, pp. 129–140.

[43] P. O. Boykin, J. S. A. Bridgewater, J. Kong, K. Lozev, B. Rezaei, and V. P. Roychowdhury, "Brunet software library," http://brunet.ee.ucla.edu/brunet/, 2009.

[44] "Openid," http://openid.net/, 2008.

[45] E. Rescorla and N. Modadugu, "Datagram transport layer security," http://www.rfc-editor.org/rfc/rfc4347.txt, April 2006.

[46] "Universal tun/tap driver," http://vtun.sourceforge.net/tun/index.html, 2008.

[47] 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.*, vol. 33, no. 3, pp. 3–12, 2003.

[48] "Google app engine cloud infrastructure," http://code.google.com/appengine/, 2009.

[49] M. Varvello and M. Steiner, "Traffic localization for dht-based bittorrent networks," in *Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part II*, ser. NETWORKING'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 40–53.

[50] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, no. 3, pp. 19–20, Jul. 2003.

[51] "Rfc 5389 - session traversal utilities for (nat) (stun)," http://tools.ietf.org/html/rfc5389.

[52] "Rfc 5766 - traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)," http://tools.ietf.org/html/rfc5766.

[53] "Rfc 5245 - interactive connectivity establishment (ice): A methodology for network address translator (nat) traversal for offer/answer protocols," http://tools.ietf.org/html/rfc5245.

[54] "ejabberd - the erlang jabber/xmpp dameon," http://www.ejabberd.im/.

[55] "About libjingle - google talk for developers," https://developers.google.com/talk/libjingle/.

[56] H. Jamjoom, "Virtualwire: system support for live migrating virtual networks across clouds," in *Proceedings of the 7th international workshop on Virtualization technologies in distributed computing*, ser. VTDC '13. New York, NY, USA: ACM, 2013, pp. 21–22. [Online]. Available: http://doi.acm.org/10.1145/2465829.2465838

[57] D. Williams, H. Jamjoom, and H. Weatherspoon, "The xen-blanket: virtualize once, run everywhere," in *Proceedings of the 7th ACM european conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 113–126. [Online]. Available: http://doi.acm.org/10.1145/2168836.2168849

[58] "Openvpn - open source vpn," http://openvpn.net/.

[59] "Hamachi - instant, zero configuration vpn," http://secure.logmein.com/products/hamachi/vpn.asp.

[60] "tinc wiki," http://www.tinc-vpn.org/.

[61] "Vtun - virtual tunnels over tcp/ip networks," http://vtun.sourceforge.net/.

[62] "n2n," http://www.ntop.org/products/n2n/.

[63] B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris, "Persistent personal names for globally connected mobile devices," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 233–248. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298455.1298478

[64] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "Ip over p2p: enabling self-configuring virtual ip networks for grid computing," in *Proceedings of the 20th international conference on Parallel and distributed processing*, ser. IPDPS'06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 49–49. [Online]. Available: http://dl.acm.org/citation.cfm?id=1898953.1898983

[65] "Open networking foundation," https://www.opennetworking.org/.

[66] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, ser. SOSP '01. New York, NY, USA: ACM, 2001, pp. 131–145. [Online]. Available: http://doi.acm.org/10.1145/502034.502048

[67] "Socialvpn - a free and open-source p2p vpn that connects you to your friends," http://socialvpn.org, 2009.

[68] "Important concepts - google talk for developers," https://developers.google.com/talk/libjingle/important_concepts.

[69] "Turnserver - open-source turn server implementation," http://turnserver.sourceforge.net/.

[70] G. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo, S. Smallen, W. Smith, and A. Grimshaw, *FutureGrid - a reconfigurable testbed for Cloud, HPC, and Grid Computing*, ser. CRC Computational Science. Chapman & Hall, 04/2013 2013.

[71] "Linux containers," https://linuxcontainers.org/.

[72] "Networkx - high productivity software for complex networks," http://networkx.lanl.gov/index.html.

[73] D. Wolinsky, Y. Liu, P. Juste, G. Venkatasubramanian, and R. Figueiredo, "On the design of scalable, self-configuring virtual networks," in *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, 2009, pp. 1–12.

[74] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 335–348. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000027

[75] R. Cox, A. Muthitacharoen, and R. T. Morris, "Serving dns using a peer-to-peer lookup service," in *In IPTPS*, 2002.

[76] M. Walfish, H. Balakrishnan, and S. Shenker, "Untangling the web from dns," in *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 17–17.

[77] K. Park, V. S. Pai, L. Peterson, and Z. Wang, "Codns: improving dns performance and reliability via cooperative lookups," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 14–14.

[78] "Socialdns.net project," http://socialdns.net/, 2010.

[79] M. Allman, "Personal namespaces." ACM SIGCOMM Hotnets 2007, November 2007.

[80] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '06. New York, NY, USA: ACM, 2006, pp. 267–278.

[81] M. D. Bauer, "Securing dns and bind," *Linux J.*, p. 2, 2000.

[82] "Networkx - high productivity software for complex networks," http://networkx.lanl.gov/, 2010.

[83] P. Mockapetris, “Domain names - implementation and specification,” http://tools.ietf.org/html/rfc1035, 1987.

[84] D. Liben-Nowell, J. Novak, R. Kumar, P. Raghavan, and A. Tomkins, “Geographic routing in social networks.” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 33, pp. 11 623–11 628, August 2005.

[85] E. K. Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe, “Towards ip geolocation using delay and topology measurements,” in *IMC ’06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2006, pp. 71–84.

[86] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, “Characterizing residential broadband networks,” in *IMC ’07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2007, pp. 43–56.

[87] “Amazon elastic compute cloud,” http://aws.amazon.com/ec2/, 2008.

[88] D. I. Wolinsky, P. St. Juste, P. O. Boykin, and R. Figueiredo, “Towards social profile based overlays,” University of Floriday, Tech. Rep., Feb 2010.

[89] D. Sandler, A. Mislove, A. Post, and P. Druschel, “Feedtree: sharing web micronews with peer-to-peer event notification,” in *Proceedings of the 4th international conference on Peer-to-Peer Systems*, ser. IPTPS’05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 141–151.

[90] T. Perfitt and B. Englert, “Megaphone: Fault tolerant, scalable, and trustworthy p2p microblogging,” in *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, may 2010, pp. 469 –477.

[91] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, “Scribe: a large-scale and decentralized application-level multicast infrastructure,” *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1489 – 1499, oct 2002.

[92] D. R. Sandler and D. S. Wallach, “Birds of a fethr: open, decentralized micropublishing,” in *Proceedings of the 8th international conference on Peer-to-peer systems*, ser. IPTPS’09. Berkeley, CA, USA: USENIX Association, 2009, pp. 1–1.

[93] A. Shakimov, H. Lim, R. Caceres, L. Cox, K. Li, D. Liu, and A. Varshavsky, “Vis-a-vis: Privacy-preserving online social networking via virtual individual servers,” in *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, jan. 2011, pp. 1 –10.

[94] K. Graffi, C. Gross, P. Mukherjee, A. Kovacevic, and R. Steinmetz, “Lifesocial.kom: A p2p-based platform for secure online social networks,” in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, aug. 2010, pp. 1 –2.

[95] B. Wong and S. Guha, “Quasar: a probabilistic publish-subscribe system for social networks,” in *Proceedings of the 7th international conference on Peer-to-peer systems*, ser. IPTPS’08. Berkeley, CA, USA: USENIX Association, 2008, pp. 2–2.

[96] R. Zhang and Y. C. Hu, “Borg: a hybrid protocol for scalable application-level multicast in peer-to-peer networks,” in *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, ser. NOSSDAV ’03. New York, NY, USA: ACM, 2003, pp. 172–179.

[97] T. Kusumoto, Y. Kunichika, J. Katto, and S. Okubo, “Tree-based application layer multicast using proactive route maintenance and its implementation,” in *Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming*, ser. P2PMMS’05. New York, NY, USA: ACM, 2005, pp. 49–58.

[98] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, “Measurement and analysis of online social networks,” in *Proceedings of the 5th ACM/USENIX Internet Measurement Conference (IMC’07)*, October 2007.

## BIOGRAPHICAL SKETCH

Pierre Tony St. Juste was born in Haiti on the island of Hispaniola. At the age of 13, he migrated to the US with his mom and younger brother. Once in the land of opportunity, he was encouraged by his mother to pursue the highest academic achievements. He started college in August 2001 at the University of Florida in computer engineering. He completed his bachelor's degree in 2005 and his Master of Science in 2007. He began his doctorate program in August 2007 and joined the Advanced Computing and Information Systems Laboratory under the advisement of Dr. Renato Figueiredo.

His research has been mainly focused on SocialVPN, a peer-to-peer VPN that integrates into social networking infrastructure to provide an easy-to-use trusted network to communicate with friends. His research has led him to study peer-to-peer networks, network security, virtual networking, cybersecurity, mobile and cloud computing. As a systems research, Pierre has deployed experiments with hundreds of virtual machines on cloud infrastructures such as Amazon EC2. He also has extensive experience launching and managing services on globally dispersed testbeds such as PlanetLab. His work currently provides a virtual networking solution that addresses both the needs of inter-cloud networking and mobile device communication.