SELF-MANAGING VIRTUAL NETWORKS FOR WIDE-AREA DISTRIBUTED COMPUTING

By

ARIJIT GANGULY

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

2008

To my teachers, my family and my friends.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere gratitude to my advisor Dr. Renato Figueiredo, for his constant support and guidance throughout the course of my study. Without his encouragement, appreciation and openness for research ideas, it would not have been possible to publish the defining papers of my work. I also thank him for supporting my studies with a Research Assistantship and also for the opportunity to present research papers at various conferences. I am equally thankful to Dr. Oscar Boykin, whose ideas and initial work on the Brunet P2P library have been the baseline of my research and also the development of the IPOP system. I would also like to thank Dr. Jose Fortes for giving me an opportunity to be a part of the Advanced Computing and Information Systems (ACIS) Laboratory, and do cutting-edge systems research. I am thankful to my other committee members, Dr. Richard Newman, Dr. Alan George and Dr. Paul Avery, for some stimulating discussions, and the inspiration which I drew from their research work, which in turn kindled my own creativity.

In today's competitive world, it is not possible to make an impact without team work. I am thankful to the members of ACIS P2P team for sharing with me several tools and infrastructures that have helped me in my research. In particular, I thank David Wolinsky, whose grid-appliance was very useful for gathering data for one of my papers. I thank ACIS researchers, Mauricio and Andrea, for their efforts in ensuring the high availability of lab facilities and also for their spontaneous help whenever needed . I am also thankful to Vineet and Girish, for sharing a wonderful work environment.

I am thankful to my parents, who have taught me the ability of the right judgement, and have always taken pride and expressed confidence in me. I am thankful to my sister, Inakshi, who has been encouraging at times of despair. I should also express my thankfulness to my girlfriend, Swasti for being so supportive, understanding and patient all this while. Finally, no achievement is possible without the blessings and wish of the All Mighty, who I thank for all my accomplishments and seek his blessings for the future.

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

SELF-MANAGING VIRTUAL NETWORKS FOR WIDE-AREA DISTRIBUTED COMPUTING

By

Arijit Ganguly

August 2008

Chair: Dr. Renato J. Figueiredo
Major: Computer Engineering

Sharing of computing and storage resources among different institutions and individuals connected over the Internet is seen as a solution to meet the ever-increasing computation and storage demands of modern applications. Several factors curtail the ability of existing applications to run seamlessly on Wide-area Networks (WANs): heterogeneous resource configurations, obscured access to resources due to Network Address Translators (NATs) and firewalls, inability to express sharing policies and lack of isolation provided by operating systems.

This work addresses the problem of providing bi-directional network connectivity among wide-area resources behind NATs and firewalls. At the core of the presented approach is a self-managing networking infrastructure (IPOP) that aggregates wide-area hosts into a private network with decoupled address space management, and is functionally equivalent to a Local-area network (LAN) environment where a wealth of existing, unmodified IP-based applications can be deployed. The IPOP virtual network tunnels the traffic generated by applications over a P2P-based overlay network, which handles NAT/Firewall traversal (through hole-punching techniques) and dynamically adapts its topology (through establishment of direct connections between communicating nodes) in a self-organized, decentralized manner. Together with classic virtual machine technology for software dissemination, IPOP facilitates deployment of large-scale distributed computing environments on wide-area hosts owned by different organization and individuals. A real

deployment of the system has been up and running for more than one year, providing access to computational resources for several users.

This dissertation makes the following contributions in the area of virtualization applied to wide-area networks: a novel self-organizing IP-over-P2P system with decentralized NAT traversal; decentralized self-optimization techniques to create overlay links between nodes based on traffic inspection; creation of isolated address spaces and decentralized allocation of IP addresses within each such address space using Distributed Hash Tables (DHTs); tunneling of overlay links for maintaining the overlay structure even in presence of NATs and routing outages; and techniques for proxy discovery for tunnel nodes using network coordinates.

I describe the IPOP virtual network architecture and present an evaluation of a prototype implementation using well-known network performance benchmarks and a set of distributed applications. To further facilitate deployment of IPOP, I describe techniques that allow new users to easily create and manage isolated address spaces and decentralized allocation of IP addresses within each such address space. I present generally applicable techniques that facilitate consistent routing in structured P2P systems even in presence of overlay faults, thereby benefiting different applications of these systems. In the context of the IPOP system, these techniques provide improved virtual IP connectivity. I also describe and evaluate decentralized techniques for discovering suitable proxy nodes to establish a 2-hop overlay path between virtual IP nodes, when direct communication is not possible.

CHAPTER 1
INTRODUCTION

The growth of the Internet has given an opportunity to share resources such as CPU cycles and storage capacity among different institutions connected over wide-area networks. Besides collaboration, such sharing is particularly useful to meet the ever-increasing computation and storage demands of modern applications from different domains, such as high-energy physics, medical imaging, business data analysis, among others. Several middleware solutions [1] have been proposed to facilitate resource sharing in a way that not only respects the policies defined by the resource owners, but also provides maximum flexibility to consumers. Systems have also been conceived and implemented to harness idle cycles from desktops of users connected to the Internet [2, 3]. Common to these efforts is the vision of providing computing as a utility that can be delivered by a pool of distributed resources in a seamless manner. The terms Grid and utility computing are used to refer to such systems for wide-area distributed computing.

As distributed computing moves from within an organization to a wide-area collaboration, several new challenges arise that limit the class of applications that can benefit from resource sharing, compared to local-area environments.

Firstly, resources owned by different domains tend to differ in their hardware and software configurations. Resource providers independently choose the configurations of the resources they own (including O/S kernels, libraries, and applications), which might be incompatible with application or middleware requirements. It is difficult for the application/middleware developer to maintain different distributions for every possible resource configuration [4, 5]. Secondly, the increasing use of NATs and firewall routers at Internet sites hinder bi-directional access to resources across different domains [6–8]. The network policies at each site are designed with focus on securing hosts; often, the implementation of such policies makes it difficult to support applications from external users. Although some mechanisms to access firewalled resources (for example, secure shell

tunneling) are provided, they cannot be directly used by most distributed applications. Another problem that arises when running untrusted applications from external users is providing strong isolation and security of local resources from malicious users and/or faulty applications.

To facilitate collaboration, several middleware-level solutions have been proposed. The Globus [1] toolkit provides a public-key based security infrastructure and Web-service based standards for interactions between components. BOINC [9] provides a platform for writing distributed applications that can harness CPU cycles on desktop hosts belonging to individual users connected to the Internet. However, these approaches require existing applications to be re-written with the application programming interface (API) they provide for operations such as scheduling and data transfers.. The heterogeneity of wide-area hosts further complicates middleware deployment and configuration.

In this dissertation, a novel approach to wide-area distributed computing is proposed and investigated. This approach exploits virtualization to provide to applications their preferred execution environment. The execution environment of an application consists of the machine on which it runs and the network over which it communicates with other processes, middleware components and applications. In this approach, an application executes inside a virtual machine (VM) and is connected to other applications through a virtual network that is functionally equivalent to a local-area TCP/IP network.

"Classic" VMs [10–12] enable multiple operating systems, completely isolated from each other, to time-share resources of a single machine. VMs encapsulate the software dependencies of an application consisting of the entire OS, libraries, and applications within a closed environment, which is completely decoupled from the physical host. A VM-based execution environment can be quickly instantiated on any physical host [4, 5], with the only software dependence being the presence of a virtual machine monitor (VMM).

Complementary to VMs, virtual networks [13–16] provide to applications a communication environment with decoupled IP address management and all-to-all connectivity. In a virtual network, idiosyncrasies of heterogeneous network access are handled by the virtualization layer, while applications perceive an environment that is functionally equivalent to a local-area network.

Within the context of such a virtualized distributed system, this dissertation focuses on the architecture, design, implementation and evaluation of a novel virtual networking technique. The proposed approach combines IP tunneling and peer-to-peer (P2P) overlay networking to aggregate wide-area hosts into an IP-over-P2P (IPOP) virtual network. The IPOP virtual network is architected to be scalable, fault-tolerant and self-managing, requiring minimal administrative control. The virtual network also incorporates novel techniques to protect itself from the connectivity and performance degradation due to wide-area connectivity constraints that prevent communication between pairs of nodes.

The remainder of this dissertation is organized as follows. Chapter 2 describes the IPOP architecture. Chapter 3 presents and evaluates systems which combine VM technologies and IPOP virtual networking to provide homogeneously configured Wide-area Overlays of Virtual Workstations (WOWs). Chapter 4 presents the design of a storage layer based on Distributed Hash Table (DHT) and how it can be leveraged to facilitate the deployment of WOWs. Chapter 5 describes how connectivity hazards in a Wide-area can affect consistency of P2P routing, and subsequently connectivity and performance within the IPOP virtual network. Novel techniques that facilitate consistent routing under connectivity constraints are presented in Chapter 6, and their implementation in the IPOP system is described. Chapter 7 investigates techniques to discover suitable proxy nodes to minimize end-to-end latency when direct communication is not possible, and also validates existing techniques to reduce latency of multi-hop overlay routing. Finally, the dissertation is concluded in Chapter 8.

1.1 Wide-Area Distributed Computing

1.1.1 High-Throughput Computing

For many scientific domains, the quality of results depends on the system throughput, i.e. the number of typically independent jobs (simulations with different parameter values) that can be executed over a period of time. High throughput computing (HTC) environments [17, 18] consist of a shared pool of loosely-coupled compute resources managed by middleware components [19, 20] which include job scheduler, resource monitor, among others. The system throughput can be readily increased by adding more (heterogeneous with respect to hardware configurations) compute resources. Similarly, computationally-intensive loosely-coupled parallel applications (such as master/worker applications with small communication-to-computation ratios) often achieve a lower execution time when more resources (workers) are available, irrespective of relative CPU speeds of the workers.

Adding a new resource to the shared pool requires configuring a new host with the necessary software (OS, libraries, application binaries and middleware) and also ensuring that the new resource is accessible over the network – a process that becomes non-trivial when resources are owned by multiple organizations. This dissertation investigates mechanisms by which new resources can be added to such pool with minimal manual intervention, with focus on accessibility of the new resource.

1.1.2 Cross-Domain Collaboration and Development

Within an organization, people collaborate in many different ways. For software development, there are versioning systems such as Concurrent Versioning Systems (CVS), Mercurial, among others. Users can share files and content using distributed file systems and local-area network user/group identifiers for authentication and authorization. Directory services (such as Network Information System, Active Directory) together with shared file systems (Network File System, Samba) allow users to access their data and applications from any host.

Many of these technologies do not work seamlessly in a wide-area environment. For example, collaborative software development using frameworks such as Mercurial requires each developer to maintain a repository on a machine that is accessible to other developers, and NFS over wide-area would require the file server to be running on a public machine. Despite these limitations, the popularity of these technologies among users is undeniable. New technologies are available for wide-area collaboration, but introduce new abstractions and concepts and hence hinder collaboration. Ideally, users in different domains would want to collaborate using the tools they use within their own domain and are already familiar with.

As an illustration, experiences with researchers in the coastal sciences application domain reveal that researchers in different universities (e.g. University of Florida, Virginia Institute of Marine Sciences) collaborate by exchanging data sets and prediction models. They run predictions by coupling models that run at different locations, and the simulation models often require not only an executable binary but also a variety of support tools (scripting languages, data conversion tools, geographical information systems). Users at each site are very familiar with OS level abstractions, such as Unix accounts and distributed file systems. Even though Grid frameworks and tools are available to facilitate cross-domain computation and data transfers, in such scenarios users still want to reuse their existing software infrastructure without extensive modifications.

1.2 Issues with Traditional Approaches

1.2.1 Heterogeneity of Compute Resources

Compute resources in wide-area are typically owned by multiple organizations that exercise complete control on the hardware platform and operating system. The execution environment for an application typically consists of (1) instruction set architecture (ISA) (2) operating system kernel and associated system call interface, and (3) system libraries. Although x86 has emerged as a popular instruction set architecture, the operating system distributions and runtime libraries vary considerably from system to system. This

heterogeneity is seen as a challenge to both the resource owner and application users. Resource owners independently choose the software that runs on their hosts and the need to change their resource configurations in response to an external application requirements is seen as obtrusion.

Many applications have strict software dependencies and therefore cannot be deployed on arbitrary hosts. To deal with heterogeneity, application programmers must re-write their applications to fit within the resource framework they are provided. Modifying the wealth of existing applications to make them run on remote resources is seen as an obstacle to wide-area distributed computing.

1.2.2 Obscured Internet Connectivity

The growth in the deployment of NATs/firewalls has created a situation where some nodes can create outgoing connections but cannot accept incoming connections. This lack of all-to-all connectivity is seen as a hindrance to the deployment of various distributed applications, such as desktop grid/voluntary computing systems, instant messaging, online gaming, VoIP and content sharing systems. Many of these applications rely on publicly accessible servers for dissemination of data between NATed/firewalled hosts or setting up of public proxies at each site that forward connections between NATed/firewalled hosts.

Mechanisms for NAT traversal with UDP (STUN)[21] and TCP (STUNT)[22] exist, and are being used by applications such as Skype [23]. However, these mechanisms require setting up and managing publicly reachable (STUN or STUNT) servers. Furthermore, they cannot be directly usable by existing TCP/IP applications that were not designed to use these mechanisms.

1.2.3 Lack of Isolation

A common issue in wide-area distributed computing is the potential threat posed by running untrusted codes from remote users on a local resource. This threat forces the resource owners to either restrict the class of applications that execute on their resource

(such as those signed by trusted parties), or restrict access to their resources, both of which hinder deployment of arbitrary applications on wide-area resources.

Its is equally important to isolate from each other the applications from different users that share the same resource. Traditional approaches rely on the operating systems to provide the desired security and performance isolation. However, the complexity of modern operating systems introduces several loop-holes that can be exploited by malicious applications.

1.3 Solution: Virtualization

Virtualization offers an altogether new approach to wide-area distributed computing where instead of adapting an application to constraints imposed by a wide-area resource framework, a virtualized environment preferred by the application can be provisioned on arbitrary resources. Such environments are be completely decoupled from each other and from the underlying physical environment, thus offering unprecedented security and performance isolation.

1.3.1 Virtual Machines

Classic virtual machines (e.g. VMware [12] and Xen [10]) were originally introduced by IBM (Systems/370) in 1970 to allow for time-sharing of its expensive mainframe platforms. Virtual machines allow simultaneous execution of multiple full-blown operating systems on same host (see Figure 1-1). This process is achieved through a thin layer of software called virtual machine monitor (VMM), which provides to each running operating system (guest) an abstraction of a complete hardware platform (CPU, memory, storage and peripherals). Virtual machines provide an interface equivalent to ISA offered by the underlying platform and allow most instructions issued by the guest OS and guest applications to be directly executed on the CPU, thus reducing the overhead of emulation.

Virtual machines completely decouple the software environment for an application from the underlying physical host. The entire execution environment (operating system, libraries and application binaries) can be encapsulated into a single large file (called a VM

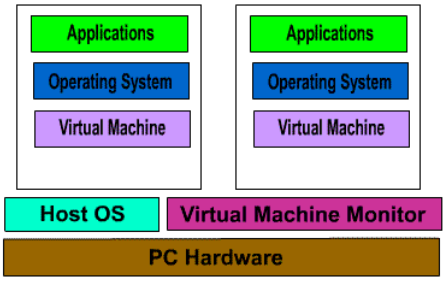


Figure 1-1. Hosted virtual machine monitor running two isolated virtual machine "guests"

image), that can be copied and instantiated on any physical host with a suitable virtual machine monitor. This *encapsulation* enables homogeneous configurations of wide-area resources using unmodified middleware and applications, without interfering with the local site policies. In [24], the authors propose to use VM images to deploy and maintain software in an organization. The entire memory, CPU state of a VM can be checkpointed and resumed, thus allowing for migration of unmodified applications. VMs confine the guest applications within a closed sandbox, which can prevent a malicious application from causing harm to the physical host resources.

A virtual machine monitor operates at a level where it has to manage only a few resources (CPU, memory and devices) as opposed to a modern OS that manage several entities (users, files, processes etc). This simplicity not only makes VMMs more secure than modern OSes, but also make resource usage policies to be easily expressed and enforced [10].

1.3.2 Virtual Networks

Virtual private networks (VPNs) are commonly used to connect remote desktop machines to local site LANs to provide access to the local resources such as printers and servers. The remote machine appears as another machine in the site LAN, and applications can run seamlessly over this virtualized network. The key technique used in VPNs is configuring a software-based network interface on the remote machine with a local IP address, and tunneling the virtual network traffic over an IP-based overlay.

In the context of wide-area distributed computing, virtual networks [13–15] decouple the network environment of an application from the physical environment and provide an opportunity to aggregate wide-area resources. The virtualized network has its own address space, and all application traffic is isolated from the physical network. The virtualization layer handles all complications relating to the presence of NATs/firewalls, thus presenting to applications an environment similar to local-area networks.

1.3.3 Combining Virtual Machines and Virtual Networks

By combining virtual machine and virtual networking technology, it is possible to provide homogeneous software environments on wide-area hosts that can be readily used by unmodified distributed applications, even when they are behind NATs/firewalls. These techniques facilitate deployment of desktop grids supporting unmodified batch schedulers, distributed file systems, and parallel application environments that are very familiar to system administrators and users.

1.4 Focus and Contributions

With multi-core architectures making their way into user desktops and virtualization built into operating systems, it is conceivable that VM-based distributed computing is going to become more prevalent in the future. Providing access to these resources in an unobtrusive manner still remains a challenge. Network virtualization [13–15] techniques exist, but impose non-trivial management overhead.

In this work, I focus on the IPOP self-managing virtual network, which uses P2P techniques for overlay routing and decentralized establishment of direct connections among nodes behind NAT/firewall routers. IPOP can scale to a large number of hosts, and is highly resilient to node and link failures. The novelty of this work lies in the application of structured P2P techniques to the different aspects of the virtual network – address space management, route discovery and NAT-traversal. In combination with classic VM technology, IPOP facilitates creation of homogeneously configured Wide-area clusters of Virtual Workstations (WOWs). These systems support execution/checkpoint/migration

of unmodified applications and middleware (Globus [1], Condor [18]), thus providing excellent infrastructure for deployment of desktop grids and cross-domain collaboration.

Within the context of the IPOP system, the key contributions of this work are listed below:

1. **Decentralized hole-punching**: Hole-punching techniques allow creation of direct UDP-communication between hosts in different private networks. The existing implementations of these techniques (such as Teredo [25] from Microsoft) require highly-available servers for out-of-band exchange of information relevant to NAT-traversal. The decentralized NAT-traversal that uses structured P2P routing for the out-of-band messaging between NATed hosts, requiring only an easy-to-manage seed network of public nodes. The implementation of this technique in the IPOP system is described in Chapter 2.

2. **Decentralized Dynamic Host Configuration Protocol (DHCP)**: To achieve virtual IP configuration of IPOP nodes, I have developed a decentralized DHCP protocol that leverages the DHT to store virtual IP leases and avoid conflicts. This protocol is described in Chapter 4.

3. **Consistent P2P routing**: Several connectivity constraints in wide-area prevent pair-wise communication between nodes, severely affecting consistency of P2P routing [26] and several applications and services that build on top of it. I have developed generally applicable techniques to facilitate consistent P2P routing in presence of overlay faults. These techniques and the subsequent improvements in routing are presented in Chapter 6.

4. **Improving latency of structured P2P routing**: Proximity Neighbor Selection (PNS) [27] is a well-known technique to reduce latency of structured P2P routing, by selecting overlay links based on proximity. To assess network proximity without explicit latency measurements, a technique called network coordinates [28] is used that allows embedding node latencies in a low-dimensional space such that the distance in the coordinate space provides an estimate of latency. I have investigated the usefulness of PNS based on network coordinates, as a means to improve route latency of the IPOP overlays. Chapter 7 describes the technique and presents analysis of a deployment on PlanetLab [29].

5. **Proxy discovery**: In wide-area distributed systems, when direct communication is not possible between end nodes, it is possible to establish a 2-hop overlay path between nodes through a suitable proxy node. In this work, I investigate decentralized techniques to discover a proxy nodes to minimize communication latency between end nodes. These techniques have been evaluated using IPOP overlays of more than 400 hosts PlanetLab – a representative testbed for the Internet (in Chapter 7).

CHAPTER 2
SELF-ORGANIZING VIRTUAL NETWORKING

The increasing use of Network Address Translators (NATs) and firewalls creates a situation that some nodes on the network can create outgoing connections, but cannot receive incoming connections. This lack of bi-directional connectivity is recognized as a hindrance to programming and deploying distributed applications [6, 7, 30]. Protocols for NAT/firewall traversal [21] exist, but require applications to be re-linked with the new protocol libraries.

Virtual networks [13–15] can provide to applications running in a virtualized infrastructure the perception of an environment functionally identical to a local-area network, despite the presence of NATs and firewall routers in the physical infrastructure. Virtual networks also confine communication of a distributed application within an environment that is logically isolated from the physical network infrastructure, thus reducing vulnerability to non-participating hosts and users at a site.

In the current techniques for network virtualization, overlay routing tables are either setup by an administrator or rely on virtual network routers/switches to have all-to-all connectivity among themselves. Hence, the process of adding, configuring and managing clients and servers that route traffic within the overlay is difficult to scale. Although topology adaptation is possible using techniques proposed in [31], adaptive routes are coordinated by a centralized server. These approaches can provide a robust overlay through redundancy. However, the effort required to preserve robustness would increase every time a new node is added and the network grows in size.

For wide-area collaborations, it is also necessary that a network virtualization technique is scalable, fault-tolerant and requires minimal administrative control. This work presents IPOP – a network virtualization technique based on IP tunneling over peer-to-peer (P2P) networks that meets these requirements. P2P networks are

self-configuring, scalable, allow user mobility, and provide extremely robust service, thus motivating the choice of P2P routing as the basis for IPOP.

The rest of this chapter is organized as follows. Section 2.1 gives an overview of P2P networks, applications and different P2P architectures. In Section 2.3, I present the IPOP architecture, and mechanisms to discover, establish and maintain overlay links between nodes behind NATs/firewalls. A comprehensive evaluation of the virtual network performance is presented in Section 6.5.

2.1 Peer-to-Peer Networks and Architectures

Traditionally, distributed computing is based on the client/server model: a central server offers various services (computation, storage, applications) and several client machines discover and access the server for these services. The server is designed to handle several clients simultaneously and typically runs on high quality hardware. For high availability of these services, it is required that the server be carefully managed to ensure continuous operation. Managing servers requires skilled IT personnel, which contributes in a significant manner to the overall costs associated with running a computing infrastructure. The system capacity can only be increased by adding more resources at the server, which also becomes a central point of failure.

As desktop computers become more and more powerful, there is an increasing interest to use resources from commodity computers at the edge of the Internet. Peer-to-peer (P2P) networks refer to distributed computing architectures that are designed to facilitate this exchange of computer resources (content, storage and CPU cycles) by direct exchange, rather than requiring intermediation from centralized servers. P2P networks are designed to function, scale and self-organize in the presence of a highly transient population of nodes, and that of network and computer failures. Applications of P2P networks include:

1. **Content distribution and file systems**: These systems and infrastructures are designed for the sharing of digital media and other data between users. Peer-to-peer content distribution systems range from relatively simple direct file sharing applications, to more sophisticated systems that create a distributed storage

medium for securely and efficiently publishing, organizing, indexing, searching, updating, and retrieving data. Examples of these systems include: Napster [32], Gnutella [33], Kazaa [34], Oceanstore [35], PAST [36], CFS [37].

2. **Communication and collaboration**: These include systems that provide the infrastructure for facilitating direct (usually real-time) communication and collaboration between peer computers. Examples include chat and instant messaging applications, such as Chat/Irc, Instant Messaging (Aol, Icq, Yahoo, Msn), and Jabber.

The first generation of P2P networks (such as the original Napster system) required centralized servers to keep track of location of different files in the P2P network. Once a file was correctly, located, it could be transferred directly without involving the server. Gnutella and Kazaa alleviated the requirement of high-capacity centralized servers by storing a distributed index of files across a set of P2P nodes. A common point among these protocols was that none of them imposed any structure on the overlay topology, and a file could potentially be located on any node in the network. These P2P networks thus incurred high overheads on searching for files. Caching techniques reduced search cost, but limited the applicability of these systems to immutable data. Nevertheless, the lack of structure also made these protocols highly resilient to churn.

In the next generation of P2P systems [38–41], the P2P nodes arrange themselves into a well-defined topology (such as a ring, or a hypercube) based on their P2P identifiers (also called P2P addresses) chosen from a large address space. The overlay topology and routing protocols bound the number of hops (with asymptotic complexity sub-linear with respect to the size of the network) between P2P nodes.

Figure 2-1 illustrates routing over a ring- structured P2P network. Each node maintains a routing table with network endpoints (IP address and port) of only a few nodes in system. At each node, the following routing rule is executed: if the destination appears in the local routing table, the message is directly communicated to the destination; otherwise it is sent to the node (in the routing table) that is closest to the destination; and in case the current node is closest, the message is delivered locally.

At each routing hop, the message gets monotonically closer to the destination in the P2P identifier space. Eventually, the message is delivered to the destination or the two nodes closest to the destination in identifier space. Structured routing is highly resilient to node failures.

To illustrate P2P overlay routing techniques in Figure 2-1(A), node 100 sends a message to address 130. The message is forwarded to node 118 and subsequently to node 128, which has node 130 in its routing table. Eventually, the message is forwarded to node 130 where it is delivered to the local application. In Figure 2-1 (B), node 100 sends a message to address 133 (destination node not present in the network); the message is routed through nodes 118, 128 and 130 to node 134 (closest to address 133), where it is delivered to the local application.

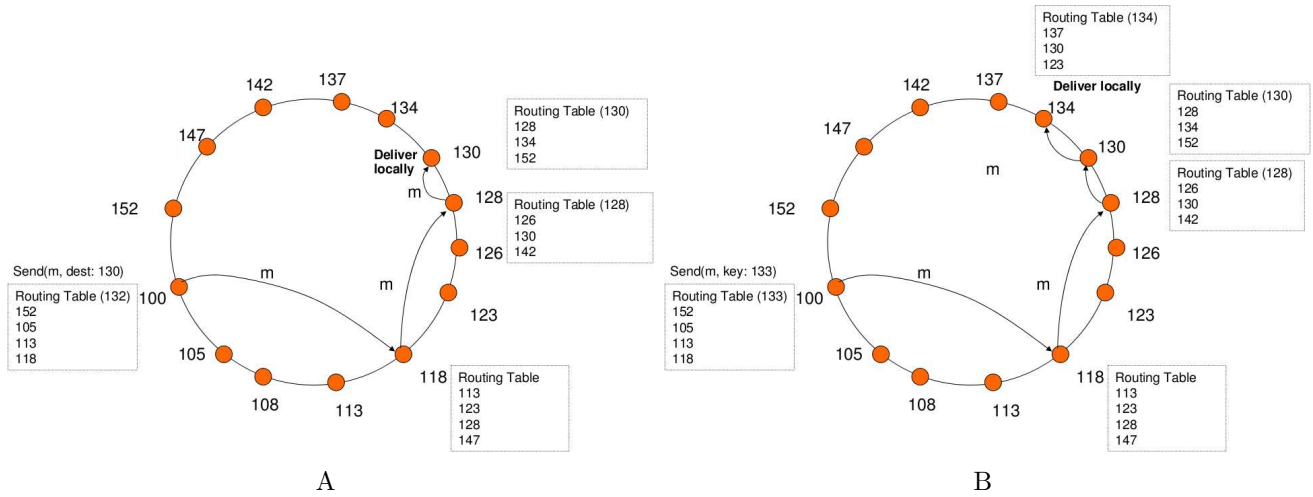


Figure 2-1. Structured P2P routing

Structured P2P systems provide an object storage facility called a Distributed Hash Table (DHT). Each object is associated with a key that belongs to the same address space as node identifiers. The ownership of keys is partitioned among participating nodes, such that each key is stored on a set of nodes that are closest to the key in the identifier space. This partitioning of key ownership together with efficient routing bounds lookup overhead for an object stored in the DHT.

Although DHTs provide efficient object lookup, there is an overhead involved in maintaining the network topology on node arrivals/departures and the subsequent changes in ownership of keys.

2.2 Tunneling IP over P2P – IPOP

This work leverages on the ability of P2P networks to resiliently route virtual IP packets between wide-area resources. The use of P2P routing to overlay virtual IP traffic differentiates IPOP from existing network virtualization solutions with respect to:

- **Scalability**: Network management in a P2P-based routing overlay scales to large numbers because routing information is naturally *self-configured*, *decentralized*, and dynamically *adaptive* in response to nodes joining and leaving. Adding a new resource to the virtual IP network requires minimal effort, which is independent of current size of the network. Performance scaling leverages from the fact that each node contributes bandwidth and processing power so that the resources of a system grow as nodes join.

- **Resiliency**: P2P networks are robust even in the face of high failure rates [42][43]. An IP-over-P2P overlay benefits from the synergy of fault-tolerant techniques applied at different levels. The IPOP overlay dynamically adapts routing of IP packets as nodes fail or leave the network; even if packets are dropped by such nodes, IP and other protocols above it in the network stack have been designed to cope with such transient failures.

- **Accessibility**: Existing network virtualization techniques can leverage mechanisms described in [21][22] to traverse NAT/firewalls. These approaches require setting up globally reachable STUN or STUNT servers that aid building the necessary NAT state by carefully crafted exchange of packets. With P2P networks, each overlay node can provide this functionality for detection of NATs and their subsequent traversal. This approach is decentralized and introduces no dedicated servers.

A useful application for IPOP is in area of Grid Computing [1], where wide-area nodes (irrespective of their physical locations), can be aggregated into a virtual IP network (Figure 2-2). The IPOP layer sits between applications (e.g. Grid clusters of physical and/or virtual machines, voice-over-IP) and physical computing nodes interconnected by existing IP networking infrastructures. This virtual network is completely decoupled from the physical network, which not only isolates the Grid application traffic, but also allows for migration of virtual IP nodes into new subnets.

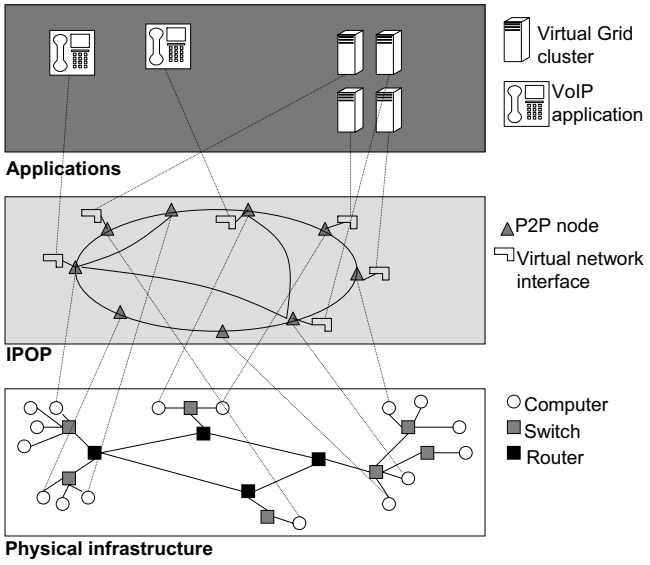


Figure 2-2. Virtualizing IP over a P2P overlay

The IPOP architecture is described next.

2.3 Architecture of IPOP

The IPOP architecture essentially has two key components (Figure 2-3): a virtualized network interface for capturing and injecting IP packets into the virtual network, and a P2P node that encapsulates, tunnels and routes packets within the overlay. IPOP builds upon a user-level framework provided by the Brunet P2P protocol suite [44], which provides mechanisms to discover, establish and maintain overlay links between nodes, even across NATs and firewalls. The next section describes how IP packets are captured/injected at the end hosts, and routed on the Brunet P2P network; and the following section will discuss mechanisms for overlay link setup and NAT/firewall traversal.

2.3.1 Virtual IP Packet Capture and Routing

The IPOP implementation uses the *tap* device, which appears as a virtual network interface inside the host and is available for read/write as a character device. In its current implementation, IPOP is a user-level C# application that uses the Brunet P2P library for overlay routing and the tap virtual device for packet capture/injection.

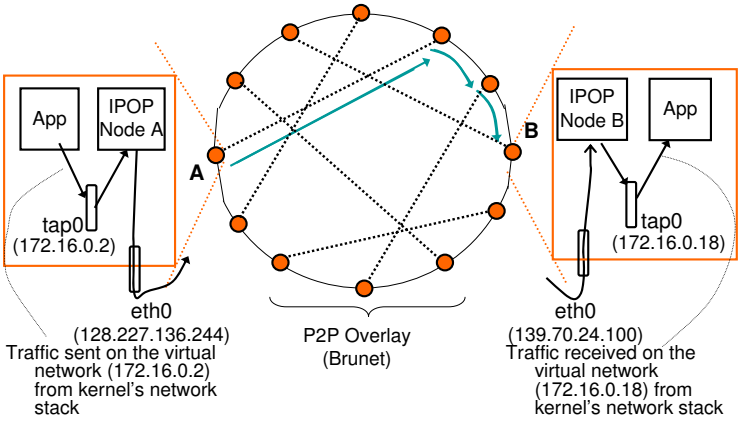


Figure 2-3. Architectural overview of IPOP

Figure 2-3 shows the flow of data between two applications communicating over the virtual IP network provided by IPOP: 1) Application (on left) sends data to a virtual IP destination (src: 172.16.0.2, dest: 172.16.0.18). 2) IPOP reads out the ethernet frame from the *tap* and extracts the virtual IP packet, 3) The virtual IP packet is encapsulated inside a P2P (Brunet) packet addressed to P2P node B (right) associated with the virtual IP destination, 4) and then routed within the P2P overlay to a destination node B. 5) At node B, IPOP extracts the virtual IP packet from the P2P packet, 6) builds an ethernet frame that it injects into the *tap*. 7) Eventually, data is delivered to application (on B). While IPOP sees Ethernet frames, it only routes IP packets; non-IP traffic, notably ARP traffic, is contained within the host.

The P2P address of an IPOP node is the 160 bit SHA-1 hash of the IP address (virtualized IP) of the *tap* device, and this is used for mapping a destination virtual IP address to P2P address (of destination IPOP node) and vice-versa.

2.3.2 Overlay Connection Management

This section describes the Brunet P2P protocol suite [44], particularly the mechanisms that enable new nodes to join an existing P2P network and connections to form between nodes behind NATs. The term "connection" is used to refer to an overlay link between P2P nodes over which packets are routed.

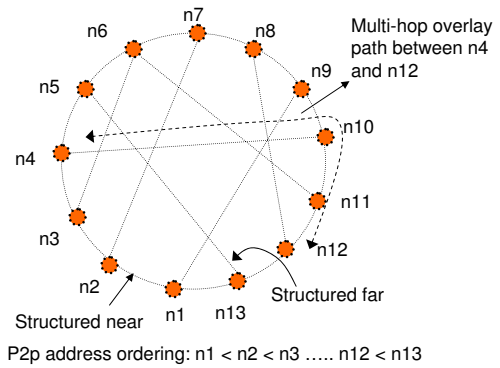


Figure 2-4. Peer-to-peer connections in Brunet

Brunet maintains a structured ring of P2P nodes ordered by 160-bit Brunet addresses (Figure 2.3.2). Each node maintains connections to its nearest neighbors in the P2P address space called *structured near* connections. [1] When a new node joins the P2P network, it must find its right position on the existing P2P ring and form *structured near* connections with its nearest neighbors in the P2P address space. From that point onwards, the node can communicate with every other node in the P2P network and vice versa. Each node also maintains k connections to distant nodes in the P2P address space called *structured far* connections, which reduces the average number of overlay hops between nodes to $O(\frac{1}{k}\log^2(n))$ for a network of size n using the algorithm of [45]. The system supports decentralized traffic inspection and bootstrapping of direct overlay connections between frequently communicating P2P nodes, which we called *shortcut* connections.

Brunet uses greedy routing of packets over structured (near and far) connections, where at each overlay hop the packet gets closer to the destination in the P2P address space. The packet is eventually delivered to the destination; or if the destination is down, it is delivered to its nearest neighbors in the P2P address space.

[1] A node also maintains structured near connections to its other close (not necessarily nearest) neighbors on the P2P ring.

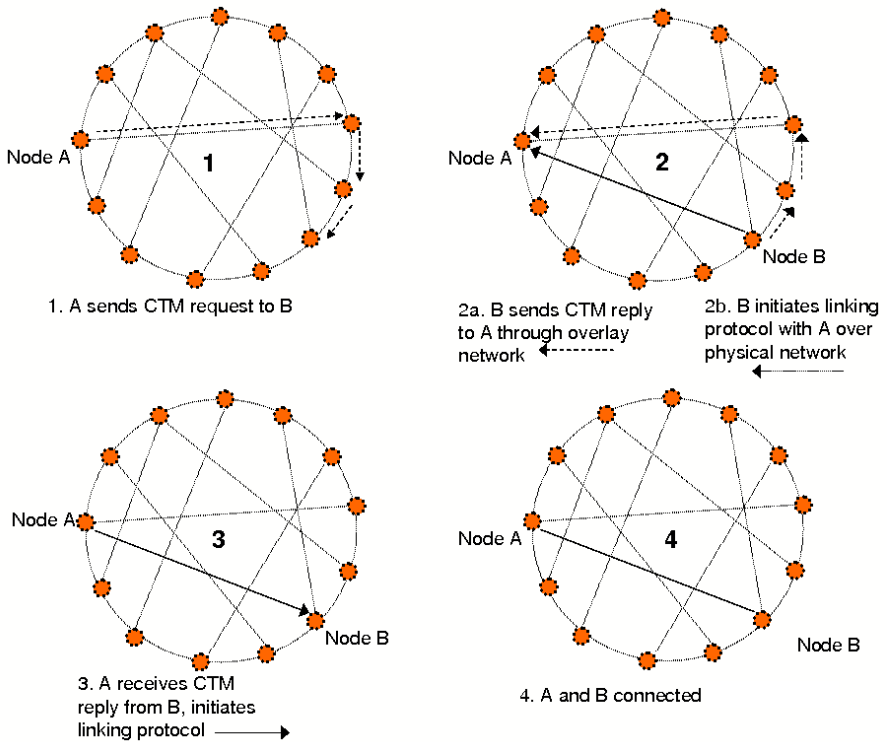


Figure 2-5. Connection setup between P2P nodes

Connections between Brunet nodes are abstracted and may operate over any transport. The information about transport protocol and the physical endpoint (e.g. IP address and port number) is contained inside a Uniform Resource Indicator (URI), such as *brunet.tcp:192.0.1.1:1024*. Note that a P2P node may have multiple URIs, if it has multiple network interfaces or if it is behind one or more levels of network address translation. The encapsulation provided by URIs provides extensibility to new connection types; currently there are implementations for TCP and UDP transports.

2.3.2.1 Connection setup

Figure 2-5 illustrates connection setup between two P2P nodes. The mechanism for connection setup between nodes consists of conveying the intent to connect, and resolution of P2P addresses to URIs followed by a linking handshake, which are summarized as follows:

1. **Connection protocol:** Through the connection protocol, nodes convey their intent to setup a connection and their list of URIs to another P2P node. A node that wishes to connect to another node in the network sends a *Connect To Me* (CTM) request message to that node. The CTM request contains the list of URIs of the initiating node. The message is routed over the P2P network and delivered to the target node, which in turn sends a CTM reply (containing its own URIs) back to the source node, and also initiates the linking protocol which is described next. The source node on getting the CTM reply initiates the linking protocol with the target node. It should be noted that the target node for a prospective connection may not be up, in which case the CTM request is delivered to its nearest neighbors which then become the targets for linking.

2. **Linking protocol:** A P2P node initiates the linking protocol when it gets a CTM request or when it gets a reply for the CTM request it sent out. Following a request/reply handshake (over the physical network) using the URIs learned in previous step, the two nodes record the new connection state which can subsequently be used for routing packets over the network. The linking protocol is complicated by the fact that not all URIs can be used to communicate with the target node. This issue is discussed in detail in Section 2.3.2.3.

Nodes keep an idle connection state alive by periodically probing each other through *ping* messages, which also involves resending of unresponded pings and exponential back-offs between resends. A succession of unresponded pings is perceived as the target node going down or a network outage, and the current node discards the connection state. These ping messages incur bandwidth and processing overhead at end nodes, which restricts the number of connections a node can maintain.

It should be noted that the linking protocol is initiated by both the peers, leading to a potential race condition that must be broken in favor of one peer succeeding while the other failing. Therefore, each node records its *active* linking attempt to the target, before sending out a link request. Now, if the current node now gets a link request from its target, it responds with a link error message stating that the target should give up its *active* attempt and let the current node go ahead with the protocol. The target node gives up its connection attempt, and eventually the current node would succeed. It is possible that both the nodes (current and target) initiate *active* linking, get link error messages

from the other and give up their attempts, in which case they restart with an exponential back-off, to reduce the likelihood of another race condition.

2.3.2.2 Joining an existing network and acquiring connections

A new P2P node is initialized with URIs of a few nodes already in the network. The new node creates a *leaf* connection with one of these initial nodes by directly using the linking protocol. In this case the linking is unidirectional and the target passively participates in the protocol. These initial nodes are typically public. If the new node is private (behind a NAT), the *leaf* connection target sees linking messages sourced at the NAT assigned public IP/port which is then used to communicate with the private node. The target node also communicates this NAT assigned IP/port back to the private node inside the packet payload. This way a newly joining node also discovers and records its own URIs corresponding to the NAT assigned IP/port. Once the *leaf* connection is established, the *leaf* target node (which is already in the ring) acts as forwarding agent for the new node. At this stage, there is still no guarantee that any packet originating at an arbitrary point in the network and addressed to the new node will be delivered to it.

The new node must now identify its correct position in the ring, and form *structured near* connections with its left and right neighbors. It sends a CTM request addressed to itself on the network through the *leaf* target. The CTM request is routed over the structured network, and (since the new node is still not in the ring) eventually delivered to its two nearest neighbors. The CTM replies received by the forwarding node are passed back to the new node. The node now knows the URIs of its nearest (left and right) neighbors and vice versa, and can form *structured near* connections with them. At this point, the node is fully routable. The time taken for a new node to join an existing P2P network of more than 130 nodes and become fully-routable is found to be of the order of seconds (see Figure 2-7).

Once the node is at its right place in the ring, it acquires *structured far* connections to other P2P nodes (as required by [45]) on the ring using protocols described in Section 2.3.2.1.

2.3.2.3 Establishing communication through NATs

The process of connection setup (linking handshake) gives an opportunity to a node to discover if any IP translation is going on, and if so learn its NAT-assigned IP address/port. As documented by the STUN[21] protocol, there are four types of NATs in common use today (described in Appendix A). Of these four types, all have the property that if a UDP packet is sent from IP address A port p_a to IP address B port p_b, the NAT device will allow packets from IP address B port p_b to flow to IP address A port p_a. In addition to the above property, three out of four of the common NAT types (all but the symmetric) use the same mapping for the NAT's port $\rightarrow$ internal *(IP, port)* pair, irrespective of the destination *(IP, port)*. The UDP transport implementation of Brunet is designed to deal with NAT traversal for large class of NAT devices found in practical deployments. The bi-directionality of linking handshake is what enables nodes punching holes into their own NATs as described in STUN [21, 46, 47]. This happens because one of the incoming packets is perceived as a reply to an outgoing packet, and allowed to pass. Furthermore, this approach is decentralized and introduces no single points of failure or dedicated servers (unlike the STUN protocol).

Based on the description of URIs presented earlier, it follows that a node inside a private network behind a NAT can have multiple URIs (corresponding to the private IP/port, and NAT assigned IP/port when it communicates with nodes on public internet). Furthermore, not all URIs can be used to communicate with it. Which URIs are usable depend on the locations of communicating nodes and the nature of NATs. For example, two nodes behind a NAT that does not support "hairpin" translation [46] can communicate only using URIs corresponding to their private IP/port, and they fail when using the NAT assigned IP/port. In contrast, two nodes behind "hairpin" NATs

can communicate using both URIs (private and NAT assigned). A public node can communicate with a node inside a private network only using the URI corresponding to the NAT assigned IP/port. During the linking protocol, nodes try each others URIs one at a time, until they find one over which they can send and receive handshake messages. The linking handshake involves the resending (with exponential back-off of the resend interval) of link requests that are not responded within a certain interval. If a node does not get responses for link requests it sends out even after a maximum number of retries, it restarts the linking handshake over the next URI in the list, until it eventually succeeds or gives up[2] .

Since private IP addresses are not unique across LANs, it is possible that trying a URI with private address leads to communication with a node other than the intended connection target. However, each node has a unique P2P address. Linking messages contain the P2P addresses of both the peers. This information is used to detect such false hits in the same LAN and suppress the linking attempt to the intended target using that URI.

2.3.2.4 Adaptive shortcut creation

Figure 2-6 shows observed latencies of as high as 1600 ms between IPOP nodes (in [16]) connected to P2P network of over 100 nodes on PlanetLab. These high latencies were due to multi-hop overlay routing through highly loaded PlanetLab nodes. This section describes the technique for decentralized adaptive shortcut creation which enables the setup of single-hop overlay links on demand, based on traffic inspection. Section 6.5 shows that shortcuts greatly reduce latency and improve bandwidth of the virtual network.

[2] Currently, the back-off factor and number of retries have been chosen conservatively in Brunet to account for highly loaded nodes in environments such as PlanetLab, which can lead to delays of the order of 150 seconds before giving up on a bad URI and trying the next one.

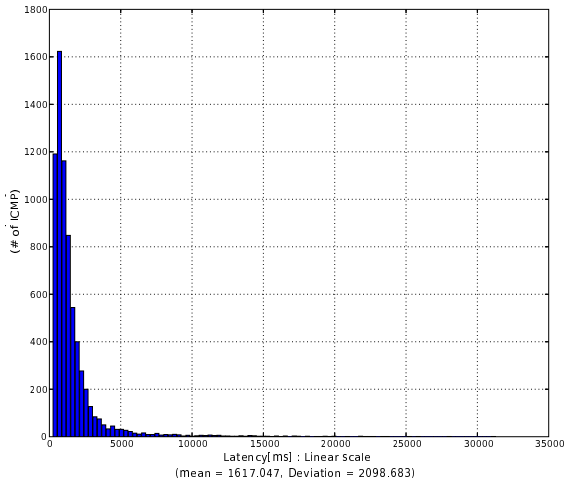


Figure 2-6. Distribution of round-trip latencies for ICMP/ping packets over 118-node Planet-Lab overlay. Two hops separate the ping source from the destination.

The Brunet P2P library is an extensible system which allows developers to add new routing protocols and connection types. For each connection type, a P2P node has a *ConnectionOverlord* that ensures the node has the right number of connections of that type.

To support shortcut P2P connections, I have implemented a *ShortcutConnectionOverlord* within the Brunet library. The *ShortcutConnectionOverlord* at a node tracks communication with other nodes using a metric called *score*. The algorithm is one based on a queueing system. The number of packets that arrive in the ith unit of time is a_i. There is a constant service rate on this work queue. The score is the amount of remaining work left in this virtual queue. If the score at time i is s_i, and the rate of service is c, it follows:

$$s_{i+1} = \max(s_i + a_i - c, 0)$$

The higher the *score* of a destination node, the more communication there has been with it. The nodes for which the virtual queue is the longest are the nodes it connect to. The *ShortcutConnectionOverlord* establishes and maintains *shortcut* connections with nodes whose *scores* exceed a certain threshold.

Although it is conceivable to create shortcuts to every other node in the network, for large overlays the bandwidth and processing overhead incurred during connection setup and maintenance poses a practical limit on the number of shortcuts a node can maintain. It is possible to determine the appropriate *score* threshold based on models that capture the relationship between number of *shortcut* connections and the cost associated with maintaining them (processing and bandwidth) . However, in the current implementation, this threshold is a constant value.

The next few sections present experimental results comparing bandwidth and latency of the virtual network with and without the adaptive shortcuts. I also report the time required for the network to adapt and create shortcuts between communicating nodes.

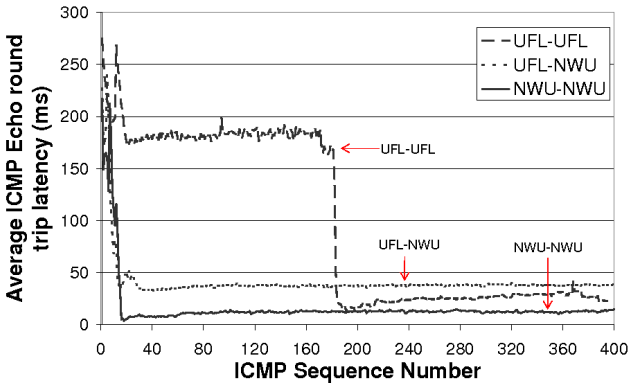
2.4 Experiments

The performance of the IPOP virtual network is evaluated with respect to 1) time taken for a new node to join an existing network and become fully-routable and eventually establish short-cut connections with nodes it is communicating with; and 2) latency and bandwidth overhead (due to packet capture and tunneling) incurred over a single IPOP link both over local and wide-area networks. Throughout this section, for brevity's sake, I refer to the Northwestern University site as "NWU" and to the University of Florida site as "UFL", Virginia Institute of Marine Sciences as "VIMS" and University of California Los Angeles as "UCLA".

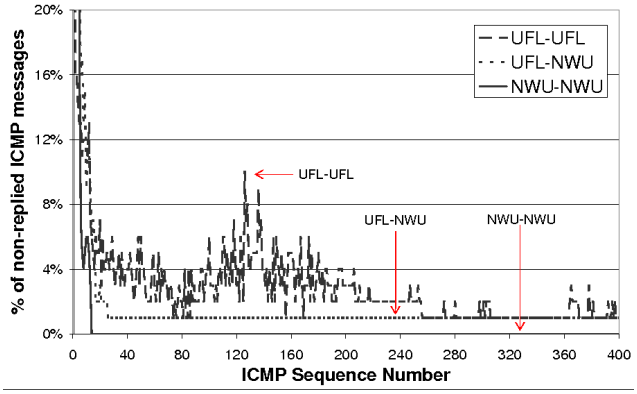
2.4.1 Shortcut Connections: Latency and Bandwidth

The focus of this experiment is on the process of joining an IPOP node to an existing overlay network (of 118 P2P nodes on PlanetLab). The PlanetLab nodes run the IPOP software for routing, but do not have "tap" virtual network interfaces attached to pick/inject packets from/into the host. The PlanetLab router nodes are used to investigate the effectiveness of the described techniques for large overlay networks. An IPOP node "A" was instantiated a priori at UFL (behind a site NAT) with a virtual IP address that remained fixed during the experiment (172.16.1.2), followed by an iterative process of: (1)

starting up an IPOP node "B" on a host at NWU; (2) sending 400 ICMP echo packets from B to A at 1 second intervals; and (3) terminating the IPOP node B. This process was repeated for 10 different virtual IP addresses (mapping B to different locations on the P2P ring), with 10 experiments conducted for each IP address, resulting in a total of 100 trials. Experiments were also conducted for two other scenarios: both nodes A and B at NWU, and both nodes at UFL. These IPOP nodes are VMs deployed on hosts whose configurations are shown in Table 3-1. The nodes in UFL were located in the same private network behind a site NAT, while the ones at NWU were in separate private networks (behind VMware NATs on two different hosts) behind a common site firewall.



A Round trip latencies



B Dropped packets

Figure 2-7. Profiles of ICMP echo round-trip latencies and dropped packets during IPOP node join

Figure 2-7 summarizes the results from this experiment. The experiment considers three combinations of the location of the node A joining the network and the node B it communicates to: UFL-UFL, UFL-NWU and NWU-NWU. 2-7A: The plot shows latencies averaged over 100 trials as reported by the ping application for packets which were not dropped. 2-7B: The plot shows the percentage of lost packets (over 100 trials) for each ICMP sequence number as reported by the ping application.

Focusing initially on the UFL-NWU case, analyzing the data for the initial tens of ICMP packets shows three different regimes (see Figure 2-8). For the first three ICMP requests, on average, 90% of the packets are dropped[3] . Within this short period, node B is unlikely to establish a route to other nodes. Between ICMP sequence numbers 4 and 32, the average percentage of lost packets steadily drops to below 1%, and the average round-trip latency is also drops from 146ms to 43 ms, with a standard deviation of 4.9 ms. These values indicate that during this period the newly joined node is very likely to be routable, and may have also established a shortcut connection with node A.[4]

Finally, between ICMP sequence numbers 33 and 400, the percentage of lost packets drops to 1%, and the round trip latencies drop to 38ms (standard deviation of 1.3ms), indicating that a shortcut connection has been established.

The UFL-UFL case also reveals the same three regimes; however, the timings differ. It takes up to about 40 ICMP ping packets before the node becomes routable over the P2P network. Furthermore, it takes about 200 ICMP packets before shortcut connections are formed. This high delay is because of the nature of the UFL NAT and the implementation

[3] Due to the y-axis scale chosen for Figure 2-7, these few initial packets do not appear on the plot.

[4] For each trial of this experiment, a sharp fall in ICMP latency was observed as soon as a shortcut connection was established. However, the time elapsed before the shortcut formed varied over the length of this regime for different trials; averaging over 100 such trials shows the gradual decline in latency and likelihood of packets being dropped.

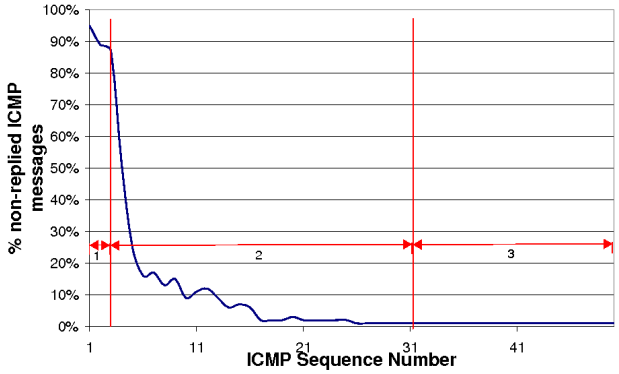


Figure 2-8. Three regimes for percentage of dropped ICMP packets in UFL-NWU (first 50 packets).

of the IPOP linking protocol, as follows. The UFL NAT does not support "hairpin" translation [46], i.e. it discards packets sourced within the private network and destined to the NAT-translated public IP/port. As described in Section 2.3.2.1, the linking handshake involves nodes trying target URIs one by one until finding one on which they can send and receive handshake messages. In IPOP, nodes first attempt the URIs corresponding to the NAT assigned public IP/port for the linking handshake during the connection setup. Because of conservative estimates for the re-send interval, the back-off factor and the number of retries for UDP tunneling, nodes take several seconds before giving up on that URI and trying the next in the list (private IP/port) on which they succeed. In Section 2.4.3, I describe implementation-level optimizations that increase the likelihood of picking the correct URI in the first attempt, thus reducing the connection setup delay in UFL-UFL case to a few seconds.

For the NWU-NWU case, the two nodes are either inside the same private network (the VMware NAT network), or on different hosts. The VMware NAT supports hairpin translation, hence both URIs for the P2P node work for connection setup. As with the UFL-NWU case, the linking protocol succeeds with the first URI it tries and hence the short cut connections are setup within a few (about 20) ICMP packets.

The bandwidth improvements over the original P2P routing by enabling the *shortcut* connection setup, between two IPOP nodes communicating over the virtual network,

Table 2-1. Bandwidth measurements between two IPOP nodes: with and without shortcuts

	Shortcuts enabled		Shortcuts disabled	
	Bandwidth Mbps	Std.dev Mbps	Bandwidth Mbps	Std.dev Mbps
UFL-UFL	12.91	0.74	0.67	0.024
UFL-NWU	10.00	1.62	0.68	0.018

were evaluated using the Test TCP (ttcp). This utility is used to measure the end-to-end bandwidth achieved in transfers of large files. Table 2-1 shows the average bandwidth and standard deviation measurements between two IPOP nodes: with and without shortcuts. The experiment considers 12 ttcp-based transfers of files of three different sizes (695MB, 50MB and 8MB) and two scenarios for the location of nodes: UFL-UFL and NWU-UFL. Two factors along the routing path limit the bandwidth between two nodes: first, the bandwidth of the overlay links, and second, the very high CPU load of machines hosting the intermediate IPOP routers, which reduces the processing throughput of the user-level IPOP implementation. Without shortcut connections, nodes communicated over a 3-hop communication path traversing the heavily loaded PlanetLab nodes and a very low bandwidth was recorded. However, with shortcuts enabled, nodes communicate over a single overlay hop, thus achieving a much higher bandwidth.

The preceding sections of this chapter have reported on the delays incurred by a new node to join the P2P network and become fully routable, and also improvements in latency and bandwidth by using shortcuts as opposed to the multi-hop routing path. Further assuming that shortcut connections are always established between two communicating IPOP nodes, the next section quantifies the latency and bandwidth overhead (due to virtualization) on a single IPOP link (using a single P2P hop) over the physical network (direct IP level communication).

2.4.2 Single IPOP link: Latency and Bandwidth

An initial analysis of the latency and throughput of IPOP links in LAN and WAN environments has been presented in previous work [16]. Based on several optimizations

Table 2-2. Configurations of machines used for evaluating performance of single IPOP link

Machine	Host type	CPU	Location
A	Physical	Pentium-4 1.8 Ghz	UFL
B	Physical	Pentium-4 1.7 Ghz	UFL
C	Virtual (VMware ESX 3.0)	Xeon 3.2 Ghz	UFL
D	Physical	Pentium-2 400 Mhz	UCLA
E	Physical	Pentium-4 3.0 Ghz	VIMS

Table 2-3. Mean and standard deviation of 10000 ping round-trip times for IPOP-UDP and physical network

		mean (msec)	std. dev (msec)
LAN	physical	0.240	0.028
(A and B)	IPOP-UDP	4.25	1.43
WAN	physical	66.13	1.20
(C and D)	IPOP-UDP	80.54	14.27

to the Brunet P2P library and IPOP, this section presents a similar comparison between IPOP-UDP and physical network. Latency was measured from the round-trip times of ICMP pings, and also by the rate of TCP request/response transactions from netperf benchmark. The bandwidth measurements were performed using iperf. The configurations of machines used for these experiments are shown in Table 2-2. The LAN experiments (latency and bandwidth) refer to machines A and B connected to the same 100 Mbps witch. The WAN latency experiments refer to machines C and D while the WAN throughput experiments refer to machines C and E connected via Abilene.

Table 2-3 summarizes the ping round-trip times for the latency experiments on an IPOP link using a single P2P hop. The latency overhead for LAN and WAN is observed to be 4 ms and 15 ms, respectively. The higher overhead on WAN is under investigation.

Table 2-4. Mean and standard deviation of rate of TCP resquest/response transactions measured over 100 samples with netperf for IPOP-UDP and physical network

		mean (trans/sec)	std. dev (trans/sec)
LAN	physical	8148.6	942.53
(A and B)	IPOP-UDP	183.6	45.19
WAN	physical	14.91	0.28
(C and D)	IPOP-UDP	13.40	0.34

Table 2-5. Comparison of throughput of a single IPOP link in LAN and WAN environments

		Abs. B/W (Mbps)	Rel. B/W (IPOP/Phys)
LAN (A and B)	physical	93.8	
	IPOP-UDP (Host A)	23.91	25%
	IPOP-UDP (Host B)	29.58	32%
WAN (C and E)	physical	13.9	
	IPOP-UDP (Host E)	12.5	90%

Latencies of the order of milliseconds/packet have also been reported in context of other user-level routing systems, such as VNET [13]. The LAN experiment provides a rough estimate of the overhead associated with the implementation of IPOP. This overhead is attributed to the traversal of kernel TCP/IP stack twice by any packet sent on the virtual network (once on the virtual interface, and additionally on the physical interface). While the relative overhead is high in the LAN environment, for the WAN used in this experiment the overhead is 31% of that of the physical network. In a WAN, the overhead of user-level routing gets amortized over the number of Internet hops (in our case, 10) that make up a P2P link.

Table 2-4 presents the rate of TCP request/reply transactions over IPOP and physical network measured with netperf with 1-byte payload. Each transaction involves sending a request to another node (which on getting the request immediately sends a response back) and waiting for a response. The netperf benchmark measures the number of such transactions one after the other that can be carried out within a time interval. The higher the latency, higher is the time per transaction and thus a lower transaction rate. Netperf measurements are representative of latencies (multiplicative inverse of transaction rate) incurred by applications which are typically based on TCP/IP. With IPOP, we observe low transaction rate on local-area (2% of physical), while on wide-area IPOP is able to achieve up to 90% of the rate on physical network.

Table 2-5 compares the throughput of a single IPOP link to that of the physical network, for both LAN and WAN scenarios. An interesting observation is that over

LAN, the bandwidth measured by the iperf client over IPOP is not the same at the two hosts. This difference is attributed to the user-level overlaying. Overall, the bandwidth is observed to be only 25% of the physical network. However over WAN, an IPOP link could harness up to 90% of the capacity of the physical network.

2.4.3 Implementation Optimizations

In the Section 2.4.1, delays greater than 150 seconds were observed for direct connection setup between two nodes in UFL behind a site NAT which does not support "hairpinning". Given the conservatively chosen timeouts for packet resends in the linking protocol, this delay happened because nodes first tried the wrong URIs (corresponding to NAT assigned IP/port of the peer) for linking. The linking protocol has been extended such that nodes try each others URIs *five* in parallel, so that they can right away try the correct URI for linking. This enhancement has resulted in UFL-UFL connection setup to be as fast (within a few seconds) as for the other scenarios.

2.5 Related Work

2.5.1 Resource Virtualization

Several initiatives [48][5][49][50][4] recognize the usefulness of virtual machines (VMware [12] [51], Xen [10]) as execution environments for Grid applications. Such environments can be deployed independent of the physical setup at each Grid site. In addition, VIOLIN [14], VNET [13, 31], and ViNe [15]) have also recognized the utility of network overlays in wide area distributed environments. In these techniques, it is necessary for administrators to set up overlay links, and a centralized entity is needed to orchestrate network adaptation in [31]. The described approach is fundamentally different in its use of a P2P-based overlay, which the nodes can join and leave in a completely decentralized, self-organizing fashion.

2.5.2 Applications of P2P

This work can be classified as applying P2P techniques to computational grids [52]. In [53] Cheema et.al and in [54] Iamnitchi et.al have investigated P2P discovery of

computational resources for grid applications. In [55] Cao et.al. have proposed a P2P approach to task scheduling in computational grids. Related to this work are the Jalapeno [56], Organic Grid [57], OurGrid [58] and ParCop [59] projects which also pursue decentralized computing using P2P technology. There is also existing body of research on various ways in which P2P systems can be applied to existing IP systems. In [60] Cox et. al. have proposed to build a Distributed DNS using DHash, a peer-to-peer distributed table built on top of Chord [40]. The IPOP system currently applies P2P techniques to achieve self-configuration of overlay network links to enable efficient and easy-to-deploy virtual private networks on which applications, including existing P2P-based systems ([36, 37]), can be deployed without concern for NAT/firewall traversal.

The use of P2P based overlay to support legacy applications has also been described in context of i3 ([61][62]). The goal is to support interoperability with new I3 applications that support multicast, anycast and mobility. In contrast, motivation of my research is to provide seamless access to Grid resources spanning different network domains by aggregating them into a virtual IP network that is completely *isolated* from the physical network.

Zhou et. al. have developed P6P [63, 64], an implementation of IPv6 on a P2P overlay. The Teredo [25] protocol developed by Microsoft tunnels IPv6 packets over IPv4 UDP packets to enable nodes behind NATs to be addressed with IPv6 connectivity. On the other hand, my focus is to enable existing grid applications (typically based on IPv4) run unmodified on wide-area. Few existing applications support IPv6.

2.5.3 Techniques for NAT-Traversal

Current implementations of NAT-traversal techniques ([46][47][21][22]) require publicly available rendezvous servers for out-of-band signalling and exchange of NAT-assigned IP address and port numbers. For example, the Teredo protocol developed by Microsoft tunnels IPv6 inside IPv4 UDP messages requires maintaining public Teredo servers; the IPv6 address of a Teredo client is derived from IPv4 addresses of the corresponding Teredo

server, which required to setup a communication session with that Teredo client. On the other hard, the Brunet P2P system incorporates decentralized NAT-traversal that uses structured P2P routing for the out-of-band messaging between NATed hosts, while using the public hosts only to discover NAT-assigned IP address and mappings.

2.6 Conclusion

In this chapter, I have described a novel network virtualization technique (IPOP), which allows aggregating resources spanning multiple domains (even behind firewalls, NATs) into a single virtual network, through the use of virtual devices and P2P networks. IPOP preserves the TCP/IP protocol stack semantics; this feature, coupled with the bidirectional connectivity it provides, enables unmodified distributed applications (written for LANs) to run seamlessly on WANs, over the virtual network. IPOP leverages the self-configuring, scalable and fault-tolerant nature of P2P networks to achieve overlay routing without centralized administrative control.

CHAPTER 3
WIDE-AREA OVERLAYS OF VIRTUAL WORKSTATIONS

In this chapter, I describe how self-configuring virtual networking through IPOP can be combined with virtual machine technology to create scalable wide-area overlay networks of virtual workstations called WOWs. These systems: (1) facilitate the addition of nodes to a pool of resources through the use of system virtual machines (VMs) and self-organizing virtual network links, (2) maintain IP connectivity even if VMs migrate across network domains; (3) present to end-users and applications an environment that is functionally identical to a local-area network or cluster of workstations [65]. By doing so, WOW nodes can be deployed independently on different domains, and WOW distributed systems can be managed and programmed just like local-area networks, reusing unmodified subsystems such as batch schedulers, distributed file systems, and parallel application environments that are very familiar to system administrators and users.

Furthermore, WOW nodes can be packaged as VM "appliances" [24] that can be instantiated without disrupting the configuration of existing, commodity desktops with a variety of hosted I/O virtualization technologies (e.g. VMware, Parallels, Linux KVM). These characteristics make WOWs an excellent infrastructure for the deployment of desktop grids that support not only applications designed for such environments, as in [2, 3, 66, 67] and systems based on BOINC [9], but also complex, full-fledged O/S environments with unmodified software and middleware components (e.g. Condor [18–20]).

Experiments with a realistic deployment consisting of 118 router nodes on PlanetLab [29] and 33 compute nodes across six different firewalled domains (Figure 3-1), demonstrate the ability of WOWs (1) to establish direct overlay links between IPOP nodes (2) to support existing middleware and compute-intensive applications and deliver good performance, and (3) to autonomously re-establish virtual network links after a VM migrates across a wide-area network, and successfully resume the execution of TCP/IP client/server applications in a manner that is completely transparent to the applications.

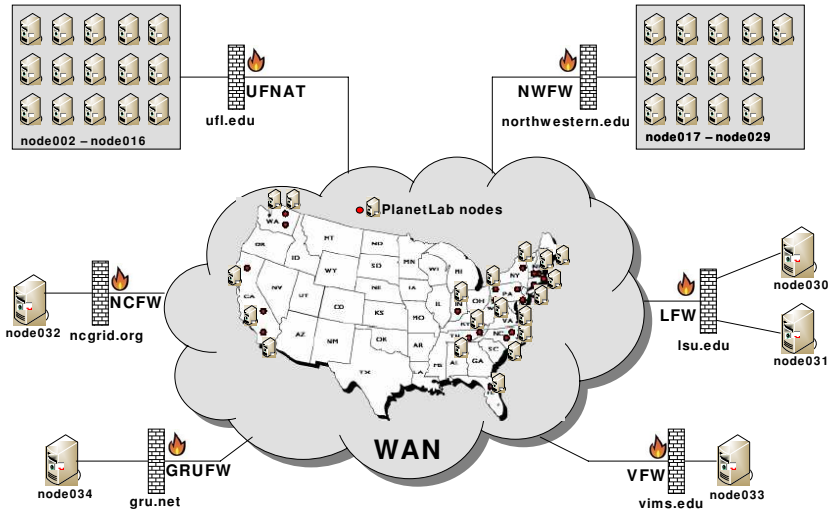


Figure 3-1. Testbed WOW used for experiments

The rest of this chapter is organized as follows. In 3.1, I describe how WOW nodes are configured and deployed. Section 3.1.4 evaluates a testbed WOW prototype. A case for using WOWs to set up ad hoc Condor pools for high-throughput computing is presented in Section 3.3. I describe related work in Section 3.2. In Section 3.4, I list the various deficiencies in current IPOP prototype that hinder the deployment of WOWs by new users; solutions to these problems are presented in subsequent chapters of this dissertation.

3.1 Node Configuration and Deployment in WOWs

3.1.1 Background and Motivations

Commodity machines connected by local-area networks are very flexible and cost-efficient resources to run high-throughput computing and parallel applications. Scaling beyond a LAN, Grid efforts address important issues that arise when resources are shared among "virtual organizations" [1]. At the resource level, system configuration heterogeneity and difficulty to establish connectivity among machines due to the increasing use of NATs/firewalls [7] substantially hinder sharing of resources. WOWs are designed to facilitate the aggregation of resources in an environment where systems in different domains have different hardware and software configurations and are subject to different

machine and network administration policies. Resources aggregated as WOWs can then be managed by existing, unmodified Grid middleware (e.g. for scheduling and access control), allowing each site to specify usage policies as necessary.

Virtualization allows for isolated, flexible and efficient multiplexing of resources of a shared, distributed infrastructure [48]. With the use of VMs, the native or preferred software environment for applications can be instantiated on any physical host, replicated to form virtual clusters [68], and checkpointed/migrated [69] to enable unique opportunities for load balancing and fault tolerance.

3.1.2 Virtual Machine Configuration

WOW nodes are deployed in a manner that shares common characteristics with cluster computing environments, thus configuring and deploying a WOW is a process that system administrators are for the most part already familiar with. A base computing node is configured with a desired execution environment, e.g. a particular Linux distribution and ancillary libraries, system utilities and application software. Each node is an independent computer which has its own IP address on a private network. While in a typical LAN cluster each physical node's disk is loaded with the desired execution environment and nodes within the cluster rack are connected by hardware switches, in WOW a virtual disk is configured and copied to all hosts, and nodes across a WAN are interconnected by a software overlay. The only software needed within the VM that is additional to a typical cluster environment is the IPOP [16] virtual network consisting of: mono .NET runtime environment, a "tap" device (Figure 2-3), and a short (tens of lines) configuration script to launch IPOP and setup the VM with an IP address on the overlay. The configuration script specifies the location of at least one IPOP node on the public Internet to establish P2P connections with other nodes. Currently, we use an overlay deployed on PlanetLab for this purpose.

One important advantage of a virtual network supporting seamless NAT traversal is that in cases where the VM monitor provides a NAT-based virtual network interface

(e.g. VMware and Xen) the WOW VM does not require a routable IP address at the deployment site. This greatly facilitates deployment, since many host networks impose limits on IP address allocations, due to address space shortage, lack of DHCP capabilities, or administrative policies.

Alternatively, it is also possible to run IPOP on the physical host that hosts the VM, and still be able to capture/inject virtual IP traffic. The VM's ethernet interface has an IP address from the virtual address space, and all network virtualization mechanisms take place outside the VM. At the cost of extra configuration (installing IPOP) on the host, such a model completely confines the VM traffic within a virtual network.

3.1.3 Deployment Scenarios

The goal of my work is to make the addition of a node to a pool of Grid resources as simple as instantiating a pre-configured VM image that can be easily disseminated by file transfer or through distribution of physical media. I envision WOW deployments where a VM "appliance" [24] is configured once, then copied and deployed across many resources, facilitating the deployment of open environments for grid computing similar in nature to efforts such as the Open Science Grid (OSG [70]). WOW allows participants to add resources in a fully decentralized manner that imposes very little administrative overhead.

An illustrative use case example of WOW techniques is a VM appliance [71] that self-configures Condor [18] pools on wide-area hosts. The VM configuration is based on a Linux 2.16 kernel and a Debian distribution that is customized to optimize the VM image size, Condor 6.8.20, and the IPOP runtime[1] . The VM images are configured to automatically obtain a WOW IP address and Condor configuration files at boot time from a DHCP server managed by a central entity. The Condor configuration points every new VM deployed on a WOW to an available central manager as its "primary" pool, as well as connecting to several other managers to which jobs can "flock". Because the IP address

[1] This VM image is available for download from http://www.grid-appliance.org.

space and the namespace within a WOW is virtualized and non-firewalled, the automatic configuration of Condor pools and flocking is straightforward, and worker nodes can be added independently to the system without any administrative overhead.

3.1.4 Testbed WOW Performance

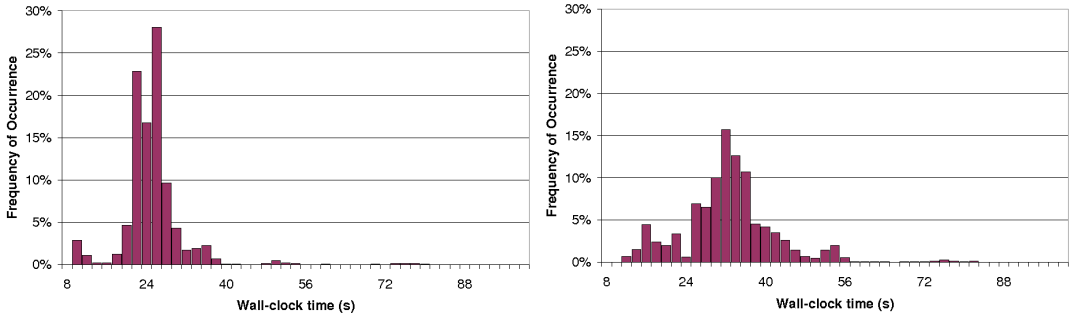
The WOW performance was measured using a realistic deployment consisting of 33 VMs configured with the same Debian/Linux based O/S, thus providing a homogeneous software environment within the cluster. These VMs were instantiated on top of a highly heterogeneous physical environment, consisting of hosts running different operating systems (Linux and Windows) and different VM monitors, located in different network domains, and subject to different firewall policies. Table 3-1 details the configuration of the various compute nodes of the testbed illustrated in Figure 3-1. The WOW has 33 compute nodes, 32 of which are hosted in universities and behind at least one level of NAT and/or firewall routers: 16 nodes in Florida; 13 in Illinois (Northwestern U.); 2 in Louisiana; and 1 node each in Virginia and North Carolina (VIMS and UNC). Node 34 is in a home network, behind multiple NATs (VMware, wireless router, and ISP provider). A total of 118 P2P router nodes which run on 20 PlanetLab hosts are also part of the overlay network, to provide a "bootstrap" overlay running on public-address Internet nodes, to which nodes behind firewalls could connect[2] .

With the only exception of the ncgrid.org firewall, which had a single UDP port open to allow IPOP traffic, no firewall changes needed to be implemented by system administrators. Furthermore, none of the sites (except UFL) provided DHCP capabilities and WOW nodes over there used VMware NAT devices, which do not require an IP address to be allocated by the site administrator.

[2] The PlanetLab router nodes were used to investigate the effectiveness of the various IPOP components for a large overlay network. WOWs can also be deployed such that overlay routing is distributed across IPOP running on compute nodes.

Table 3-1. Configuration of the WOW testbed depicted in Figure 3-1. All WOW guests run the same Debian/Linux 2.4.27-2 O/S

Node number	Physical Domain	Host CPU	Host O/S	VM monitor (VMware)
node002	ufl.edu	Xeon 2.4GHz	Linux 2.4.20-20.7smp	Workstation 5.5
node003-node016	ufl.edu	Xeon 2.4GHz	Linux 2.4.20-20.7smp	GSX 2.5.1
node017-node029	northwestern.edu	Xeon 2.0GHz	Linux 2.4.20-8smp	GSX 2.5.1
node030-node031	lsu.edu	Xeon 3.2GHz	Linux 2.4.26	GSX 3.0.0
node032	ncgrid.org	Pentium-3 1.3GHz	Linux 2.4.21-20.ELsmp	VMPlayer 1.0.0
node033	vims.edu	Xeon 3.2GHz	Linux 2.4.31	GSX 3.2.0
node034	gru.net	Pentium-4 1.7GHz	Windows XP SP2	VMPlayer 1.0.0



A Shortcuts enabled B Shortcuts disabled

Figure 3-2. Frequency distributions of PBS/MEME job wall clock times

I chose two representative life-science applications as benchmarks: MEME [72] version 3.5.0 and fastDNAml-p [73, 74] version 1.2.2. These applications ran, without any modifications, on the 33-node WOW; scheduling, data transfer and parallel programming run-time middleware also ran unmodified, including OpenPBS [75] version 2.3.16, PVM [76] version 3.4.5, SSH, RSH and NFS version 3.

In the PBS experiment, one of the WOW VMs (node002) was configured as the cluster head node and the rest were configured as worker nodes. In the PVM experiments,

the virtual IP addresses of the worker VMs were provided through a PVM console at the master node (node002).

The experiments were designed to benchmark my implementation for classes of target applications for WOW: high-throughput independent tasks and parallel applications with high computation-to-communication ratios. Specifically, the goals of the experiments are: (1) to show that WOWs can deliver good throughput and parallel speedups, (2) to quantify the performance improvements due to shortcut connections, and (3) to provide qualitative insights on the deployment, use and stability of the IPOP system in a realistic environment.

3.1.4.1 Batch application

MEME [72] is a compute-intensive application that implements an algorithm to discover one or more motifs in a collection of DNA or protein sequences. In this experiment, I consider the execution of a large number (4000) of short-running MEME sequential jobs (approximately 30s each) queued and scheduled by PBS. The jobs run with the same set of input files and arguments, and are submitted at a frequency of 1 job/second at the PBS head node. Jobs read and write input and output files to an NFS file system mounted from the head node.

For the scenario where shortcut connections were enabled, the overall wall-clock time to finish the 4000 jobs was 4565s, and the average throughput of the WOW was 53 jobs per minute. Figure 3-2 shows a detailed analysis of the distribution of job execution times, for both the cases where the WOW had shortcut connection establishment enabled and disabled. The variation in job execution times shown in the histogram can be qualitatively explained with the help of Table 3-1: most physical machines in the WOW prototype have 2.4GHz Pentium-4 CPUs; a couple of them (nodes 32 and 34) are noticeably slower, and three of them are noticeably faster (nodes 30, 31 and 33). Overall, the slower nodes end up running a substantially smaller number of jobs than the fastest nodes (node 32 runs 1.6% of the jobs, while node 33 runs 4.2%).

Figure 3-2 also shows that the use of shortcut connections decreases both the average and the relative standard deviation of the job execution times. The wall clock time average and standard deviation are 24.1s and 6.5s (shortcuts enabled) and 32.2s and 9.7s (shortcuts disabled). The use of shortcuts also reduced queuing delays in the PBS head node, which resulted in substantial throughput improvement, from 22 jobs/minute (without shortcut connections) to 53 jobs/minute (with shortcut connections).

Note that the throughput achieved by the deployed system depends not only on the performance of the overlay, but also on the performance of the scheduling and data transfer software that runs on it (PBS, NFS). The choice of different middleware implementations running inside WOW (e.g. Condor, Globus) can lead to different throughput values. The VM technology in use also impacts performance. The average execution time for MEME application inside a VM was observed to be 13% higher than that of a physical host.

3.1.4.2 Parallel application

FastDNAml is a program for the maximum likelihood inference of phylogenetic trees from DNA sequence data [73, 74]. The parallel implementation of fastDNAml over PVM is based on a master-workers model, where the master maintains a task pool and dispatches tasks to workers dynamically. It has a high computation-to-communication ratio and, due to the dynamic nature of its task dispatching, it tolerates performance heterogeneities among computing nodes.

Table 3-2 summarizes the results of this experiment for the 50-taxa input dataset reported in [74]. The parallel execution of fastDNAml on the WOW reduces significantly the execution time. The fastest execution is achieved on 30 nodes when the WOW has shortcut connections enabled: 24% faster than 30 nodes without shortcuts enabled, and 49% faster than the 15-node execution. Even though fastDNAml has a high computation-to-communication ratio for each task, the use of shortcuts resulted in substantial performance improvements. While I have not profiled where time is spent

Table 3-2. Execution times and speedups for the execution of fastDNAml-PVM in 1, 15 and 30 nodes of the WOW. The sequential execution time for the application is 22,272 seconds on node 2 and 45,191 seconds on node 34. Parallel speedups are reported with respect to the execution time of node 2.

	Parallel Execution		
	15 Nodes	30 Nodes	
	Shortcuts enabled	Shortcuts disabled	Shortcuts enabled
Execution time (seconds)	2439	2033	1642
Parallel speedup (with respect to node 2)	9.1	11.0	13.6

within the application during its execution time, increase in execution times can be explained by the fact that the application needs to synchronize many times during its execution, to select the best tree at each round of tree optimization [74].

The sequential execution times of fastDNAml are reported for two different nodes (node002 and node034) and show that the differences in the hardware configuration of the individual nodes of the WOW result in substantial performance differences. While modeling parallel speedups in such a heterogeneous environment is difficult, I report on the speedups with respect to a node which has the hardware setup most common in the network. The parallel speedup computed under this assumption is 13.6x; in comparison, the speedup reported in [74] is approximately 23x, but is achieved in a homogeneous IBM RS/6000 SP cluster within a LAN.

3.1.4.3 Virtual machine migration

VMs provide a unique opportunity to migrate unmodified applications between hosts [69, 77, 78]. However, when a VM migrates it also carries along its connection state. Such connection state can also accumulate inside other hosts with which it is communicating. This forces the VM to retain its network identity, which in turn hampers VM migration between subnets. Virtual networking provides the opportunity of maintaining a consistent network identity for a VM, even when it migrates to a different network.

The network virtualization layer of WOW is designed such that, when a VM migrates to another subnet, the connection state for the virtual network interface continues to remain valid. However, the physical network state involving the P2P overlay connections needs to be invalidated. In the case of IPOP, this can be done in a very simple manner – by simply killing and restarting the user-level IPOP program. The IPOP node then rejoins the overlay network autonomously, creating structured connections on the P2P ring following the same process described in Section 2.3.2. Clearly, packets do not get routed and are dropped until the node rejoins the P2P network; this short period of no routability is approximately 8 minutes for the 150-node network used in the setup. The TCP transport and applications are resilient to such temporary network outages, as the following experiments show.

1. **SSH/SCP file transfer:** In this experiment, a client VM is with a VMware NAT interface is instantiated at NWU. The file server was located in the UFL private network. The client VM established an SSH/SCP-based file transfer session with the server, and started to download a 720MByte file. During the transfer, at around 200s of elapsed transfer time, the migration was initiated: the IPOP process on the file server was killed, the VM was suspended, its virtual memory and disk copy-on-write logs were transferred to the host at NWU, and the VM was then resumed.
   When the VM was resumed, its virtual eth0 network interface was restarted, and because the NATs that the VM connected to were different at UFL and NWU, the VM acquired a new physical address for eth0 at the destination. However, the virtual tap0 interface did not need to be restarted and remained with the same identity on the overlay network. Then IPOP was restarted; seconds later, the SCP server VM again became routable over the P2P network, then established a shortcut connection with the SCP client VM, and eventually the SCP file transfer resumed from the point it had stalled. The sustained transfer bandwidths before and after migration are 1.36MB/s and 1.83MB/s, respectively.

2. **PBS job submission:** This experiment considers migration of a PBS worker VM across subnets. The PBS head node VM was configured in the UFL private network, and 2 worker VMs were configured on two different hosts also in the UFL private network. Communication within these nodes was performed over the IPOP virtual network.
   This experiment simulates a use case of applying migration to improve load balancing: background load was introduced to a VM host resulting in an increase in the execution time of applications executing on the VM guest; the VM guest was then migrated from UFL to a different host at NWU. IPOP was restarted on the

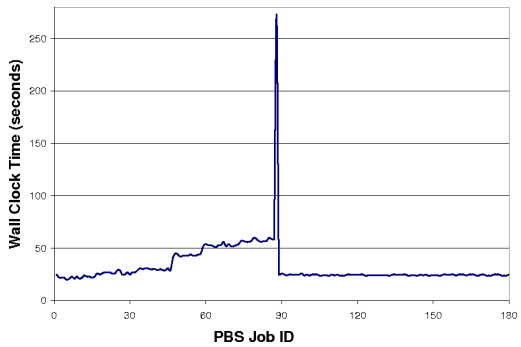


Figure 3-3. Profile of execution times for PBS-scheduled MEME sequential jobs during the migration of a worker node

guest upon VM resume. The job that was running on the VM continued to work without problems, and eventually committed its output data to the NFS-mounted home directory of the account used in the experiment. While the runtime for the job that was "in transit" during the migration was increased substantially due to the WAN migration delay. Once PBS started submitting jobs to the VM running on an unloaded host, it was observed that the job runtimes decreased with respect to the loaded host. This experiment also showed that the NFS and PBS client/server implementations were tolerant to the period with lack of connectivity. Figure 3-3 summarizes the results from this experiment. Job IDs 1 through 87 run on a VM at UFL. During job ID 88, the VM is migrated to NWU. Job ID 88 is impacted by the wide-area migration latency of hundreds of seconds, but completes successfully. Subsequent jobs scheduled by PBS also run successfully on the migrated VM, without requiring any application reconfiguration or restart.

3.2 Related Work

WOWs combine virtual machine and virtual networking technologies to build homogeneously configured wide-area clusters of VMs running on commodity workstations, which can then be used to deploy existing middleware and applications. This relates my work to the Berkeley NOW project [65] and Beowulf [79], which are very successful efforts at using commodity machines and networks for high performance distributed computing. Rather than supporting tightly-coupled parallel computation within a local-area or cluster network, the aim of WOW is to support high-throughput computing and cross-domain collaborations. The DAS [80] project built a distributed cluster based on homogeneously configured commodity nodes across five Dutch universities. Also related to my work is the Ibis project [30, 81] which lets applications span multiple sites of a grid, and copes

with firewalls, local IP addresses, secure communication, and TCP bandwidth problems. Similar to WOWs, combination of virtual machine and virtual networking techniques to provide isolation and management decoupling from individual Grid sites has also been described in [82].

Several efforts on large-scale distributed computing have focused on aggregating wide-area resources to support high-throughput computing, but at the expense of requiring applications to be designed from scratch [2, 3, 66, 67, 83]. Legion [84] is a system also designed to scale to large numbers of machines and cross administrative domains. Globus [1] provides a security infrastructure for federated systems and supports several services for resource, data and information management. Condor [18–20] has been highly successful at delivering high-throughput computing to large number of users.

My work differs from these approaches in that it is an end-to-end approach to providing a fully connected wide-area cluster environment for running unmodified applications. Nonetheless, the use of virtualization makes my approach one that does not preclude the use of any of these systems. Quite the contrary, because virtualization enables us to run unmodified systems software, it is possible to readily reuse existing, mature middleware implementations when applicable, and rapidly integrate future techniques.

In [82], authors describe a virtualized infrastructures for wide-area distributed computing based on virtual machines and virtual networking. The key distinguishing feature of my approach is the use of peer-to-peer (P2P) techniques for overlay routing and establishment of overlay connections among nodes behind NAT/firewall routers in a highly scalable manner.

3.3 Discussion

The WOW techniques [71] have been used to deploy a resource pool for running compute-intensive jobs through Condor. The pool consists of more than 80 nodes in several NATed/firewalled domains, and runs jobs submitted from nanoHub [17] and users

at University of Florida. Here, I present some of the qualitative insights obtained from the practical usage of this system.

The deployment of this pool has been greatly facilitated by packaging all the software (IPOP, Condor middleware) within the VM and requiring only a NAT network, as well as by the availability of free x86-based VM monitors, notably VMPlayer and VMware Server. Once a base VM image was created, replicating and instantiating new nodes was quite simple. The VM image can be download and instantiated by ordinary users on their desktops. Once instantiated, the VM automatically becomes the part of the shared pool. Through this VM, users can submit jobs to the shared pool and also run jobs from other users.

The system has been observed to be tolerant to physical nodes failures – neighbors of a failed node respond by creating connections to other nodes and thus maintain routability within the network. have been shut down and restarted during this period of time. The overlay network has also exhibited resiliency to changes in NAT IP/port translations. IPOP is able to deal with these translation changes autonomously by detecting broken links and re-establishing them using the connection techniques discussed in Section 2.3.2.

An application of WOW techniques within a LAN is facilitating deployment of Condor pools at University Computer Centers consisting of hundreds of idle desktops, usually running the Windows operating system. Condor middleware runs on Linux, and can be deployed inside Linux VM container on Windows machines. However, to be able to communicate with each other, these VMs need unique, routable IP addresses. Managing additional IP addresses for VMs can be very difficult for administrators, because many new VMs can be readily instantiated by users from a VM image, and can also be easily migrated across physical hosts. The decentralized NAT-traversal support in IPOP allows instantiation of these VMs behind NAT interfaces provided by VMMs (such as VMware and Xen), and be able to provide connectivity between them without requiring a routable IP addresses for VMs.

3.4 Making Deployment of WOWs Simpler

Although the process of adding new nodes to an existing WOW requires a simple procedure (copying and instantiating a VM image) and can be performed in a matter of seconds [85], the deployment of WOWs by new users is still hindered by:

1. **Management of virtual IP addresses**: The original version of IPOP required static assignment of IP addresses. In [71], IPOP supports dynamic virtual IP configuration using unmodified DHCP clients, by capturing DHCP packets from the *tap* interface, and making requests to SOAP server that maintains virtual IP leases. With the virtual network provided by IPOP potentially involving hosts spanning wide-area networks and owned by multiple organizations, maintaining such dedicated DHCP servers is difficult. Moreover, dedicated servers introduce central points of failures.

2. **Separate overlay per virtual network**: Each virtual private network of VMs must have a separate P2P overlay. This requires creating a bootstrap network of public nodes and initializing each IPOP node with the addresses of these bootstrapping nodes. This non-trivial effort hinders easy deployment of new WOWs.

Furthermore, earlier versions of IPOP [16, 71] have also suffered from limitations with respect to:

1. **Mapping from virtual IP to P2P address**: The P2P address of each IPOP instance was originally the SHA-1 hash of the virtual IP address, which enabled nodes to quickly and independently determine the address of a P2P node on the destination VM. However, because of the one-to-one mapping from virtual IP to P2P address, a single P2P node on the host cannot route for multiple VMs inside the virtual network. When virtual IP addresses are mobile – a situation that can occur when virtual machines are allowed to migrate ([78], [13]), thus requires killing and restarting the P2P node on the target host as shown in [85].

2. **Resilience to overlay faults**: Although Brunet P2P library provides robust support for NAT traversal, symmetric NATs and Internet route outages can still create situations where pairs of nodes cannot communicate with each other and form a connection. The effect of these outages is two fold. Firstly, the inability to create *near* connections can result in inconsistent view of the local neighborhood (in P2P identifier space) at a node, leading to incorrect routing decisions at that node. This inability of overlay nodes to correctly route messages affects the all-to-all virtual IP connectivity between nodes. Secondly, these outages prevent creation of *shortcut* connections between virtual IP nodes, communication between such nodes is routed through other P2P nodes. Since structured P2P routing is oblivious to the load and geographical location of nodes, this multi-hop routing suffers from poor latency and bandwidth.

These limitations motivated me to conduct research in a variety of different areas: (1) exploring DHT-based techniques for managing virtual IP addresses within a WOW, (2) techniques to facilitate overlay structure maintenance and routing in connectivity constrained wide-area environments and (3) using network coordinates [28] and resource discovery to find proxy nodes that can route communication between virtual IP nodes when *shortcut* connections cannot form. Subsequent chapters of this dissertation describe and evaluate these techniques.

In the next chapter, I describe an implementation of a DHT over the Brunet P2P system and how it can be leveraged to make deployment of WOWs easier for new users.

CHAPTER 4
DISTRIBUTED HASH TABLES AND APPLICATIONS IN IPOP

Enterprise information systems involve packing and storing large amounts of storage devices throughout a series of shelves in a room, all linked together. The information in these storage systems can be accessed by a supercomputer, mainframe computer, or personal computer. These systems can only be accessed by authorized users and require constant attention and management by experts within an organization. The management activities include keeping the system up and running, hardware and software upgrades, backup and disaster recovery.

In a wide-area environment that involves several organizations and individual users, using such centralized systems poses several issues: Who manages the system? What should be the targeted system capacity? Is the system accessible to all users?

Architectures based on self-managing peer-to-peer storage (CFS [37],PAST [36]) have been proposed as an alternative to centralized approaches for various applications. As described in Chapter 2, structured P2P systems provide a primitive called the distributed hash table (DHT) for storing and locating objects. Each object is associated with a key that belongs to the same address space as node identifiers. The ownership of keys is partitioned among participating nodes, such that each key is stored on a set of nodes that are closest to the key in the identifier space. This partitioning of key ownership together with efficient routing between nodes bounds lookup overhead for an object stored in the DHT. The following properties make DHTs useful wide-area storage architectures:

1. **Decentralization**: All nodes in a DHT are responsible for storing data and the entire system is self-managing, and hence no external coordination is required. This is particularly important in an environment featuring multiple organizations and individuals.

2. **Scalability**: Every new node that is added increases the storage capacity of the entire system. Besides, each node is only required to maintain a small amount of state about other nodes.

3. **Fault-tolerance**: All nodes are treated equal and hence there are no central points of failures. When a node goes down only a part of the data (that is stored at that node) becomes unavailable. DHTs automatically replicate data on several nodes, and can provide high-availability of data even when participating nodes are volatile.

4.1 Related Work

4.1.1 Systems Based on Distributed Hash Table

There is a rich literature on using structured P2P techniques to build scalable and fault-tolerant systems. Notable among these are large scale storage utilities: CFS [37] based on Chord [40], and PAST [36] developed by Microsoft based on Pastry [39]. In [60] Cox et. al. have proposed to build Distributed DNS using DHash, a distributed hash table (DHT) based on Chord [40]. SCRIBE[86] is a large scale application-level multicast and event notification infrastructure based in Pastry P2P system. ePost [87] describes a decentralized email service, also based on Pastry. OpenDHT [88] is a public DHT service based on Bamboo P2P system [89] operating on PlanetLab, and can be used by third-party applications.

By restricting the key ownerships to a small set of nodes and not requiring caching to achieve efficient lookup, DHTs can also be used to store mutable data. In [90], authors propose an algorithm to provide atomicity of mutable data stored in a DHT. Comet [91] uses DHT to provide a scalable and decentralized coordination space as in Linda [92].

In [93], the authors propose to use a universal overlay to provide a scalable infrastructure to bootstrap multiple service overlays providing different functionality. It provides mechanisms to advertise services and to discover services, contact nodes, and service code. In this work, I demonstrate how a universal overlay can be used to facilitate bootstrapping of multiple WOWs, each supporting a different community and having its own virtual private IP address space.

4.1.2 Distributed Hash Table Design

DHTs operate over structured P2P networks, and routing in these networks is largely oblivious of load and geographical location. These systems impose a well-defined topology

of nodes which are all treated equal, and the arrangement of nodes depends only on their P2P identifiers. No consideration is made on their capabilities, load and connectivity, which tend to differ a lot. This heterogeneity raises the following issues for wide-area DHT deployments:

1. **Latency**: Even though DHTs can bound the number of overlay hops on key lookups, these lookups can be routed through nodes that are too heavily loaded or are geographically scattered, and hence incur a high latency. Besides, a slow storage node also hits latency of a DHT operation.
   Proximity Neighbor Selection (PNS) [27] is a technique in which whenever a P2P node has a choice on its routing table entries, it picks the node that has the least latency to it. It has been shown that this local minima at each hop can bound the total latency of a lookup to within a certain fraction of the actual latency to the node storing the key. PNS is already employed in existing P2P systems (Chord [40],Pastry [39]) and can be incorporated into Brunet to achieve the bounds on the total transit latency incurred by a virtual IP packet.
   It is possible that the messages containing DHT operations get lost and need retry. The Bamboo P2P overlay [89] uses the Internet round-trip latency information to calculate the right timeouts on DHT operations. The P2P routing at each node also tries to avoid a high latency hop. The Bamboo DHT also replicates a key at several nodes, so that a single slow node storing the key does not delay the lookup process.
   In SBARC [94] and Brocade [95], some P2P nodes upgrade to supernodes based on their high resource capacities and high stability. All intermediate P2P hops involve the only supernodes, thus achieving a quick lookup latency. Skype uses a similar approach, except that it is based on an unstructured P2P network.

2. **Routing consistency**: In the absence of churn, a DHT operation on a key must always be routed to same set of nodes, no matter where it originates. This property is called consistency. The arrangement of nodes in a structured P2P network depends only on their P2P identifiers. Two adjacent nodes in the topology should be able to communicate with each other, an assumption that breaks down in wide-area in presence of NATs/Firewalls and Border Gateway Protocol (BGP) outages [26]. A node can potentially miss a neighbor with which it cannot communicate, which can lead to an incorrect routing decision and subsequently a DHT operation to be routed to wrong set of nodes.

3. **Load balancing**: DHTs rely on consistent hashing (SHA1, MD5) of data to generate object keys that are expected to distribute uniformly over the P2P nodes. P2P nodes generally differ in their capabilities and therefore a uniform distribution does not suffice. The following two techniques can used for load balancing: (1) a more capable machine runs multiple P2P nodes which increases the likelihood of a random key getting assigned to it, and (2) active load balancing where a lightly

loaded node (by restarting with a different P2P address) migrates to some region of the overlay that has more keys, thus sharing a part of the storage and lookup load.

4. **Multi-attribute range queries**: DHTs have been designed to provide key-based lookup on objects. This makes them excellent candidates in systems where resources are identified by names (file systems, name resolution etc). However in most resource discovery systems, resource queries are based on one or more attributes, which are often specified as ranges of values. Serving multi-attribute range queries has been an area of recent research [96–98].

5. **Content-path locality**: The lack of control on key placement and routing paths in structured P2P systems raises concerns over autonomy, administrative control and autonomy of participating organizations. In [99], the authors present techniques to provide content/path locality and support for NATs and firewalls, where instances of conventional overlays are configured to form a hierarchy of identifier spaces that reflects administrative boundaries and respects connectivity constraints among networks.

4.2 Distributed Hash Table on Brunet

The Brunet P2P library has been extended with the functionality to support object storage and retrieval, including replication for fault-tolerance. Each DHT key is stored at two P2P nodes which are to its immediate left and right in the key (or address) space. The P2P nodes support migration of keys and their associated values to reflect changes in the ring due to node arrival and departure.

4.2.1 Handling Topology Changes

Figure 4-1A shows how a new node arrival is handled. Initially, node 123 stores keys in the range $[110, 123] \bigcup [123, 128]$, while node 110 stores keys in the ranges $[110, 123] \bigcup [110, 123]$. In response to the arrival of the node 116, node 123 migrates the keys in range $[110, 116]$ to the new node. Similarly, node 110 migrates the keys in range $[116, 123]$ to the new node.

Figure 4-1B shows how a node failure is handled. Initially, node 123 stores keys in the ranges $[116, 123]$ and $[123, 128]$, while node 110 stores keys in the ranges $[110, 116]$ and $[110, 123]$. In response to the failure of node 116, node 123 copies the keys in range $[116, 123]$ to node 110, while node 110 copies the keys in range $[110, 116]$ to node 123.

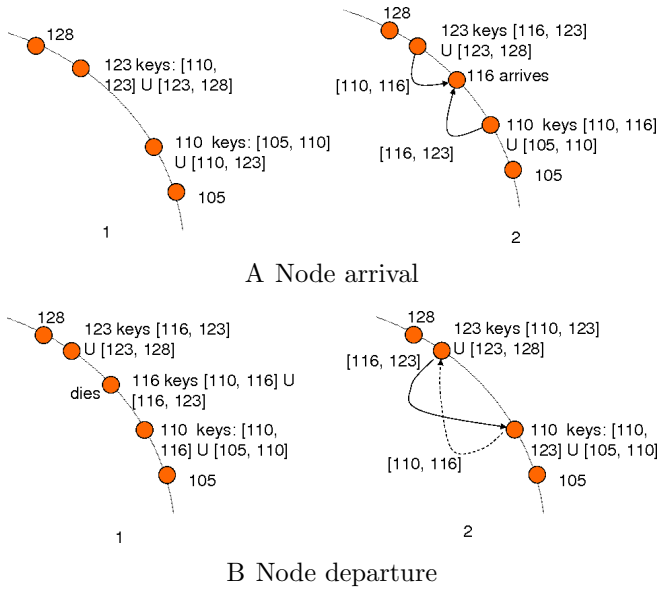


Figure 4-1. Handling topology changes

4.2.2 Application Programmers Interface (API)

The DHT implementation presents the following API to applications:

1. *Create(key k, password p, value v, time-to-live ttl)*: Insert a key-value pair (k, v) into the hash table with password p for *ttl* seconds, only if the key k already does not exist. Returns *true* on success, otherwise returns an *error*. Note that the entry is stored only for *ttl* seconds.

2. *ReCreate(key k, password p, value v, time-to-live ttl)*: Extends the lifetime of entries with key k, value v and password p by *ttl* seconds.

3. *Get(key k)*: Returns all live values (*time-to-live* not expired) associated with key k.

Since the objects are stored in the DHT as soft-state for the lifetime specified in *time-to-live* (after which they are automatically garbage collected), there is no primitive to delete an object associated with a some key[1] . To prevent data loss due

[1] Deletion of keys can be achieved by inserting a deletion record into the DHT with the key k and password p with the time-to-live equal to a maximum value. As long as a matching deletion record exists in the DHT, any operation on the key is ignored.

to garbage collection, applications are thus required to re-insert (using a *ReCreate*) objects periodically into the DHT.

4.2.3 Tolerance to Inconsistent Roots

Structured P2P routing assumes each node has a consistent view of its local neighborhood, which is reflected in its ability to communicate with its left and right neighbors in the P2P identifier space. However, connectivity constraints due to symmetric NATs and Internet route outages [26], often prevent communication between immediate neighbors in the identifier space, thus affecting overlay structure maintenance and routing. This inability to communicate with a neighbor node is perceived as the neighbor being down, thus resulting in an inconsistent view of the local neighborhood and subsequently incorrect routing decisions at that node. In such cases, DHT operations on the same key k originating at different sources may not always routed to the same node (called the *root* for that key). This problem is referred to as inconsistent roots in the DHT literature and hinders the ability of DHT *Create* primitive to detect duplication of keys, as shown in Figure 5-3.

Nodes 115 and 110 cannot form a connection. A message is sent to key 112, and the closest node is 110. Left: message (*Create*) addressed to key 112 arrives at node 115; it believes that it is the closest to the destination – the message is delivered locally and the key is successfully created (also replicated at node 100). Right: another message *Create* addressed to the same key arriving at node 83 is correctly routed to node 110 – here the key is not found and is created again (this *Create* operation also returns *success* instead of returning an *error*).

The inability of the DHT *Create* primitive to detect duplication of keys subsequently affects the correct operation of applications requiring such uniqueness guarantees, such as the decentralized Dynamic Host Configuration Protocol (DHCP) described in Section 4.4.2.

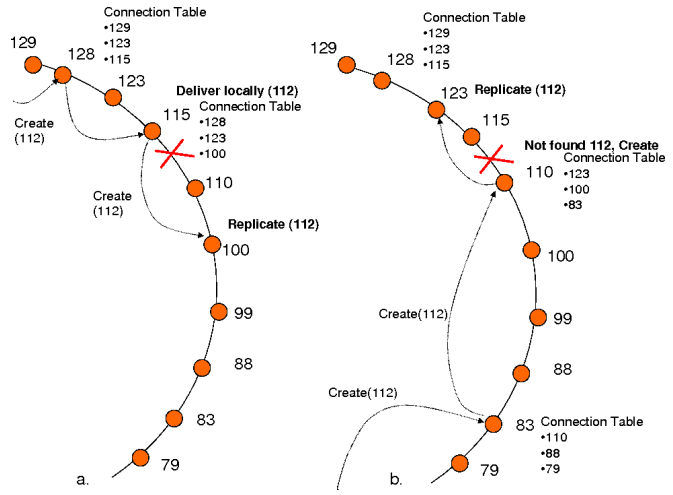


Figure 4-2. Inconsistent roots in DHT

To reduce the likelihood of inconsistent roots, each application specified key k is internally re-mapped to n keys $(k_1, k_2 ... k_n)$, which are then stored (together with the associated value) at n different locations on the P2P ring. Applications can choose this degree of re-mapping for each key, and expect DHT operations to separately provide return values for each re-mapped key, thus allowing applications to implement schemes like majority vote on results obtained for each such re-mapped key. For a fault to occur now, the roots of as many as half (more than one) of the re-mapped keys have to be inconsistent. Majority voting also has the advantage that by not requiring results for all re-mapped keys, a few slow nodes cannot slow down the entire DHT operation.

Furthermore, a new P2P node joining the overlay is not allowed to perform any DHT operation until it gets connected correctly, i.e. it forms connections with its nearest left and right neighbors on the ring. This is because an incorrectly connected node has an inconsistent view of the ring and may observe roots for the DHT keys that are inconsistent with those observed by existing nodes. The time taken for a new node to get correctly connected to an existing network of over 100 nodes has been observed to be about 5 seconds on average.

4.3 Extensions to IPOP

The IPOP prototype has also been extended to support dynamic creation and discovery of virtual IP to P2P address mappings. These mappings can be arbitrary (many-to-one), thus allowing a single P2P node to route for multiple VMs on a host. These mappings are stored as objects in the DHT. Another extension to IPOP is the support for different virtual private networks (each with its own address space) on top of a common P2P overlay. Each such private network is called an IPOP namespace. All nodes within a WOW node belong to the same namespace, and cannot communicate with nodes belonging to other WOWs (or namespaces).

Figure 4-3 shows how different WOWs (or IPOP namespaces) can exist on top of a common P2P overlay. Each virtual IP node belongs to some IPOP namespace and is associated with a P2P node. In this example, the $IP \rightarrow P2P$ mappings for nodes $A1, B1, A2, B2$ ($A1 \rightarrow X8$, $B1 \rightarrow X1$, $A2 \rightarrow X2$ and $B2 \rightarrow X4$) are stored at nodes X3, X5, X6 and X7 respectively. The DHT key for each such mapping is a combination of a globally unique identifier for the namespace and the virtual IP address within that namespace. The inclusion of the namespace identifer allows virtual IP nodes in different namespaces to have same IP addresses. To send a virtual IP packet to node B1, the node A1 queries the DHT with $(N1, B1)$ as key. The value associated with this key is the P2P address (X1) of the P2P node associated with B1, and is quickly retrieved from node X5. From this point onwards, communication proceeds as described in Chapter 2.

Creating a new IPOP namespace only requires executing a simple program with information about the IPOP namespace (assignable virtual IP addresses and other network parameters). The namespace identifier is then provided as a parameter inside the IPOP configuration of the appliance VMs for distribution. Experiments show that a new node joining a WOW takes about 20-30 seconds on average to acquire a virtual IP address.

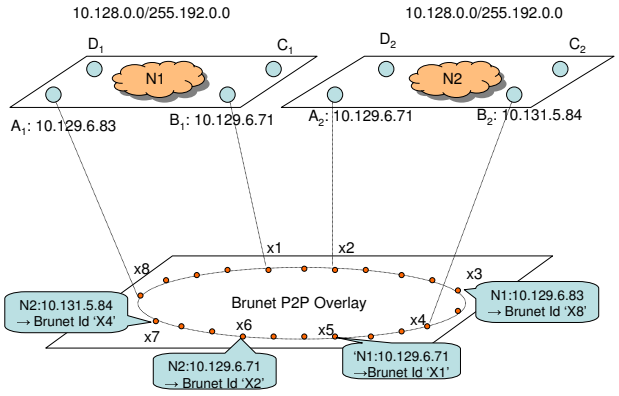


Figure 4-3. Example of two different WOWs with namespaces N1 and N2 sharing a common Brunet overlay

Similar to DHT-based systems described earlier, this decentralized IP address management (1) eliminates any dedicated components, (2) scales to large numbers by harnessing the resources at participating nodes and (3) provides resilience to node failures, which are common requirements in large-scale desktop grid environments.

4.4 Lifecycle Management of WOWs

Based on the discussion in previous sections, I now describe techniques that facilitate the deployment and management of WOWs by enabling 1) nodes of a WOW to dynamically obtain IP addresses using existing DHCP clients but without relying on centralized DHCP servers, and 2) different WOWs to multiplex a single overlay network while independently managing their own virtual IP address spaces.

The described functionality is comparable with tasks an administrator would typically perform to setup a private network. Following the setting up of switches and cables, a private IP address range is set aside. Hosts connecting to the private network are assigned unique IP addresses from this IP address range. To enable dynamic network configuration of hosts connecting to the network, one or more DHCP servers are configured with the list of assignable IP addresses and other network parameters. New hosts discover

DHCP servers through LAN broadcasts, acquire leases on IP addresses, which they renew periodically. For exchange of packets between hosts, their IP addresses must be resolved to their hardware addresses through Address Resolution Protocol (ARP). The network switches are automatically configured for routing packets through their ports.

In contrast, to setup a new WOW, a user is only required to create an IPOP namespace with a unique identifier and a private address space. The namespace identifier is specified inside the IPOP configuration of the WOW VM appliance image. Each deployed instance of the appliance on boot retrieves the namespace information (virtual IP address range) and configures itself with a unique virtual IP address. These steps are described below.

4.4.1 Creating an IPOP namespace

Creating an IPOP namespace is a simple procedure: executing a simple program and providing information about the namespace (assignable IP addresses, netmask and lease times). This program is already initialized with Uniform Resource Indicators (URIs, [85]) of nodes in the universal overlay and starts up a P2P node that connects to that overlay. The node tries to insert the namespace information into the DHT (using *Create*) with a randomly chosen identifier as the key. If the key already exists, the Create returns an error and the program retries with a different identifier until it succeeds. Since the DHT does not store objects indefinitely, this object holding the namespace information has to be periodically recreated (using Recreate). This namespace identifier is specified inside the IPOP configuration of the WOW appliance image.

4.4.2 Dynamic Host Configuration

In [71], IPOP supports dynamic virtual IP configuration using unmodified DHCP clients. This is achieved by capturing DHCP request packets from the *tap* and making SOAP requests to a publicly accessible server that stores the list of assignable IP addresses and active leases, and eventually injecting DHCP response packets to the *tap*. The SOAP server can be a single point of failure. The decentralized DHCP uses the DHT as

distributed database for storing all information the SOAP server would otherwise keep, assignable IP addresses and active leases on IP addresses.

On intercepting a DHCP packet, IPOP retrieves information about its namespace (assignable IP addresses range, netmask, lease times) from the DHT (using a *Get*) on the namespace identifer as the key. It then chooses a random IP address from that range, belonging to the namespace, and attempts to create a DHT entry (using a *Create*) with: combination of namespace identifier and the guessed IP address as the key, a randomly chosen password, and its P2P address as the value. The entry is successfully created only if there is no other entry with the same key. This prevents IP address conflicts between WOW nodes belonging to the same namespace. In case *Create* returns an error, IPOP tries another (randomly chosen) IP address until it eventually succeeds. The DHCP response packet with information about the lease is written to *tap*. The password is recorded for subsequent operations on the key.

The entry is only created with a time-to-live (TTL) equal to the lease time for that namespace, and thus needs to be recreated (using a *ReCreate*) periodically. This process is again triggered by the DHCP client, which attempts to renew a virtual IP lease after half the lease time has elapsed. In this case, IPOP attempts to *ReCreate* the same DHT key corresponding to the virtual IP address bound to *tap*.

DHT inconsistencies as described in Section 4.2.3, can create a situation where multiple IPOP nodes acquire the same virtual IP address. The DHT implementation in IPOP therefore achieves fault-tolerance to inconsistencies by re-mapping each application specified key k to 8 different keys $(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8)$ and perform the corresponding operation on each of these keys. To consider a *Create* or *Recreate* to have successfully happened, it is expected that at least 5 of these operations be successful (return a *true*). Otherwise, the operation is considered to have failed and a different IP address is tried. However, the initial implementation of this technique that is evaluated in Section 4.5 still waits for all 8 results before performing a majority vote – in cases where

not all 8 results arrive at the source node (due to packet losses) the operation is considered to have failed.

Figure 4-4 shows a time-line of events, from the startup of the IPOP and DHCP client (*dhclient*) process, to having an IP address bound to *tap*. It should be noted that the IP address lease acquisition cannot start until the associated P2P node is correctly connected (i.e., with left/right neighbors). Once correctly connected, IPOP tries different virtual IP addresses (one-by-one) until it is assured that no other node within the same IPOP namespace has acquired the same IP address. It is possible to have a large IP address range for the private network, which reduces the chances of two IPOP nodes guessing the same IP address.

4.4.3 Resolution of Virtual IP to P2P Address

Whenever an IPOP node has a virtual IP packet to send out, it must first must learn the P2P address of the IPOP node associated with the destination IP address. This mapping is created in the DHT when the destination node acquires its virtual IP address. It can be retrieved by the source IPOP node (using *Get*) in less than a second and then cached locally. During this short period (less than a second), a few packets to that virtual IP address are dropped at the source IPOP node. As shown in Chapter 3, most TCP/IP based applications communicating over the IPOP virtual network are resilient to such transient packet losses. This process of resolving a virtual IP address to a P2P address is called "Brunet-ARP".

4.5 Experiments

This section presents an experimental evaluation of the decentralized technique for virtual IP address configuration, with respect to the delay incurred from IPOP and DHCP client startup to the point that an IP address is bound to the *tap*. A bootstrap overlay network of over 100 P2P nodes is setup on PlanetLab and an IPOP namespace is created. The namespace consists of over 650,000 assignable IP addresses and has a lease time of 12 hours.

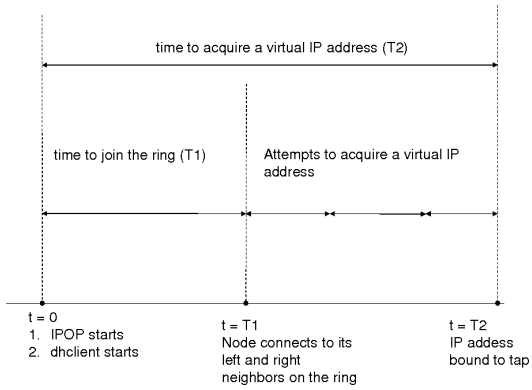


Figure 4-4. Events leading to virtual IP address configuration of tap interface

An experiment between two desktop machines A and B is then performed as follows. On desktop A, start IPOP and DHCP client so that it acquires a virtual IP address, which remains fixed during the experiment. On the desktop B, proceed an iterative process of: (1) start IPOP node and DHCP client (2) wait until an IP address is bound to *tap* (3) start pinging the virtual IP address of desktop A for 200 seconds (4) kill IPOP and DHCP client on B. This process was repeated 250 times.

In each trial of the experiment, the IPOP node on desktop B had a different (randomly chosen) P2P address. The virtual IP leases from different trials persisted in the DHT (since the leases are not relinquished), and this prevented the desktop B from acquiring the same IP address over different trials. This mapping between virtual IP address and P2P address is arbitrary and is discovered automatically in every trial (as described in Section 4.4.3) before ping packets start flowing between the desktops over the virtual network.

Figure 4-5 shows the cumulative distribution of delay seen by the DHCP client (*dhclient*) to acquire an IP address on the *tap*. We observe that in 90% of the cases, DHCP process finishes within 30 seconds of IPOP and DHCP client startup. As shown in Figure 4-4, this delay depends on (1) time taken for the IPOP nodes to get correctly connected and (2) number of different IP addresses that are tried. The cumulative distributions of these components are shown in Figure 4-6 and Figure 4-7.

In most cases, DHCP succeeds to acquire the first IP address it tries. However, despite picking up a large IP address range, samples were observed where more than one IP addresses were tried. This observation is explained as follows. Due to P2P message losses, the IPOP node doing DHCP did not get sufficient results to consider a *Create* or *Recreate* successful. The DHT implementation conservatively assumed a failure and tried a different (randomly chosen) IP address until it eventually succeeded.

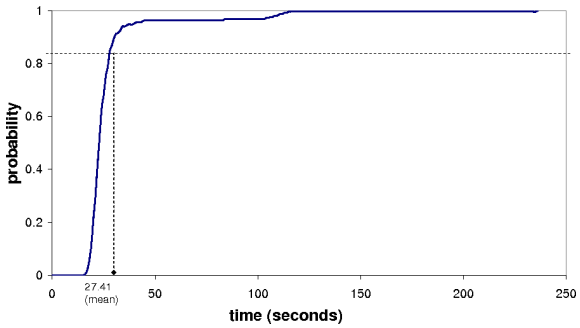


Figure 4-5. Cumulative distribution of the time taken by a new IPOP node to acquire a virtual IP address (T2 in Figure 4-4). Mean and variance are 27.41 seconds and 20.19 seconds, respectively.

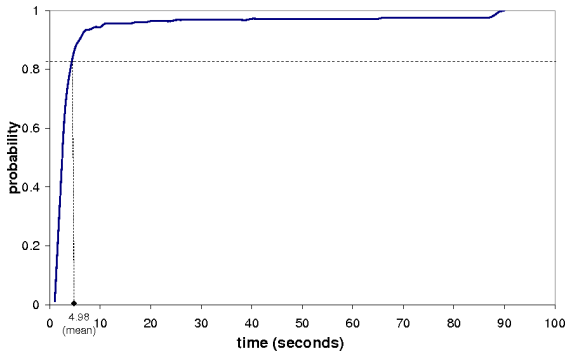


Figure 4-6. Cumulative distribution of the time taken by a new P2P node to get connected to its left and right neighbors in the ring (T1 in Figure 4-4). Mean and variance are 4.98 seconds and 13.99 seconds, respectively.

4.6 Conclusion

In this Chapter, I have described techniques to facilitate deployment of isolated WOWs by individual users without requiring any bootstrapping infrastructure or centralized components. I presented a decentralized DHCP protocol for virtual IP address

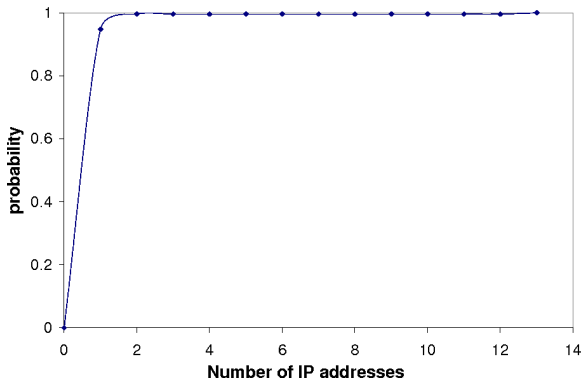


Figure 4-7. Cumulative distribution of the number of different virtual IP addresses tried during DHCP. Mean and variance are 1.096 and 0.784, respectively.

management within a WOW that leverages the DHT functionality of the P2P network. Experiments have shown that a new WOW node can acquire a unique virtual IP address within 20-30 seconds on average. The implementation of this protocol has now been made resilient to P2P message losses. Instead of waiting for each internally re-mapped key operation (*Create* or *Recreate*) to return a result, the operation is considered successful and the protocol proceeds forwards as soon as sufficient results are available to assure that the operation has succeeded on majority of nodes. These enhancements have resulted in reduction in the average time to acquire a virtual IP address to less than 10 seconds.

Additional decentralized configuration can be integrated into WOWs. In particular, the WOW-based Condor pools that are currently configured from a central server, can be extended to leverage DHT-based techniques to achieve manager discovery for worker nodes.

CHAPTER 5
IMPACT OF WIDE-AREA CONNECTIVITY CONSTRAINTS ON IPOP

Like several related efforts (such as Chord [40], Kademlia [100] and Pastry [39]), IPOP relies on structured P2P overlays to provide the core service of message routing and additional capabilities such as object storage and retrieval. Structured P2P routing assumes each node has a consistent view of its local neighborhood in the P2P identifier space, which is reflected in its ability to communicate with its neighboring nodes. Related work on structured P2P systems have implicitly assumed an environment where P2P nodes are able to establish direct connections to one another, and have mainly focussed on efficient overlay topologies [27], correct routing of object lookups under churn [89, 101], and proximity-aware routing [102].

However, in practice, wide-area environments are becoming increasingly constrained in terms of peer connectivity, primarily due to the proliferation of NAT and firewall routers. Studies have shown that about 30%-40% [103] of the nodes in a P2P system are behind NATs. Even though majority of the NAT devices (cone NATs) can be "traversed" through UDP-hole punching, up to 20% of the NATs [46] (symmetric NATs) cannot be traversed using the existing techniques. In addition, studies have also shown the existence of permanent or transient route outages between pairs of nodes on the Internet; for example, [26] reports 5.2% pair-wise outages among nodes on PlanetLab [29]. Together, these connectivity constraints pose a challenge to overlay structure maintenance: two adjacent nodes cannot communicate directly, creating false perceptions of a neighbor not being available. In general, these missing links on a P2P structure lead to inconsistent routing decisions, and subsequently affecting routability and services built upon the assumption of consistent routing.

The existing implementations of structured P2P systems have recognized the problem of overlay structure maintenance when only a small fraction of pairs (about 4% [104]) cannot communicate with each other [26, 105]. However, practical experiences with WOW

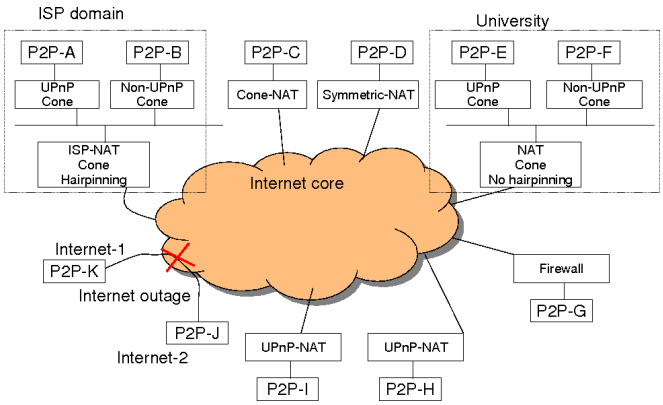


Figure 5-1. Connectivity-constrained wide-area deployment scenario targeted by deployments of the IPOP P2P system

deployments reveal that this fraction can be significantly larger due to nodes behind (multiple) NATs, and NATs that are symmetric or do not support "hairpin" translation that preclude hole-punching.

5.1 Connectivity Hazards in Wide-Area Networks

Several structured P2P systems have been deployed on wide-area infrastructures when participating hosts are on the public Internet. For example, OpenDHT [88] relies on non-firewalled PlanetLab P2P nodes to deploy its DHT; however, nodes behind NATs and firewalls can only act as OpenDHT clients and do not store keys. In order to aggregate the increasing number of hosts behind NATs/firewalls as WOW nodes, the IPOP virtual network must be able to deal with a complex wide-area environment as the one depicted in Figure 5-1, where typical end users of a P2P system are constrained by NATs in which they do not have the control (or expertise) necessary to set up and maintain firewall exceptions and mappings necessary for NAT traversal.

It is common for broad-band hosts to be behind two levels of NAT (a home gateway/router and the ISP edge NAT, nodes A and B in Figure 5-1). IPOP supports establishment of UDP communication using hole punching techniques for "cone" type NATs (e.g. between nodes A and C in Figure 5-1), and there is empirical evidence pointing to the fact that these are the common case [46]. However, nodes behind NATs for

which hole-punching does not prove effective cannot communicate with nodes in different private networks, and only communicate with public nodes or full-cone NATs (e.g. nodes C and D).

Recognizing the importance of supporting traversal, some of recent NATs have started supporting Universal Plug and Play (UPnP)[106] which allow them to be configured to open ports so that other hosts (outside the NAT) can initiate communication with hosts behind the NAT (e.g. hosts I and H). However, UPnP is not ubiquitous, and even when it is available, multi-level NATs create the problem that hosts can only configure their local NATs through UPnP, while having no access to control the behavior of the edge NAT, which renders the UPnP approach ineffective outside the domain. For example, although hosts A and E in Figure 5-1 are connected to UPnP NATs, they are also subject to rules from an ISP NAT and a University NAT respectively, which they do not control.

Some NATs support "hairpinning", where two nodes in the same private network and behind the same NAT can communicate using each other's translated IP and port. Such a behavior is useful in a multi-level NAT scenario, where two hosts behind the same public NAT but different semi-public NATs are able to communicate only using their IP address and port assigned by the public NAT. However, not all NATs support hairpinning, creating a situation in which two nodes in the same multi-level NATed domain may not be able to use hole-punching to communicate directly (e.g. nodes E and F) as depicted in Figure 5-2. Nodes E and F are behind two different semi-public NATs respectively, which in turn are behind a public NAT-P. While forming its initial connections with bootstrap nodes on PlanetLab, these nodes only learn their IP endpoints assigned by the public NAT-P. When E and F try to form a connection they send link messages to each others IP endpoints assigned by NAT-P. In case NAT-P does not support "hairpinning", E and F cannot form a connection. Only 24% of the NATs tested in [46] support hairpinning.

Some hosts are behind firewall routers (e.g. host G) that might block all UDP traffic altogether. Only a few P2P nodes are public and are expected to be able to

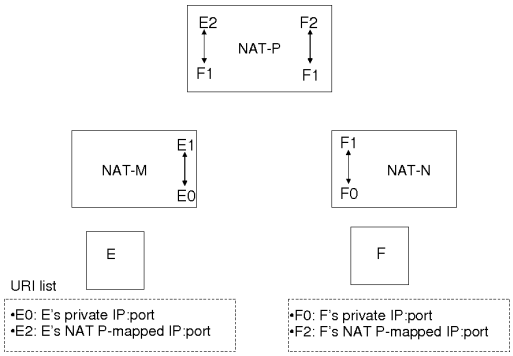


Figure 5-2. Multiple levels of NATs

communicate with each other. Connectivity even among these hosts is constrained: Internet-1 and Internet-2 hosts cannot communicate with each other (e.g. hosts J and K), while multi-homed hosts can communicate with them both. In addition, link failures, BGP routing updates, and ISP peering disputes can easily create situations where two public nodes cannot communicate directly with each other. In [26], the authors observed that about 5.2% of unordered pairs of hosts (P1,P2) on PlanetLab exhibited a behavior such that P1 and P2 cannot reach each other but another host P3 can reach both P1 and P2.

Its is observed that a typical wide-area environment presents several deterrents to connectivity between a pair of nodes, and when two such nodes have adjacent identifiers on the P2P ring, structure maintenance is affected. To the best of my knowledge, while structured P2P systems have been demonstrated in public infrastructures such as PlanetLab, where there are only a few pair-wise outages and a small amount of disorder can be tolerated [105], no structured P2P systems described in the literature have been demonstrated where the majority of P2P nodes are subject to NAT constraints of various kinds as illustrated in Figure 5-1.

5.2 Impact of Connectivity Constraints

Peer connectivity constraints result in the inability to correctly maintain an overlay structure, which in turn affects the deployment of virtual networks and WOWs in important ways. These are presented and discussed in the remaining of this section.

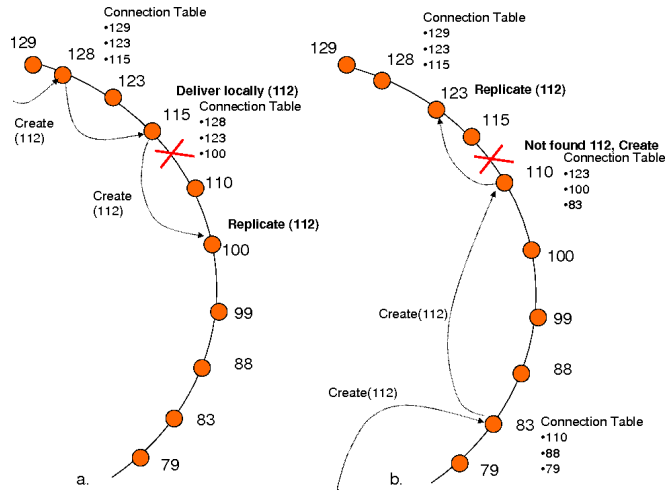


Figure 5-3. Inconsistent roots in DHT

5.2.1 Impact on Core Structured Overlay Routing

The Brunet system implements a ring-structured P2P network where each node has a randomly generated 160-bit identifier. Each node maintains $2m$ *structured near* connections, m connections on each side of the P2P ring. In addition to the neighbor connections, each node also acquires k *structured far* connections that are far away in the address space, so that the average number of overlay hops between nodes is $O(\frac{1}{k}\log^2(n))$ for a network of size n using the algorithm of [45].

Similar to other structured systems, routing in Brunet uses the greedy algorithm where at each hop a message gets monotonically closer to the destination until it is either delivered to the destination or to the node that is closest to the destination in the P2P identifier space. Greedy routing assumes each node has a consistent view of its local neighborhood, which is reflected in its ability to form *structured near* connections with its left and right neighbors in the P2P identifier space. The inability to form connections with immediate neighbors in the identifier space creates inconsistent view of the local neighborhood, thus resulting in incorrect routing decisions as shown in Figure 5-3 (a).

5.2.2 Effect on All-to-All Connectivity

Techniques for the creation of overlay links between P2P nodes behind "cone" NATs have been presented in earlier work, which incorporates decentralized NAT traversal using UDP hole-punching [85]. The notion of a *connection*, which describes an overlay link between two P2P nodes, is key to establishing such links. Connections operate over physical channels called *edges*, which in IPOP can be based on different transports such as UDP or TCP. Besides assisting in overlay structure maintenance, the connection protocols allow the creation of 1-hop shortcuts between WOW nodes to self-optimize the performance of the virtual network with respect to latency and bandwidth.

The connection setup between P2P nodes is preceded by a connection protocol for conveying the intent to connect and exchanging the list of Uniform Resource Indicators (URIs) for communication. These connection messages are routed over the P2P network. Incorrect routing leads to situations where connection messages are either misdelivered (or not delivered at all), thus affecting both overlay structure maintenance and connectivity within the virtual network.

5.2.3 Effect on Dynamic Virtual IP Configuration

The Brunet P2P system also provides decentralized object storage and retrieval based on a DHT [107], which is used for dynamic virtual IP configuration of WOW nodes, summarized as follows. IPOP supports creation of multiple mutually-isolated virtual networks (called IPOP namespaces) over a common P2P overlay. The virtual IP configuration of WOW nodes in each such private network is achieved using a decentralized implementation of the Dynamic Host Configuration Protocol (DHCP). The DHCP implementation uses a DHT primitive (called *Create*) to create key/value pairs mapping virtual network namespaces and virtual IP addresses uniquely to P2P identifiers. The *Create* primitive relies on the consistency of key-based routing to guarantee uniqueness of IP-to-P2P address mappings. That is, messages addressed to some key k must be delivered to the same set of nodes regardless of its originator.

Incorrect routing decisions as shown in Figure 5-3 (a) and (b), can cause *Create* messages addressed to the same key from different sources to be routed to different nodes. This problem is also identified in [26] and is referred to as inconsistent roots, and can lead to a situation where two WOW nodes claim the same virtual IP address.

5.2.4 Effect on Completion of DHT Operations

To reduce the impact of inconsistent roots, the Brunet-DHT internally re-maps each application specified key k to n keys $(k_1, k_2 ... k_n)$, which are then stored (together with the associated value) at n different locations on the P2P ring and the DHT operations are expected to separately provide return values for each re-mapped key. Majority voting on results obtained for each such re-mapped key is used to determine the outcome of an operation. For a fault to occur in this scenario, the roots of as many as half of the re-mapped keys have to be inconsistent. However, majority voting can reach a consensus only when results from at least half of the n re-mapped are communicated back to the source node. If the nodes close to the source node in identifier space have inconsistent view of their neighborhoods, situations can arise when not enough results arrive at the source node for consensus, causing the operation to fail. This inability to complete a DHT operation impacts both the process of acquiring a virtual IP address, and also resolution of a virtual IP address to P2P identifier.

5.2.5 Effect on DHT Dynamics

The inability to create overlay links also hinders the dynamics of a DHT as it reacts to changes in ownership of keys when nodes join, and actively replicates keys when nodes leave. Until a new node can form a consistent view of its local neighborhood by communicating with its neighbors, it can neither retrieve any keys (that it is supposed to store) from its neighbors, nor copy (or migrate) some keys that are now supposed to be stored at its neighbors. This affects the degree of replication of keys in the DHT, and subsequent reliability of object storage.

5.2.6 Effect on Topology Adaptation

To provide better latency and bandwidth, the IPOP virtual network supports creation of 1-hop communication in response to virtual IP communication between nodes. In the absence of direct communication, the virtual IP traffic is routed through one or more P2P nodes. In such cases, the performance of the virtual network is limited by the capabilities and geographical locations of intermediate nodes in the routing path; the associated latency and bandwidth overhead may be too high for certain interactive and large data-transfer applications, respectively.

5.3 Discussion

To summarize, incorrect routing in the P2P network impacts the virtual IP connectivity between WOW nodes, which directly affects the applications using the virtual network. For example, in a WOW-based Condor pool [71], the inability of a worker node to obtain an IP address implies it does not join the pool. Even if a node "N" obtains an IP address, if it cannot communicate with the central manager node "M", it is not available for computation. Furthermore, the inability of node "N' to route to a worker node "W" prevents jobs submitted by "N" to execute on "W". All these situations result in the system not being able to achievable the maximum available throughput because not all nodes can participate in computations.

Chapter 6 describes generally applicable techniques that facilitate consistent structured routing despite the connectivity constrained presented by a wide-area environment. Then in Chapter 7, I evaluate the applicability of Proximity Neighbor Selection (PNS) in conjunction with network coordinates to reduce the latency of key lookups in IPOP overlays on PlanetLab. Techniques are also described for selecting suitable proxy nodes to route communication between virtual IP nodes, when topology adaptation based on *shortcuts* is not possible.

CHAPTER 6
IMPROVING PEER-CONNECTIVITY

In this chapter, I describe and evaluate two novel, synergistic techniques for fault-tolerant routing and structured overlay maintenance in the presence of network outages: *annealing routing*, an algorithm based on simulated annealing from optimization theory, and *tunnel edges*, a technique to establish connections between P2P nodes by tunneling over common neighbors. These are fully decentralized and self-configuring techniques that have been successfully implemented in the Brunet P2P system and demonstrated in actual wide-area PlanetLab deployments as well as in NATed environments with emulated pair-wise outages. The effectiveness of these approaches are analyzed for various system configurations with the aid of analytical models, simulation, and data collected from realistic system deployments.

6.1 Annealing Routing

The first technique is a fault-tolerant routing algorithm based on simulated annealing that, unlike conventional greedy routing, does not force a message to monotonically get closer to the destination at each hop. This algorithm is inspired by optimization theory. Under the assumption of a convex function, a greedy method converges to a global minimum (or maximum); however, with non-convex functions, a greedy approach can stop at local minima and not find the global minimum. In optimization theory, a simulated annealing approach allows for a deviation from a greedy search in order to "escape" from a local minimum. In the context of P2P routing, connectivity constraints create analogous situations where a greedy algorithm can reach a local minimum when a node is not able to establish a *near* link which would allow the distance between the message and its destination to be reduced. Even if the underlying network is free from connectivity constraints, transient churn can also create situations where a node has an inconsistent view of its local neighborhood. For successful operation of connection setup protocols for overlay structure maintenance, structured routing has to be designed to be tolerant to

such disorder on the P2P ring. The annealing algorithm is described in Algorithm 6-1 and works as follows.

In lines 10-17, the node looks up its connection table to determine if it is adjacent to the destination in the identifier space. In that case, the node delivers the message locally and also sends it to the node on the other side (left or right) of the destination in the identifier space.[1] Otherwise (line 19), the node finds the two closest nodes to the destination from the connection table, u or u_{sec}.

If the message has not taken any hops yet (i.e. it originated at the current node), it is sent to the closest node u. Otherwise (lines 23-29), until the message has taken MAX_UPHILL hops it is delivered to the closest node u or the next closest u_{sec} (if it was already received from the closest node u). Until this point, the algorithm does not check for the forward progress of the message towards the destination in identifier space.

Beyond MAX_HILL hops (lines 30-41), the message is sent to u or u_{sec}, only if the next hop is closer to the destination than the previous hop. It should be noted that this condition only requires progress with respect to the previous node; it still allows a message take one hop that is farther away from destination than the current node.

The annealing algorithm is very useful for routing messages addressed to exact destinations, which include connection setup messages between P2P nodes, virtual IP packets between IPOP nodes, and the results of DHT operations back to the source node. In a perfectly-formed structured ring, the algorithm works exactly as the greedy algorithm and incurs the same number of hops.

When messages are addressed to DHT keys, this algorithm has a better chance to reach the node closest to the key, by delivering the message at each local minima. As a

[1] The state of the local connection table may not correctly reflect the local neighborhood. Greedy routing may terminate the progress of the message here; by also sending the message to the node on the other side, the annealing algorithm continues its search for the node closest to the destination.

Algorithm 1 *AnnealingNextHop*$(v, prev, dest, p)$ This algorithm describes how a packet p arriving at v from *prev* takes its next hop towards the destination *dest* using annealing mode.

```
1: if v == dest then
2:    Deliver locally.
3:    Return.
4: end if
5: if v has a connection to dest then
6:    Send to dest.
7:    Return.
8: end if
9: /** Case 1: Connection table indicates that current
   node v is adjacent to dest. Deliver locally and send
   to the node on the other side of dest, according to the
   connection table.**/
10: if v is adjacent to dest then
11:    if (v is to the left of dest) then
12:       Deliver locally.
13:       Send to v' which is on the right of dest.
14:    else if (v is to the right of dest) then
15:       Deliver locally.
16:       Send to v' which is on the left of dest.
17:    end if
18: else
19:    Find first and second closest nodes u_min and u_sec to
       dest, respectively.
20:    if p.Hops == 0 then
21:       /** Case 2: This is the first hop. Let the packet
          go to closest even if v itself is closest.**/
22:       Send to u_min.
23:    else if p.Hops ≤ MAX_UPHILL then
24:       /** Case 3: Not the first hop. We will do this for
          up to MAX_UPHILL (= 1) hops. **/
25:       if prev ≠ u_min then
26:          Send to u_min.
27:       else
28:          Send to u_sec.
29:       end if
30:    else
31:       /** Case 4: Send only if can get closer than pre-
          vious node.
32:       if prev ≠ u_min then
33:          w ⇐ u_min
34:       else
35:          w ⇐ u_sec
36:       end if
37:       d_min ⇐ DIST_ring(w, dest)
38:       if d_min < DIST_ring(prev, dest) then
39:          Send to w.
40:       end if
41:    end if
42: end if
```

Figure 6-1. Annealing routing algorithm

side effect, DHT operations for a key are performed at more that one node in the P2P overlay. This redundancy is useful for applications using only *Put* and *Get* DHT interfaces, which do not require uniqueness of key values. However, annealing is not sufficient for scenarios including the decentralized DHCP protocol of IPOP, where it is required to guarantee uniqueness of creation of a key to avoid IP address collisions. Ensuring that each key is delivered to exactly one node (closest to the key in the identifier space) is possible by using greedy routing on a completely formed overlay. The next technique I present is designed to provide a complete overlay structure in face of connectivity constraints that may prevent direct connections among overlay neighbors.

6.2 Tunnel Edges

This section describes the second novel technique – it allows an overlay link between two nodes A and B, which cannot communicate directly over TCP or UDP, to be proxied by a set nodes to which both A and B can communicate. It is a fully decentralized technique for both discovering a proxy node C, and establishing an edge "tunnel" connecting A and B through C.

The idea behind tunnel edges is as follows. Assume that each node in the network attempts to acquire connections to its closest $2m$ *near* neighbors on the P2P ring, m such neighbors at each side. Consider a situation where there is an outage between two adjacent nodes A and B on the P2P ring. Since both A and B also attempt to form *near* connections with $2m$ nodes each, their neighborhoods intersect at $2(m-1)$ nodes as shown in Figure 6-2. For a tunnel edge to exist between A and B, there must be at least one node C in the intersection I to which both A and B are connected. Such node C is a candidate to be used in tunneling the structured near connection between A and B.

This enhancement allows the connection state at a node to consistently reflect the overlay topology even when it is not possible to communicate with some neighbors using the conventional TCP or UDP transports. Tunnel edges are implemented (described in Section 6.4) such that they are functionally equivalent to UDP or TCP edges once they

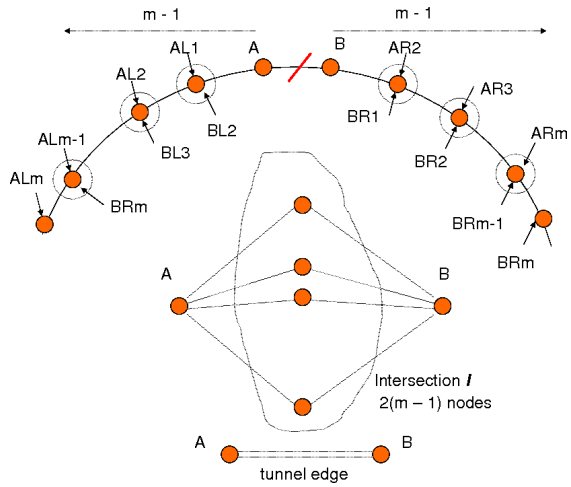


Figure 6-2. Tunnel edge between nodes A and B which cannot communicate over TCP or UDP transports

are established, allowing seamless reuse of the code responsible for state maintenance and routing logic in the system.

Two important questions arise in the context of this proposed approach: what is the probability of tunnel edges to be formed between two nodes A and B? how many nodes are candidates for proxying tunnel edges? These questions are addressed analytically in this section.

Let p be the probability of an edge outage between a pair of nodes. For a tunnel edge to exist between A and B, there must be at least one node C in the intersection I to which both A and B are connected. Assuming m *near* connections at each side of both nodes, the probability for a tunnel edge between A and B to exist through C is given by

$$\begin{aligned} & P[A \textit{ and } B \textit{ can connect}] \\ & = 1 - P[A \textit{ and } B \textit{ cannot connect}] \\ & = 1 - \prod_{C \in I} P[A \textit{ and } B \textit{ cannot connect through } C] \\ & = 1 - (1 - p^2)^{2(m-1)} \end{aligned}$$

Table 6-1 shows the probability of forming a tunnel edge between unconnected nodes A and B for different values of edge probability p and number of *near* connections m. It should be noted that there is a sharp increase in the probability of being able to form a tunnel edge when nodes acquire more than 2 *near* connections on each side. This fact is also reflected in simulation results which show that improvements in correctness of routing using tunnel edges are significantly higher when $m \geq 3$. Figure 6-5 shows 3.9% broken pairs when $m = 2$ and 0.86% broken pairs when $m = 3$.

Now consider a situation where a tunnel edge involves exactly one forwarding node. When the forwarding node departs, the current node also loses the tunnel edge connection. Therefore, for fault-tolerance, it is also important that the forwarding set of nodes for tunnel edge contains more than one node. The probability that forwarding set consists of at least 2 nodes is given by

$$\begin{aligned} & P[forwarding\ set\ of\ size\ atleast\ 2] \\ &= 1 - \sum_{k=0}^{k=1} P[forwarding\ set\ of\ size\ exactly\ k] \\ &= 1 - \sum_{k=0}^{k=1} \binom{2(m-1)}{k} \cdot (p^2)^k \cdot (1-p^2)^{2(m-1)-k} \end{aligned}$$

and the expected size of the forwarding set is given by

$$\begin{aligned} & E[expected\ size\ of\ forwarding\ set] \\ &= \sum_{k=0}^{k=2(m-1)} k \cdot P[forwarding\ set\ of\ size\ exactly\ k] \\ &= \sum_{k=0}^{k=2(m-1)} k \cdot \binom{2(m-1)}{k} \cdot (p^2)^k \cdot (1-p^2)^{2(m-1)-k} \\ &= 2(m-1) \cdot p^2 \end{aligned}$$

For $m = 3$, and $p = 0.9$, the expected size of the forwarding set for a tunnel edge is 3.24, while the probability of having a forwarding set of at least 2 nodes is 0.976.

Table 6-1. Probability of being able to form a tunnel edge as a function of edge probability and number of required *near* connections on each side

	tunnel edge probability			
edge prob	m = 2	m = 3	m = 4	m = 5
0.70	0.7399	0.9323	0.9824	0.9954
0.75	0.8085	0.9633	0.9929	0.9986
0.80	0.8704	0.9832	0.9978	0.9997
0.90	0.9638	0.9986	0.9999	0.9999

It can further be shown that if each node maintains $O(\log_2 N)$ neighbors, then the probability of not being able to form a tunnel edge is:

$$= (1-p^2)^{2(m-1)} = (1-p^2)^{O(m)} = (1-p^2)^{O(\log_2 N)}$$
$$= O(N^{\log_2 (1-p^2)})$$

For $p = 0.9$, the above expression evaluates to: $O(N^{-2.39})$. Therefore, as the network grows in size and nodes tend to acquire more *near* connections, tunnel edges become more and more probable.

6.3 Improvements in Structured Routing

In this section, I present an evaluation of the improvements in structured routing due to annealing routing and tunnel edges with respect to: (1) the all-to-all routability of the P2P network, and (2) consistent routing of keys. The analysis is conducted by simulating structured routing on randomly generated static graphs that model the Brunet overlay, for varying edge probabilities between pairs of nodes.

Scenarios such as symmetric NATs, multi-level NATs and Internet route outages result in complex models for the likelihood of two nodes being able to communicate. For example, the likelihood of a node behind a symmetric NAT being able to form an edge with another arbitrary node depends on the fraction of nodes that are public (or behind full-cone NATs). In the multi-level NAT scenario (Figure 5-2) where the outermost NAT-P does not support "'hairpinning"', the likelihood of a node E to form an edge with another arbitrary node is a function of the fraction of nodes that are behind the same NAT-P, but

in a different semi-private network. An Internet route outage between two sites A and B results in the inability of any node in A to communicate with any node in B.

In the absence of any published work that provides a fault model to capture all such scenarios, the likelihood of an edge between a pair of nodes has been modeled with a uniform pair-wise edge probability, and allowing for high probabilities of P2P edges not being able to form – as high as 30%.

6.3.1 Simulation Methodology

The simulation environment captures the algorithms used in Brunet for structured overlay creation and routing, and models pair-wise outages with configurable probability p. In each simulation, a network of 1000 nodes with randomly generated 160-bit identifiers is created. Based on the probability p of any pair of nodes being able to communicate using TCP or UDP, a connection matrix allows/disallows connections between pairs of nodes. Then the connections are added to nodes in the following steps:

1. Attempt to add *near* connections to the immediate m neighbors (on either side) respecting the connection matrix.

2. If tunneling is enabled: identify all the missing connections between pairs of nodes, compute the overlap of their connection tables to see if tunneling is possible, and add the possible tunnel edges to the network.

3. If there are nodes with fewer than m connections on each side: each such node tries to acquire more *near* connections (to its closest neighbors, and fully respecting the connection matrix), until it has successfully acquired m *near* connections on each side.

4. If there are nodes which acquired more than m connections on each side, these excess connections are trimmed in the subsequent step.

5. Iteratively, attempt to add a *far* connection at each node. The distances traveled by these connections in the structured ring follow the distribution described in [45]. This step is repeated until every node has successfully acquired at least one *far* connection that is allowed by the connection matrix.

The all-to-all routability of the network is studied by simulating the sending of a message between each pair of nodes, and count the number of times the message is

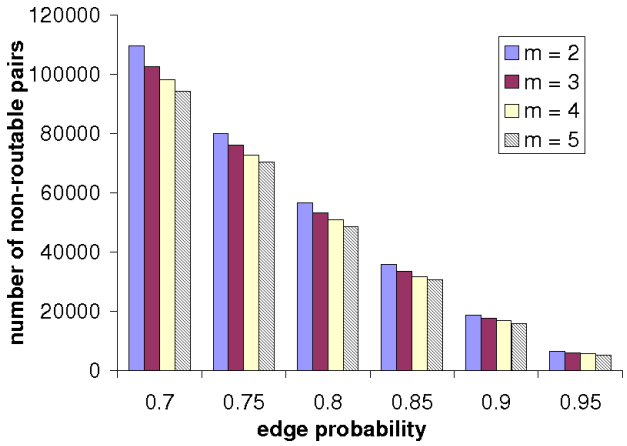


Figure 6-3. At edge likelihood of 70% (0.7), the percentage of non-routable pairs varies from 9.5% to 10.9% (the total number of pairs in the simulated network is 1,000,000)

incorrectly delivered. This experiment is conducted this for 200 different randomly generated graphs. To investigate correct routing of keys, I randomly generate 10000 different keys. For each key, I simulate the sending of a message addressed to that key from each node as the source, and count the number of times the lookup is wrongly delivered, i.e. to nodes other than the node closest to the key in identifier space. We conduct this experiment for 200 different randomly generated graphs.

Figure 6-3 shows the number of non-routable pairs (out of 1000×1000 possible pairs) of nodes for different values of the number of near neighbors m, when neither annealing routing nor tunnel edges are used. It is observed that as edge likelihood drops to 70%, the all-to-all routability of the network drops to less than 90%, i.e. more than 10% of pairs are non-routable. Similar observations are also made for the average number of instances when keys were wrongly routed (see Figure 6-4). As edge likelihood drops to 70%, there is more than 10% chance that a key is wrongly routed. Furthermore, keeping more *near* connections at each node only marginally improves the network routability.

6.3.2 Evaluating the Impact of Annealing Routing

The reduction in average number of non-routable pairs using Algorithm 6-1 is shown in Figure 6-6, with $m = 3$; tunnel edges are not enabled. It is observed that, at an edge

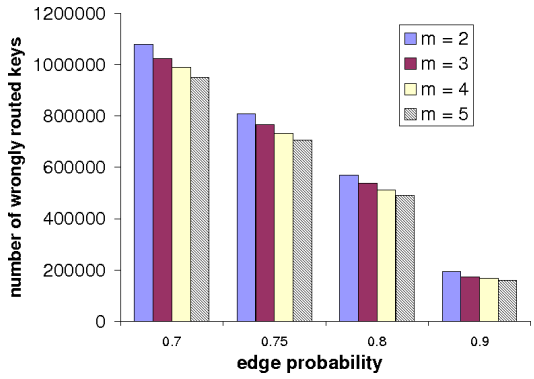


Figure 6-4. At edge likelihood of 70%, the percentage of wrongly routed keys varies from 9.5% to 10.7% (the total number of simulated messages is 10,000,000)

likelihood of 85%, the percentage of non-routable pairs with annealing routing is about 0.6%, which is less than one-fifth of the percentage when greedy routing (3.3%) is used. Even when the edge likelihood drops to 70%, the percentage of non-routable pairs (less than 3.4%) is still less than half of that when greedy routing is used (more than 10%). It should also be noted that annealing routing with $m = 3$ is more likely to reach the correct destination than using greedy routing with $m = 5$, for the same edge likelihood in this network of 1000 nodes.

The average number of hops taken by a message for both greedy and annealing routing was also measured in each simulation. In a perfectly formed structured network, both routing algorithms incur exactly the same number of hops. Otherwise, the average number of hops between P2P nodes for annealing is almost the same as for greedy routing. For an edge likelihood of 70% and $m = 3$, the ratio of number of hops incurred by annealing to that of greedy is 1.01. Therefore, annealing routing only incurs a marginal overhead in terms of number of hops.

Figure 6-7 shows the average number of wrongly routed key lookups for both annealing and greedy routing using the methodology as described in Section 6.3.1. At an edge likelihood of 70% ($m = 3$), annealing routing reduces the chances of a key being wrongly routed from 10.2% to 3.4%. By delivering a message at more than one node, the

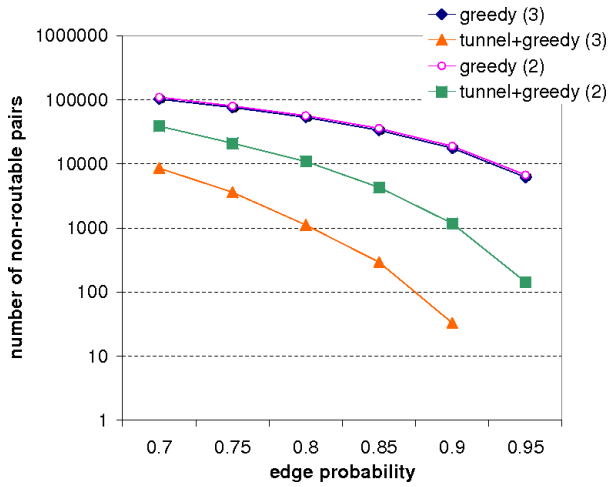


Figure 6-5. Comparing greedy routing with tunnel edges for $m = 3$ and $m = 2$. At edge likelihood of 70%, the percentage of non-routable pairs in a network of 1000 nodes is (1) 3.9% for $m = 2$, and (2) 0.86% for $m = 3$.

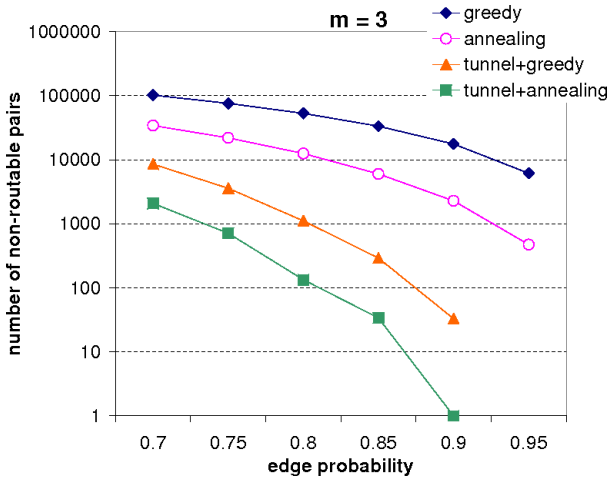


Figure 6-6. Average number of non-routable pairs. At edge likelihood of 70%, the percentage of non-routable pairs for greedy and annealing routing is (1) without tunnel edges, 10.26% and 3.4% respectively; (2) with tunnel edges, 0.86% and 0.21% respectively. At edge likelihoods of 95%, there are no non-routable pairs with tunnel edges.

annealing algorithm can result in creation of additional (more than two) replicas for a key[2] . The storage overhead due to this additional replication was found to be 9.5%.

[2] Each key in Brunet-DHT is typically replicated at two nodes, on either side of the key in identifier space.

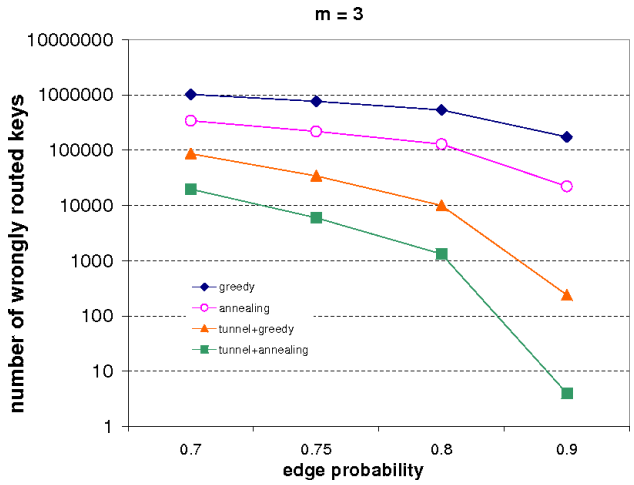


Figure 6-7. Average number of wrongly routed keys. At edge likelihood of 70%, the percentage of wrongly routed keys for greedy and annealing routing is (1) without tunnel edges, 10.2% and 3.4% respectively; (2) with tunnel edges, 0.86% and 0.19% respectively.

6.3.3 Evaluating the Impact of Tunnel Edges

The improvement in all-to-all routability of the enhanced overlay structure because of tunnel edges is shown in Figure 6-6, for $m = 3$. It is observed that at an edge likelihood of 70%, tunnel edges substantially reduce the percentage of non-routable pairs of nodes from 3.4% to 0.21% for annealing routing (from 10% to 0.86% for greedy routing).

Each virtual hop over a tunnel edge actually corresponds to two overlay hops. The actual number of hops taken by a messages addressed to exact destinations in an overlay that supported tunnel edges was also recorded. For an edge likelihood of 70% and $m = 3$, the ratio of number of actual hops to that of virtual hops was observed to be 1.14, which is a small overhead considering the improvement in routability.

Figures 6-7 also compares how tunnel edges improve the consistency of key routability of the network, for $m = 3$. It is observed that, at an edge likelihood of 70% with tunnel edges, the chances of a key being wrongly routed are 0.86% for greedy routing (and 0.19% for annealing routing).

6.4 Tunnel Edge Implementation in Brunet

Connection setup between P2P nodes is preceded by a *connection protocol* (Chapter 2) that uses the P2P overlay to rendezvous with a remote node for out-of-band exchange of information relevant for communication (through *Connect To Me (CTM)* messages), followed by a bidirectional *linking protocol* that establishes the connection. The *connection protocol* allows nodes to exchange their NAT-assigned IP address/port for hole-punching. To implement tunnel edges, the same mechanism to also used exchange information about connections to near neighbors.

Each connection in Brunet is based on an edge. Each node has one or more Uniform Resource Indicators (URIs) that abstract the edge protocols it can support and the endpoints over which it can communicate. For each type of edge, an EdgeListener is responsible for creating and maintaining edges of that type, and also sending and receiving messages over connections using that edge type. For example, to create an edge with another node using a URI *ipop.udp://128.227.56.123:4000*, the UdpEdgeListener is invoked, whereas to communicate with the same node using URI *ipop.tcp://128.227.56.123:4001*, the TcpEdgeListener is invoked. A Brunet node can have more than one EdgeListener, and new types can be easily added.

Before describing the process of creating a tunnel edge, I overview the functionality that allows each Brunet node C to also act as a message forwarder for communication between two nodes A and B. The message from the original source A is encapsulated inside a *forward request* message addressed to node C. When node C receives the message from A, it extracts the original message (from A to B), and sends it to node B. This functionality is used by a new Brunet node to identify its left and right neighbors in the P2P ring [85].

The tunneling of a connection between nodes A and B over common neighbors, is achieved by implementing an EdgeListener called TunnelEdgeListener. The URI for a node corresponding to the tunnel edges is computed dynamically by concatenating the

addresses of its closest *structured* connections. In addition to URIs corresponding to TCP or UDP, nodes also exchange their tunnel URIs inside *CTM* messages during the *connection protocol*. Once node A learns about the connections of node B, it computes the forwarding set F which is the intersection of its own connections with those listed in the tunnel URI of B.

When a node A has computed the forwarding set F with a remote node B, it then sends this information to B in an *Edge Request* using the forwarding services of one of the nodes in F. When B receives an *Edge Request*, it replies back with an *Edge Response* and also records the new tunnel edge. On receiving the *Edge Response*, the node A also records the new tunnel edge. Once the tunnel edge is successfully created, nodes A and B can subsequently create a connection between them.

This implementation does not require nodes in the forwarding to keep any state about the tunnel edges that are using them. Furthermore, the periodic *ping* messages to maintain a connection based on a tunnel edge also keep the underlying connections alive. Therefore, no extra overhead is incurred by nodes in the forwarding set. The forwarding set for a tunnel edge can change over time as connections are acquired or lost. To keep the forwarding set up to date and synchronized, nodes A and B notify each other about the changes in their connections.

When a node joins an existing overlay and cannot communicate with its immediate left and right neighbors, its tunnel URI is initially empty since it does not have any connections yet. However, it is possible that the new node can communicate with its other near neighbors, therefore it must first try to form connections with them and then use those connections to form tunnel edges with its immediate neighbors on the P2P ring. The new node learns about its other close neighbors through the *CTM* messages it receives from its immediate neighbors, which also contain a list of their *near* connections.

6.5 Experiments

In this section, I demonstrate the ability of the tunnel edge implementation to provide a complete ring of P2P nodes in environments where the majority of nodes are behind NATs and some pairs of nodes cannot communicate with each other directly. The time taken for a new node to become connected with its left and right neighbors in an existing P2P ring, in situations where these connections have to use tunnel edges, is also measured. Finally, the impact of using annealing routing and tunnel edges on connectivity within a WOW is also studied, when P2P nodes only had 81% chance of being able to setup connections.

6.5.1 Structure Verification of P2P Network

The completeness of the P2P ring is verified by iteratively "crawling" the Brunet network using the immediate right neighbor information at each node, and checking for the consistency of a node's connections with respect to its predecessor. Specifically, for every node, it is tested if its immediate left *near* connection node identifies it as immediate right *near* connection node.

When two adjacent P2P neighbors cannot form a connection, it is likely that crawling the network using neighbor information will skip a node. If the next reported node has a connection with the missing node, an inconsistency will be reported. However, in case even the second node does not have connection to the missing node, the inconsistency may go unnoticed. It is still possible to observe a 100% consistent ring with a few nodes completely missing. These hidden nodes can be detected using information logged by Brunet at each node; knowledge of the number of nodes and their identifiers is also available.

The effect of the presented techniques with respect to overlay structure is demonstrated in both, a synthetic environment with artificially created situations that prevent connection setup; and also in a large-scale PlanetLab environment, which is known to exhibit route outages between pairs of hosts.

6.5.1.1 Majority of nodes behind NATs

The first experiment demonstrates the ability of Brunet to deal with heavily NATed environments. A network of 1030 nodes involving 12 private networks (each containing 80 P2P nodes) was deployed, starting from a seed network of 70 P2P nodes that was reachable from all other nodes. Each private network consisted of virtual machines (VMs) running behind VMware NATs on 12 physical hosts. The VMware NAT has been observed to behave as a port-restricted cone; the connection setup protocols in Brunet are known to work with these NATs.

Each node was configured to form connections with 3 neighbors on its immediate left and right. Of the total of 6180 *structured near* connections reported by all nodes, about 4926 (70%) existed between nodes which were on different private networks. These connections were not possible without decentralized NAT traversal. The P2P ring was 100% consistent.

6.5.1.2 Incomplete underlying network

In this experiment, a network of 711 P2P nodes was built incrementally by adding 16 VMs, one by one, each running 40 P2P nodes, to a bootstrap network of 71 nodes. Each P2P node was configured to use a unique pre-defined UDP port number. Using IPtables, the firewall rules on the VMs were configured to drop UDP packets such that the probability of setting up UDP-based connection between any two pairs of P2P nodes was 0.95.

The P2P ring contained 35 pairs of adjacent nodes that could not setup a UDP connection because of firewall rules. These pairs of nodes were however able to connect using tunnel edges, thus rendering a complete P2P ring.

6.5.1.3 Wide-area deployment

In this experiment, a network of over 420 nodes was deployed on PlanetLab with distinct hosts in North America, South America, Asia, Europe and Australia. It was observed that as many as 9 adjacent pairs on P2P ring (listed in Table 6-2) could

Table 6-2. Adjacent P2P nodes in wide-area deployment that could not communicate using TCP or UDP

PlanetLab hosts with pair-wise routing outages
planetdev05.fm.intel.com — pli2-pa-3.hpl.hp.com
planetlab-1.fokus.fraunhofer.de — planetlab2.cosc.canterbury.ac.nz
sjtu2.6planetlab.edu.cn — planetlab1.ucb-dsl.nodes.planet-lab.org
planetlab2-tijuana.lan.redclara.net — pli1-pa-3.hpl.hp.com
planetlab-1.man.poznan.pl — pli2-pa-2.hpl.hp.com
planetlab2.ls.fi.upm.es — planetlab1.cosc.canterbury.ac.nz
planetlab2.cosc.canterbury.ac.nz — planetlab3-dsl.cs.cornell.edu
athen.dvs.informatik.tu-darmstadt.de — uestc2.6planetlab.edu.cn
uestc2.6planetlab.edu.cn — planetdev02.fm.intel.com

not communicate using TCP or UDP transports. Their inability to connect, which was observed indirectly through the fact that tunnel edges had been created, was verified directly by logging into each host and observing that ICMP messages (and SSH connections) to its peer did not go through either.

The ability of tunnel edges to form was further evaluated by deploying additional 20 P2P nodes on hosts H1 and 20 nodes running on host H2. These nodes were configured to use only UDP transports, and their hosts H1 and H2 were configured to drop UDP packets between them, thus modeling a scenario where there is a routing outage between two sites. Two instances were observed where one of the adjacent pairs was running on H1, while the other was running on H2. Tunnel edges formed between these nodes in both cases, again rendering a 100% consistent P2P ring. Without tunnel edges, these nodes would have had inconsistent view of their local neighborhoods (in identifier space), and the messages addressed to them were likely to be misdelivered.

The delay incurred by a new P2P node (on a home desktop) to get connected with its left and right neighbors on the ring using tunnel edges was also measured, over several trials. The average time to get connected with neighbors is less than 10 seconds, using UDP or TCP. The home desktop did not have an Internet path to a few nodes on PlanetLab and every time it became a neighbor to one of these nodes, it relied on tunnel edges to get connected, which took 41 seconds on average. The current

implementation delays creation of tunnel edges by an arbitrarily chosen interval of 15 seconds (to accommodate for the delay in setting up TCP or UDP connections due to hole-punching or packet losses). However, in some cases it took up to 124 seconds to form tunnel edges with the neighbors. This delay in forming tunnel edges is explained as follows.

The linking protocol for connection setup is executed through one or more linkers; each linker sends link messages using the different URIs of the remote node in parallel until it starts receiving replies. Only one linker is active at a time, during which it sends several link messages over a URI until it starts receiving replies or gives up. Initially, the new node does not have any connections to tunnel over and its tunnel URI is empty. The first linker that is created can thus only succeed using TCP or UDP. When TCP or UDP communication is not possible, it takes several attempts for the linker to finish, and the next linker to be activated. In some cases, the linker containing a usable tunnel URI (created after the node has acquired a few connections) is still waiting in the queue. An alternative implementation is possible that allows updations to the tunnel URI listed in the currently active linker, which would obviate the need to wait for the next linker.

6.5.2 Connectivity within a WOW

This section investigates the impact of using annealing routing and tunnel edges with respect to improvements in connectivity within a WOW-based Condor pool. A WOW consisting of 180 Condor worker nodes and 1 manager was setup, using a bootstrap infrastructure of 20 P2P nodes. The number of workers reported by the manager (using the *condor_status* command) was measured, which is representative of the achievable throughput of the Condor pool. Furthermore, once a worker has been chosen for job execution by the Condor manager through matchmaking, the process of job submission involves direct communication between the submit node and the worker. The all-to-all connectivity between worker nodes is also reported. The connectivity within the WOW

was measured by sending 30 ICMP ping messages between each pair of worker nodes, and counting the pair as not being connected if ping reported 100% packet loss.

Initially P2P edges are allowed to form without constraints. The Condor manager reported all 180 workers, and the workers were all-to-all connected. The P2P ring was 100% consistent. To create situations where direct communication was not always possible, the UdpEdgeListener at each node was configured to deny UDP-based connections with a probability 0.10. The probability of two nodes being able to form a UDP-based connection is thus given by $(1 - 0.10)^2 = 0.90^2 = 0.81$.

The P2P nodes were configured to only use greedy routing and no tunnel edges. The Condor manager reported at most 160 nodes, i.e. only 88% of the worker nodes were available. In addition, there were 6020 pair-wise worker connections (out of 180×180) that could not form. In another experiment, the P2P nodes were configured to use annealing routing but no tunnel edges. The Condor manager reported 177 worker nodes, and there were 859 pairs of workers than could not communicate with each other.

Finally, both annealing routing and tunnel edges were enabled. The P2P ring consisting of 201 nodes (20 bootstrap, 1 manager and 180 workers) reported 40 tunnel edges, which formed when UDP communication was denied by one of the UdpEdgeListener between adjacent P2P nodes. Only one inconsistency was observed on the P2P ring, where a tunnel edge did not form because the P2P nodes did not have any overlap on their UDP-based connections, the overlapping connections were already based on tunnel edges and the existing implementation does not support recursive tunneling. The Condor manager reported all 180 workers, and there were only 7 pairs of workers that could not communicate.

6.6 Related Work

In [8], the authors describe the implementation of a Sockets library that can be used by applications for communication between nodes subject to a variety of constraints in wide-area networks. My work, on the other hand, investigates an approach where

applications can be deployed without any modifications, by providing a virtual network with all-to-all connectivity. Furthermore, the described techniques are self-configuring and fully decentralized.

Structured P2P systems (Chord [40], Pastry [39], Bamboo [89], Kademlia [100]) have primarily focused on efficient overlay topologies [27], reliable routing under churn [89][101], and improving latency of lookups through proximity-aware routing [102]. In [26][105], the authors describe the affect of a few (5% broken pairs) Internet routing outages on wide-area deployments of structured P2P systems. On the other hand, my focus is to enable overlay structure maintenance in when a large majority of nodes behind NATs, and several scenarios hinder communication between nodes. The techniques described in this chapter facilitate correct structured routing, even when many (up to 30%) pairs of nodes cannot communicate directly using TCP or UDP.

In [99], the authors present techniques to provide content/path locality and support for NATs and firewalls, where instances of conventional overlays are configured to form a hierarchy of identifier spaces that reflects administrative boundaries and respects connectivity constraints among networks. In a Grid scenario, however, network constraints are not representative of collaboration boundaries, as virtual organizations (VOs) are known to span across multiple administrative domains.

A technique similar to tunnel edges is also described in [87], in the context of a P2P-based email system built on top of Pastry. My work, on the other hand, uses tunneling to improve all-to-all virtual-IP connectivity between WOW nodes. I also quantify the impact of the described techniques on structured routing through simulations, under different edge probabilities between nodes. Unmanaged Internet Protocol (UIP) [108] proposes to use tunneling in Kademlia DHT to route between "unmanaged" mobile devices and hosts in ad hoc environments, beyond the hierarchical topologies that make up the current Internet. However, my focus is to facilitate IP communication between Grid resources in different "managed" Internet domains.

In [109], the authors describe an algorithm for providing strong consistency of key-based routing (KBR) in dynamic P2P environments, characterized by frequent changes in membership due to node arrivals and departures. The improvements in eventual consistency by using the techniques described in this chapter can also benefit the implementation of strongly consistent KBR. Similarly in [110], authors provide asymptotic upper bounds on the number of hops taken by messages under varying rates for link and node failures, and describe heuristics to improve routing under those failures. However, their work does not consider failures of links with neighbor nodes and the subsequent impact on consistent structured routing. To complement fault-tolerant routing, the tunneling technique described in this chapter also attempts to correct the overlay structure in presence of link failures.

6.7 Conclusion

In this chapter, I have described and evaluated two synergistic approaches for improving routing in structured P2P networks: annealing routing and tunnel edges. Together, these approaches improve the all-to-all routability of a 1000-node ring structured overlay from 90% to 99%, when pairs of nodes in the underlying network only have a 70% chance of being able to communicate. Probabilistic analysis and simulation-based experiments suggest that tunnel edges are effective when each node maintains at least 3 neighbors on each side. Experiments with an implementation demonstrated the ability of Brunet to provide a consistent P2P ring, even when adjacent pairs of node in identifier space cannot communicate using TCP and UDP, in both synthetic environments and a wide-area deployment. The consistent structured P2P routing has also been shown to improve the virtual-IP connectivity within a WOW-based condor pool experimentally.

The current implementation of tunnel edges "passively" relies on an overlap to exist between connections of two nodes, for forming an tunnel edge between them. Symmetric NATed nodes can be difficult to handle using this implementation since they can only communicate with public nodes (or nodes behind cone-NATs). These nodes may not

be able to connect with any of their close neighbors[3] . Alternative implementations can also be conceived that allow nodes to discover suitable proxy nodes, "actively" form connections to these nodes, and then use these connections to form tunnel edges.

[3] Although port-prediction has been proposed to setup communication between nodes behind symmetric NATs, these techniques have not proved very effective

CHAPTER 7
IMPROVING END-TO-END LATENCY

In structured P2P systems such as IPOP, nodes arrange themselves in a well-defined topology that is dictated by their randomly chosen node identifiers. As a result, structured routing is oblivious of the geographical location of nodes. The subsequent routing delays affect the latency observed by services such as the DHT operations, the connection setup protocols, and the applications of IPOP virtual network when *shortcut* connections cannot form. Furthermore, several tools used to gather information about the IPOP deployments on PlanetLab (such as P2P ring crawler as described in Chapter 6), also rely on overlay routing to communicate with a node.

Techniques [111] have been proposed to embed locality information into node identifiers to make the overlay routing (that follows the node identifiers) latency-aware, however doing so adversely affects the uniform distribution on node identifiers and subsequently the bounds on average number of overlay hops and DHT data reliability. A well-known technique called Proximity Neighbor Selection (PNS) [27] reduces latency of structured P2P routing by requiring nodes to consider proximity information while choosing only some (not all) of their connections. It works with random assignment of P2P identifiers and does not affect the bounds on the number overlay hops and DHT reliability. In this chapter, an implementation of PNS in the IPOP system is presented, and is shown to achieve up to 30% improvement in route latency of the IPOP overlays deployed on PlanetLab.

As described earlier, connectivity constraints in wide-area can prevent creation of adaptive *shortcut* connections between IPOP nodes. Even with PNS, the latency of (multi-hop) P2P routing for virtual IP packets is still very high for many applications. In such cases, it is possible to establish a 2-hop overlay tunnel between IPOP nodes, by selecting a proxy node based on some criteria. This chapter also investigates techniques to

discover proxy nodes that minimize end-to-end latency between IPOP nodes, that cannot form a *shortcut*.

Both PNS and proxy discovery require the knowledge of Internet latencies between pairs of nodes in the network. One possibility is to take explicit round-trip time (RTT) measurements to another node. However, in a heavily NATed environment (as depicted in Sections 5.1 and 6.5.1.1, measuring latency to an arbitrary node may also require setting up a connection to that node through hole-punching, that could incur a messaging overhead of up to 10.2 Kbytes. Since connection setup incurs non-trivial cost and delay, it is not useful to establish connections for short-lived communication with arbitrary nodes, such as measuring RTT to a node. To alleviate this need for creation of short-lived connections, a low-overhead technique based on network coordinates is used that allows arbitrary pairs of nodes to estimate Internet latencies to each other without requiring an explicit measurement.

The rest of this Chapter is organized as follows. In Section 7.1, I describe network coordinates and their implementation in the IPOP system. Section 7.2 focusses on improvements in overlay routing. I describe PNS, its implementation in IPOP and evaluate the corresponding improvements in route latency of IPOP overlays on PlanetLab. The focus of Section 7.3 is on improving IPOP latency by setting up a 2-hop path through a suitably chosen proxy node when direct communication is not possible between end-nodes. I describe and evaluate techniques to discover proxy nodes that minimize latency between end-nodes, under different scenarios.

7.1 Network Coordinates

In this approach, nodes in the system compute coordinates in some low-dimensional space such that the distance between coordinates of two nodes gives an estimate of the latency between them. Two classes of algorithms for network coordinate computation exist: (1) schemes that require a fixed number of landmark nodes (GNP [112], Lighthouses [113]), and all other nodes compute their coordinates by measuring latency to these landmark

nodes and (2) schemes where all nodes in the system participate in coordinate computation Vivaldi [28], and their is no dependence on external nodes. The second class of algorithms are more suitable for an autonomous system such as IPOP, since they facilitate deployment and maintainability by allowing all nodes in the system to execute the same code.

In the Vivaldi algorithm, all nodes in the system start at random points in space, initially. By periodically taking RTT samples to a fixed set of a few random nodes, a node adjusts its coordinates. Gradually, these coordinates evolve to represent the Internet latencies between nodes. The Vivaldi algorithm is described in detail in Appendix B.

Support for Vivaldi network coordinates has been incorporated into the Brunet P2P system. Periodically, a node takes a RTT measurement to ones of its existing connections. These measurements do not impose any extra overhead, since they supplement the periodic *ping* messages to keep idle connections alive. The periodic RTT measurements are taken at an application-level, and hence exhibit high-variances that can cause slow convergence and instability The implementation of network coordinates in the IPOP system uses statistical filtering of latency samples [114], to achieve tolerance to the noise in RTT measurements, and still be responsive to sustained change in latencies.

Figure 7-1 shows the cumulative distribution function (CDF) of the estimation error for latencies between pairs of nodes (using network coordinates) on an overlay consisting of than 350 nodes on PlanetLab, measured at different instants. Each node in the overlay acquires about 15 connections on average, which it periodically samples every 10 seconds. The embedding space for the network coordinates is a 2 dimensional, with a scalar height vector. The prediction accuracy of network coordinates is observed over time. After 5 hours of bootstrapping the system, the median prediction error is observed to be about 15%, which is also in agreement with other studies on using network coordinates [115]. Even though a network of 350 nodes, all bootstrapped at once, take about 5 hours to achieve this accuracy, such networks typically grow incrementally, one node at a time. Related work [28] has shown than once there is a critical mass of well-placed nodes in a

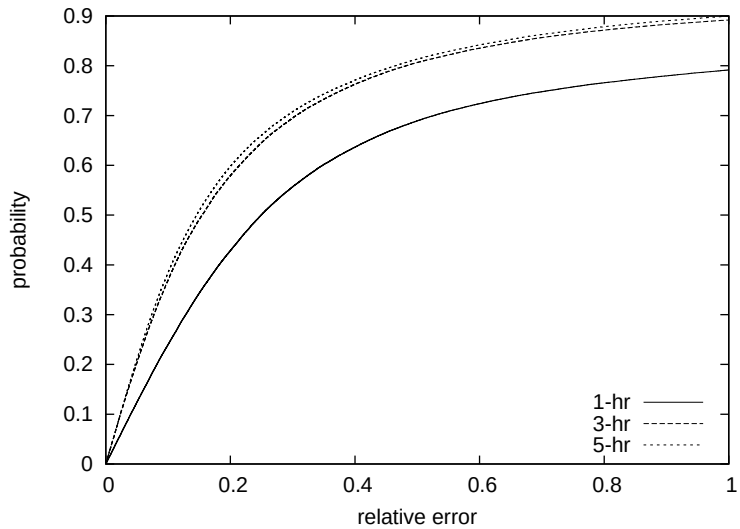


Figure 7-1. Relative error for round-trip time prediction using network coordinates measured after 1, 3 and 5 hours of bootstrapping

Vivaldi network, a new node joining the system needs to make few measurements in order to find a good place for itself.

7.2 Improving Latency of Overlay Routing

In this section, I describe a technique called Proximity Neighbor Selection (PNS) [27], to improve latency of multi-hop structured P2P routing in IPOP, by requiring P2P nodes to select a subset of their connections based on network proximity.

7.2.1 Proximity Neighbor Selection in Brunet

As described earlier, each node in the Brunet system maintains k *structured far* connections, besides the *structured near* connections it maintains to its neighbors in identifier space. By default, these connections are chosen as follows. The distances traveled by all the shortcut connections in the structured ring must follow a probability distribution function (pdf) of the following form: $pdf(d) \propto \frac{1}{d}$, where d denotes the distance traveled by the connection in identifier space. The local density of addresses is used to estimate the network size and thus d_{ave}, the average distance between nodes. Then, a random number d is chosen between d_{ave} and 2^{160} with probability proportional to $\frac{1}{d}$ and connect to the node B that is about distance d away in the address. These far

connections bound the number of overlay hops between pairs of nodes to $O(\frac{1}{k}log^2(n))$, where n is the size of the network.

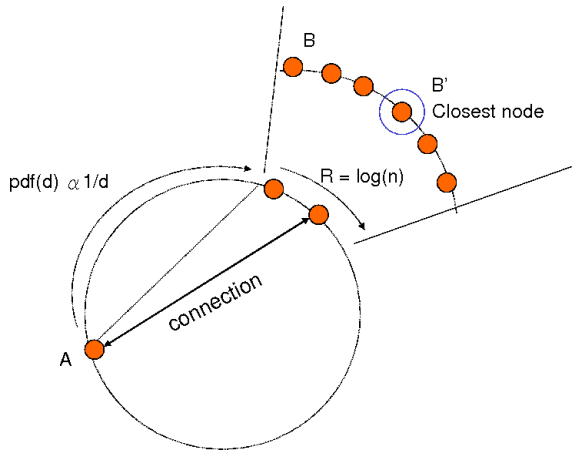


Figure 7-2. Proximity neighbor selection

PNS requires nodes to also consider network proximity while choosing *far* connections, as illustrated in Figure 7-2. In this figure, the node A considers a range R of size $O(log(n))$ nodes starting at the random target node B (in identifier space), selected using the technique described above. Instead of connecting to the node B, the node A connects with the node $B\prime$ in range R that has the smallest Internet latency to itself. In addition, each node also keeps a connection to the closest node (in terms of latency) among its $log(n)$ nearest neighbors in the identifier space. In general, an overlay path between two nodes (adequately separated in identifier space) is dominated by *far* connections, since these connections travel most of the distance in identifier space. The number of hops over *far* connections is thus a function of n, while only the last few (constant, say M) hops are over connections. By selecting *far* connections based on proximity, messages can avoid taking long latency hops for most of its progress towards the destination, thus leading to reduction in overlay latency. Furthermore, choosing *far* connections using this technique also preserves the bound on the number of overlay hops between pairs of nodes to $O(\frac{1}{k}log^2(n))$, where n is the size of the network.

The average latency of routing between pairs of nodes is evaluated as follows. Let $k = O(log(n))$, be the number of *far* connections per node. The average number of hops in an overlay path is thus given by $O(\frac{1}{k}log^2(n)) = O(log(n))$. The overall latency of the routing path is thus given by $\sum_{i=1}^{i=O(log(n))} X_i + \sum_{j=1}^{j=M} Y_j$, where X_i is the latency of the ith hop over a *far* connection and Y_j is the latency of the jth hop over a *near* connection. The expectation of this latency is thus given by

$$
\begin{aligned}
&= E[\sum_{i=1}^{i=O(log(n))} X_i + \sum_{j=1}^{j=M} Y_j] \\
= &\sum_{k=1}^{k=O(log(n))} E[X_i] + \sum_{j=1}^{j=M} E[Y_j] \\
&= \sum_{k=1}^{k=O(log(n))} E[X_i] + MA,
\end{aligned}
$$

where A is the average Internet latency between nodes. Further assume that the pair-wise latencies between nodes are exponentially distributed. The minimum of m homogeneous exponentially distributed random variables (with rate λ and mean μ) is another exponential random variable with rate $m\lambda$ and mean $\frac{1}{m}\mu$. For $m = O(log(n))$ (the size of range R in PNS), this mean evaluates to $\frac{1}{O(log(n))}\mu$. If each of the *far* connections is chosen using PNS, then the expected latency evaluates to:

$$
\begin{aligned}
&= \sum_{k=1}^{k=O(log(n))} \frac{1}{O(log(n))} A + MA \\
&= \frac{O(log(n)}{O(log(n))} A + MA \\
= NA + MA = O(A) \ \ &(M \ and \ N \ are \ constants) \\
&= O(1),
\end{aligned}
$$

which is independent of n.

To determine the closest node in range R, the PNS implementation in IPOP uses network coordinates to estimate Internet latency to another node. I now describe the implementation of PNS in the IPOP overlay.

7.2.2 Implementation in Brunet

In the Brunet P2P system, the entire 160-bit address space is divided into different address classes. One class of is that of structured addresses, which correspond to the node identifiers and DHT keys. Another address class contains directional addresses, "Left" and "Right", and is used to send messages to nodes on either side of the P2P ring. A message bearing a destination address "Left" is routed (towards left) along the structured ring, until its time-to-live (TTL) expires. In addition, a routing mode called *path-deliver* also delivers the message to each intermediate node in the path, thus providing a mechanism to address each node in a range of size R (specified as TTL).

Given the above functionality, a component called *VivaldiTargetSelector* (VTS) has been implemented that enables a node to discover the closest node $B\prime$ in the range R starting at its guessed random address s. The VTS takes as input(1) start address s of the range and (2) size r of the range. Using the start address s as forwarder, it sends a query requesting the local network coordinates to the directional address "Left" with TTL set to r, and routing mode *path-deliver*. This query is the delivered to each node in the range of size r to the left of address s, and the results are communicated back to the source node. The VTS then measures the distance of in the network coordinate space to each candidate node, and selects the closest node $B\prime$ for connection setup. An estimate of the network size is made using address range spanned by the *near* connections at a node. Periodically every 300 seconds, a node randomly selects a far connections to check if it is still optimal by querying the same range R for network coordinates. In case the connection node is no longer the closest, the connection is trimmed. These periodic checks takes care of the evolution of network coordinates over time and also arrival of a closer node in the range R.

The next Section evaluates the improvements in path latency by using PNS based on Vivaldi network coordinates. One of the key contributions of this work is verification of the applicability of network coordinates to the Brunet overlays deployed on PlanetLab.

7.2.3 Experiments

To measure the improvements in latency of overlay routing, two overlays were simultaneously deployed on two different slices (on over 350 hosts) on Planet Lab, one with PNS enabled, and the another with this functionality disabled. The node identifiers were computed using the hash of the hostname, so that P2P nodes running on the same host (but belonging to different slices or overlays) have the same identifiers. This selection of nodes identifiers assured that both the overlays had the same structure for the P2P ring. Both networks were then simultaneously crawled using the right neighbor information at each node. Using each crawled node as source, a *ping request* is sent to 10 randomly generated addresses on the overlay. These requests are routed on the overlay, until they reach a node that is closest to the destination. The destination then replies back with a *ping response*, which is routed back to the source node. The incurred delays in getting responses provide an estimate of the latencies of overlay routes. This process is repeated over 5 times for different networks.

The cumulative distribution of the observed round-trip delay is shown in Figure 7-3. It is observed that PNS results in up to 40% reduction in the average overlay latency. This reduction in overlay latency is also reflected in the decrease in crawl times and DHT operations on IPOP overlays, thus validating the applicability of PNS and network coordinates to the structured overlays deployed in wide-area.

The ability of network coordinates to correctly predict the closest node in each queried range R is also evaluated. The candidate set (with their coordinates) for each PNS query is logged. The round-trip times (RTTs) between PlanetLab nodes as measured by ICMP pings, are then used to determine the percentage of nodes that were closer to the current node than the one selected by the $VivaltiTargetSelector$ (using network

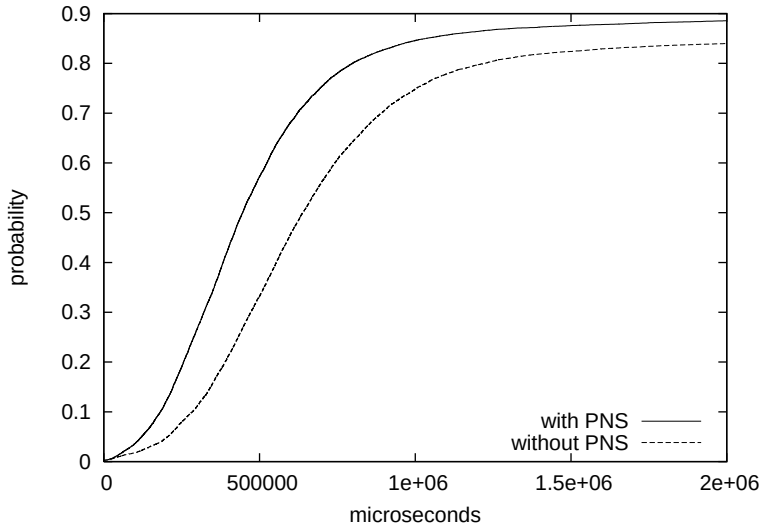


Figure 7-3. Round-trip times (RTTs) for overlay pings with PNS and without PNS. Average RTT is (1) 1.75 secs (with PNS), (2) 2.86 secs (without PNS).

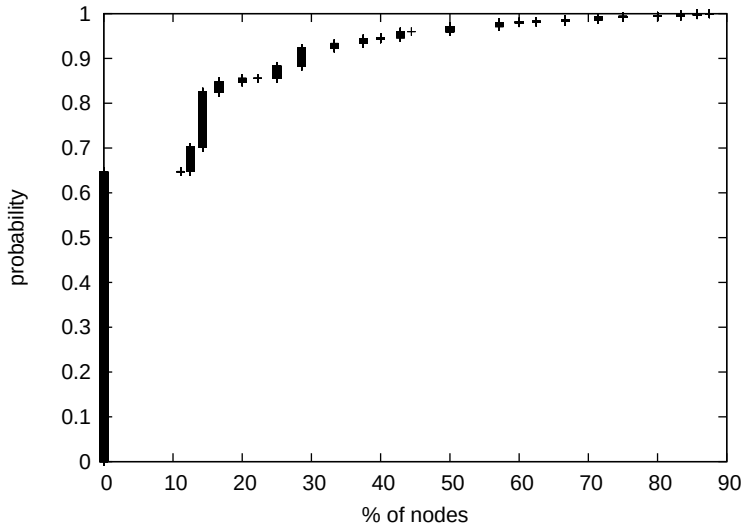


Figure 7-4. Percentage of nodes in the queried range R that are closer to the source node than the one selected by network coordinates. On average, 8.76% of the nodes in the queried range R are closer than the selected node.

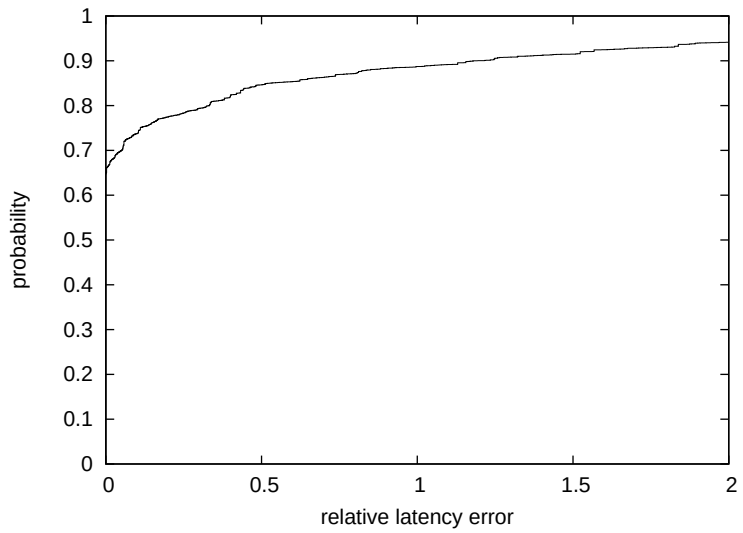


Figure 7-5. Relative latency error for choosing the closest node in the network coordinate space. On average, the relative latency error is 1.432.

coordinates). This distribution is shown in Figure 7-4. It is observed that with probability greater than 0.60 , network coordinates can correctly identify the closest node, while with probability 0.90 the selected node is among the closest 20% of the queried nodes.

The relative latency error for each selection is defined as $\frac{d_{sel}-d_{opt}}{d_{opt}}$, where d_{sel} is the latency to the closest node selected using network coordinates, and d_{opt} is the latency to the closest node indicated by ICMP ping times. The CDF of relative latency error is shown in Figure 7-5. It is observed that in more than 80% of the cases, the relative latency error is less than 0.50.

Even with PNS, the average latency on the overlay is still observed to be very high (in excess of a second) for efficient routing of virtual IP packets when direct communication is not possible between IPOP nodes. In the next section, I investigate techniques to reduce end-to-end performance of the IPOP virtual network in such scenarios, by setting up a 2-hop communication path through proxy nodes with desired capabilities.

7.3 Improving Latency of Virtual IP Communication

When direct communication is not possible, the virtual IP traffic between IPOP nodes is routed through one or more P2P nodes. In such cases, the performance of the virtual network is still limited by the capabilities of the nodes in the routing path. Scenarios have been observed where some P2P nodes limit the traffic that can flow through them. For example, the PlanetLab nodes that are used to bootstrap WOWs run on hosts that impose a cap on maximum network bandwidth that they use; large amount of IPOP traffic flowing through these nodes can cause the cap to be exceeded and all processes (including P2P node) to be killed.

The latency of virtual IP communication can be reduced by setting up a 2-hop communication path between end nodes, A and B (that cannot establish direct communication), through another node P in the network that has sufficient resources to efficiently route virtual IP packets. A similar approach is also used in the Skype [23] system. In Skype, some nodes in the system elevate to status of *supernodes*, on basis of their capabilities (CPU, memory and disk) and network connectivity. These nodes can proxy communication between other nodes in the system. The information about the supernodes in maintained in a central directory. It is retrieved by a new node on start-up, and cached locally for the future. In a system such as IPOP where nodes belong to different organizations and individuals, managing such central servers is a problem as described in Chapter 4. To achieve autonomous operation of the IPOP virtual network, decentralized techniques to discover proxy nodes are more suitable, and are investigated in this chapter.

In an IPOP overlay, a node is only aware of a few other nodes, the ones it is connected with. In order to discover a proxy node P to communicate with another node B in the network, a node A must be able to query the network for a node with certain desired characteristics that depend on the nature of the application – an interactive application might require a proxy node such that the end-to-end delay is within a delay budget, whereas a data-transfer application would need a proxy that will provide sufficient

end-to-end bandwidth. This problem can thus be formulated as a wide-area resource discovery problem. Before describing techniques that have been developed for resource discovery in IPOP, I give an overview on related work on wide-area resource discovery in the next section.

7.3.1 Wide-area Resource Discovery

Condor [19] uses a central manager to match application requirements to resource characteristics. Grid technologies [116] require each participating site in the collaboration to maintain a local directory, which maintains information about the local resources at that site. These systems require some setting up and maintaining dedicated servers, which becomes difficult when users and resource providers are individual users on the Internet.

SWORD[97] presents an extensive framework for discovering wide-area hosts for service deployment that meet certain user-defined criteria. Each resource is described by a set of attribute values which can be time-varying. Wide-area hosts form structured overlay where besides being a resource that periodically publishes its state, every host is also responsible for storing part of information about other resources. XenoSearch[117] is another system for discovering suitable Xen hosts on the Internet, for deploying services inside VM containers. The absence of special servers to store resource characteristics makes these systems highly self-managing, and thus more suitable for a fully-decentralized environment.

Both these systems use a DHT to store resource attributes. For each relevant resource attribute, an entry $(attribute\ value, resource\ identifier)$ is created in the DHT. This information is only stored as soft-state since attribute values may vary over time. In addition, SWORD also handles range queries on attribute values. Many resources might satisfy the constraints imposed by a query, which affects the query processing cost. Each query therefore also specifies the maximum cost that must be incurred to process it.

7.3.2 Discovering Proxies in IPOP

Several characteristics of the proxy node impact the performance of the virtual network as perceived by the application. These include (1) Internet latency to the end nodes, (2) network bandwidth (3) CPU speed and load that impact both latency and bandwidth. In this work, the focus is on the sum of Internet latencies of the end nodes to the proxy ($A \leftrightarrow P$ and $P \leftrightarrow B$), to achieve optimal end-to-end latency. I now describe two techniques to discover such proxy nodes.

7.3.2.1 Closest connection (CC)

In this basic approach, a node A selects its minimum latency connection as the proxy for communication with another node B. The idea is to reduce the latency of one of the 2 hops to the destination. This technique does not impose any discovery overhead. Furthermore, in the IPOP overlay, each node has $O(log(n))$ connections. In a network of about 450 nodes, it is observed that nodes can acquire about 15 connections on average. Since these connections are to random hosts (node identifiers are randomized) on the Internet, it is possible that these connections span different regions of the Internet space and the closest (out of 15) connections is also at a decent percentile among all the nodes. In the rest of this Chapter, I refer to this approach by CC.

The quality of proxy selection in this approach is limited by the connections at a node, which are chosen randomly. The next section describes an approach that can search other nodes in the network, beyond the existing connections at a node, to find proxy nodes that minimize end-to-end latency.

7.3.2.2 Expanding Segment Search (ESS)

Since Internet latency is an inter-node attribute, it can only be stored and queried with respect to another node. To obviate the need to compute and store latency to every other node in the system, this approach relies on estimates based on network coordinates. The latency for using a proxy P between IPOP nodes A and B is estimated as the sum of the distances, $A \leftrightarrow P$ and $P \leftrightarrow B$ in the network coordinate space.

Network coordinates are multi-dimensional attributes, and retrieving nodes based on network coordinates requires range queries on the system. DHT-based data structures such as Prefix Hash Trees (PHTs) [118] have been proposed to efficiently store multi-dimensional attributes such as network coordinates, and do range queries on them. However, network coordinates are dynamic and error-prone, therefore, using a PHT to store these attributes can be an overkill. Instead, it is possible to directly query nodes for their network coordinates. I have developed a generic framework based on map-reduce for parallel execution of such distributed queries and aggregation of results from a large number of nodes, which is described in Appendix C. I now describe the search algorithm.

The goal of this algorithm is to find a proxy node such that this sum S of distances ($A \leftrightarrow P$ and $P \leftrightarrow B$) is within a fraction f of direct distance $A \leftrightarrow B$. The search query is executed using the using map-reduce resource discovery framework by broadcasting over different segments of the P2P ring, as shown in Figure 7-6.

The Map function takes as argument the coordinates of nodes A (nc_a) and B (nc_b), and a delay fraction f. If the sum S_p of the distances of the local coordinates (nc_{local}) from end nodes (i.e. $|nc_a - nc_{local}| + |nc_b - nc_{local}|$) is within fraction f of the direct distance (i.e. $f \times |nc_a - nc_b|$), it returns a list containing a single tuple $(S_p, Identifier_p)$. Otherwise it returns an empty list. The Reduce function merges the child list of tuples with the current list based on the first component (sum of distances) in each tuple, to build a single list of at most K (typically 10) tuples with the best sums.

In Figure 7-6, the node A first queries a segment of the ring of size r, starting at itself, for the best K nodes that match the criteria. If no such node is found, the node A then queries a the next segment which is twice as big as the current segment. This expanding ring search terminates once atleast one node that matches the criteria has been found or when the entire ring has been queried. In the rest of this Chapter, I refer to this approach by ESS.

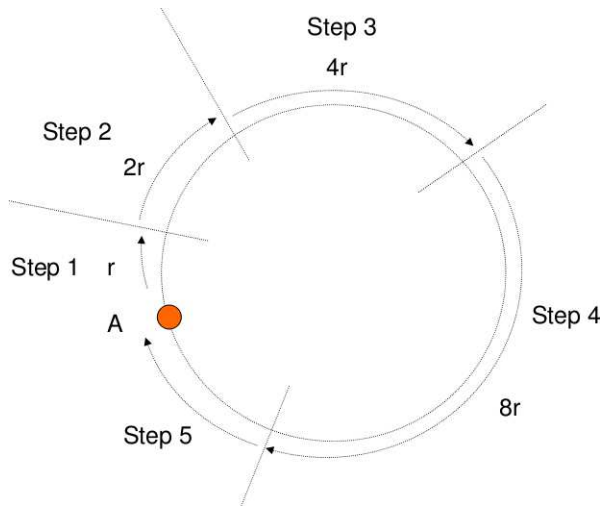


Figure 7-6. Expanding Segment Search (ESS)

7.3.3 Experiments

This section evaluates the two described techniques for selecting a proxy node to create a 2-hop communication path between IPOP nodes. Table 7-1 shows the 6 different overlays on PlanetLab that were used for experiments. The overlays within each set $((Exp1, Exp2)$, $(Exp3, Exp4)$ and $(Exp5, Exp6))$ were deployed simultaneously on two different slices, with and without PNS, respectively. These overlays were crawled to collect the list of connections (ordered by latency) and network coordinates at each node. The average number of connections acquired by overlay nodes was observed to be 15. The ordered list of connections is then used to determine the proxy node that would have been selected using CC, for each pair of nodes. Likewise, it is also possible to simulate the ESS algorithm offline for each pair of end nodes, given the node identifiers and coordinates (both of which are available from the crawl).

The correct ordered list of proxies (ordered by end-to-end latency through the proxy node) for each pair of end nodes, was computed from the ICMP ping latencies between nodes For each pair, I refer to the most ideal proxy as p_{opt}, and the end-to-end latency for using that proxy node as d_{opt}. Likewise, I refer to the proxy selected by the described

Table 7-1. Configurations of overlays used to evaluate CC and ESS

Overlay	# of nodes	PNS
Exp 1	485	Yes
Exp 2	448	No
Exp 3	482	Yes
Exp 4	454	No
Exp 5	488	Yes
Exp 6	452	No

techniques as p, and the end-to-end latency through that proxy as d_p. To compare the quality of the proxy node p selected by CC and ESS, the following metrics are used:

1. Relative Penalty (RP): The relative penalty for using p over p_{opt} is given by: $\frac{d-d_{opt}}{d_{opt}}$.

2. Absolute Penalty (AP): The absolute penalty for using p over p_{opt} is given by: $d - d_{opt}$.

In Table 7-2 summarizes the RP and AP for CC and ESS (with f: 1.2, 1.1, 1.05), observed on overlays Exp 1 and Exp 2. The average number of nodes queried in the three configurations of ESS, was 37, 57 and 87. Interestingly, it is observed that the simple approach CC performs better than either configuration of ESS, which is reflected in its lower RP and AP. Another interesting observation is that for the overlay without PNS (Exp 2), the difference in RPs and APs of proxies selected by CC and ESS is lesser than that of the overlay with PNS (Exp 1). In fact, CC performs almost the same as ESS.

This observation is explained from an information-theoretic perspective. CC relies on the information contained in a node's connections for proxy selection, whereas ESS uses the information contained in network coordinates. With 15 connections, it is possible that a node acquires a good view of the Internet space. The amount of information present in CC cannot be matched by that of network coordinates, due to their high estimation errors. Furthermore, when PNS is used, the information contained in network coordinates is also used in selecting connections close to a node, thus providing a node a better view of the network and a much better performance on CC over ESS. Similar observations were also made for other overlays as shown in Figure 7-7.

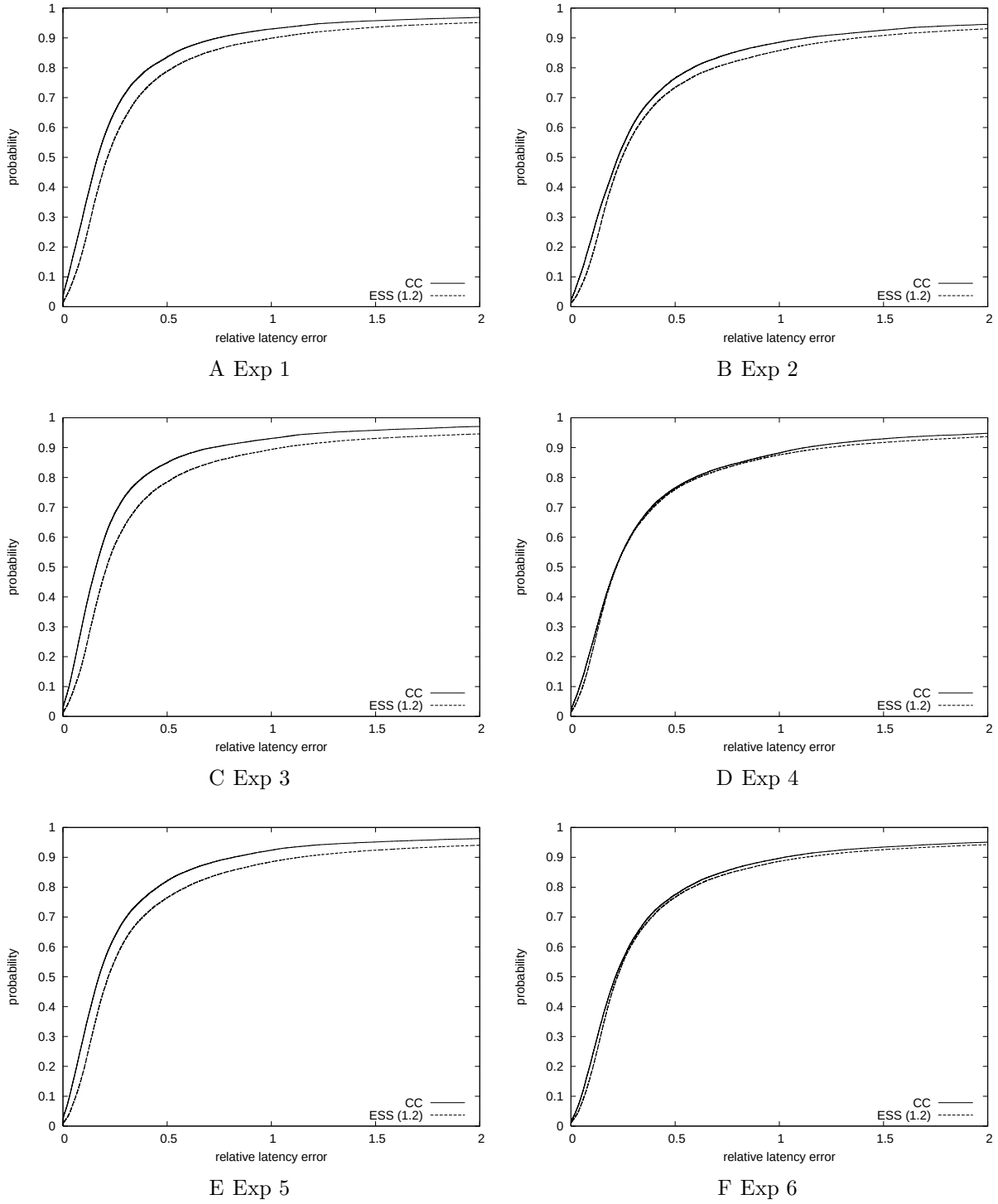


Figure 7-7. Relative penalty (RP) for using a proxy node selected by CC and ESS (1.2), when all nodes in the network can serve as proxies.

Table 7-2. Relative and absolute penalties of CC, ESS (1.1), ESS (1.2) and ESS (1.05) for overlays Exp 1 (with PNS) and Exp 2 (without PNS)

	Exp 1 (With PNS)				Exp 2 (Without PNS)			
%-ile	CC	ESS (1.2)	ESS (1.1)	ESS (1.05)	CC	ESS (1.2)	ESS (1.1)	ESS (1.05)
25	0.076 8.732	0.117 14.296	0.113 13.825	0.111 13.619	0.106 12.986	0.133 17.077	0.127 16.113	0.124 15.645
50	0.166 19.509	0.213 25.787	0.208 25.232	0.204 24.863	0.225 26.573	0.245 30.247	0.236 28.998	0.231 28.3185
75	0.337 35.572	0.427 43.622	0.417 42.693	0.415 42.003	0.470 49.051	0.537 51.238	0.519 49.689	0.511 48.98
90	0.742 63.403	1.008 71.118	1.014 69.397	1.012 68.466	1.129 92.258	1.373 110.227	1.337 106.027	1.324 105.575

Even though a low-overhead technique CC provides a much better proxy selection (in terms of latency) than ESS, the search space and the quality of proxies (by also including other criteria) in CC is still limited by the connections at a node. Scenarios arise where only a subset of the nodes in an IPOP overlay can be used as proxies. For example, PlanetLab nodes that impose limits on the amount of traffic they can handle are poor candidates for proxy selection. In addition, to proxy connections between certain pairs of nodes, it is also important that the proxy node has good network connectivity, such as it is a public node and also has sufficient CPU cycles to route IPOP traffic. It is possible that only a few (or none) of the existing connections have sufficient resources to efficiently route virtual IP traffic. With only a small subset of a node's connections to select from, it is possible that only a small amount of latency information is usable by CC.

In another set of experiments, only fraction of the nodes were randomly classified as potential proxies. These represent nodes that have good network connectivity, and other resources to efficiently route IPOP traffic. In this scenario, CC selects the closest connection that is also a potential proxy. Similarly, ESS search only considers nodes that are potential proxies. Another algorithm called ESS (first) is also introduced, which selects the first potential proxy encountered in the ESS search without considering latency at

all. This algorithm is used to verify if incorporating network coordinates provides a better proxy selection.

Table 7-3 summarizes the penalties (RP and AP) for CC and ESS when 100%, 30% and 20% of the nodes are potential proxies, for the overlays Exp 1 and Exp 2. It is observed that when less than 30% of the nodes (about 135 nodes out of 450) in the network can serve as proxies, ESS provides a better proxy selection than CC. Given a uniform likelihood of 30% for being a proxy, less than 5 (out of 15) of a node's connections can be used as proxies, which greatly reduces the latency information present available to CC for proxy selection. It is also observed that the CC performs better on overlays configured with PNS, because the connection set of each nodes already incorporates latency information from network coordinates. The average number of nodes queried by ESS (1.2) was 74 and 99 for 30% and 20% fraction of potential proxies, respectively.

Figures 7-8 and 7-9 present the CDF of the RP for the three approaches, CC, ESS (1.2) and ESS (first), on the other overlays. In each case, ESS (1.2) results in a lower RP than both CC and ESS (first).

7.3.4 Discussion

The ESS algorithm can be easily modified to also consider other criteria beyond Internet latency such as their CPU speeds, network bandwidth and connectivity, and also the policies defined by owners for usage of their resources as proxies. These criteria can be incorporated into the search by simply modifying the Map function for broadcast. It is conceivable to use a combination of search techniques for low latency – using CC when a good number of a node's connections are potential proxies and resorting to ESS only when this number is below certain threshold, such as 4.

7.4 Conclusion

In this chapter, I have addressed the latency of two different aspects in an IPOP overlay: multi-hop overlay routing and virtual IP communication between IPOP nodes in the absence of direct communication.

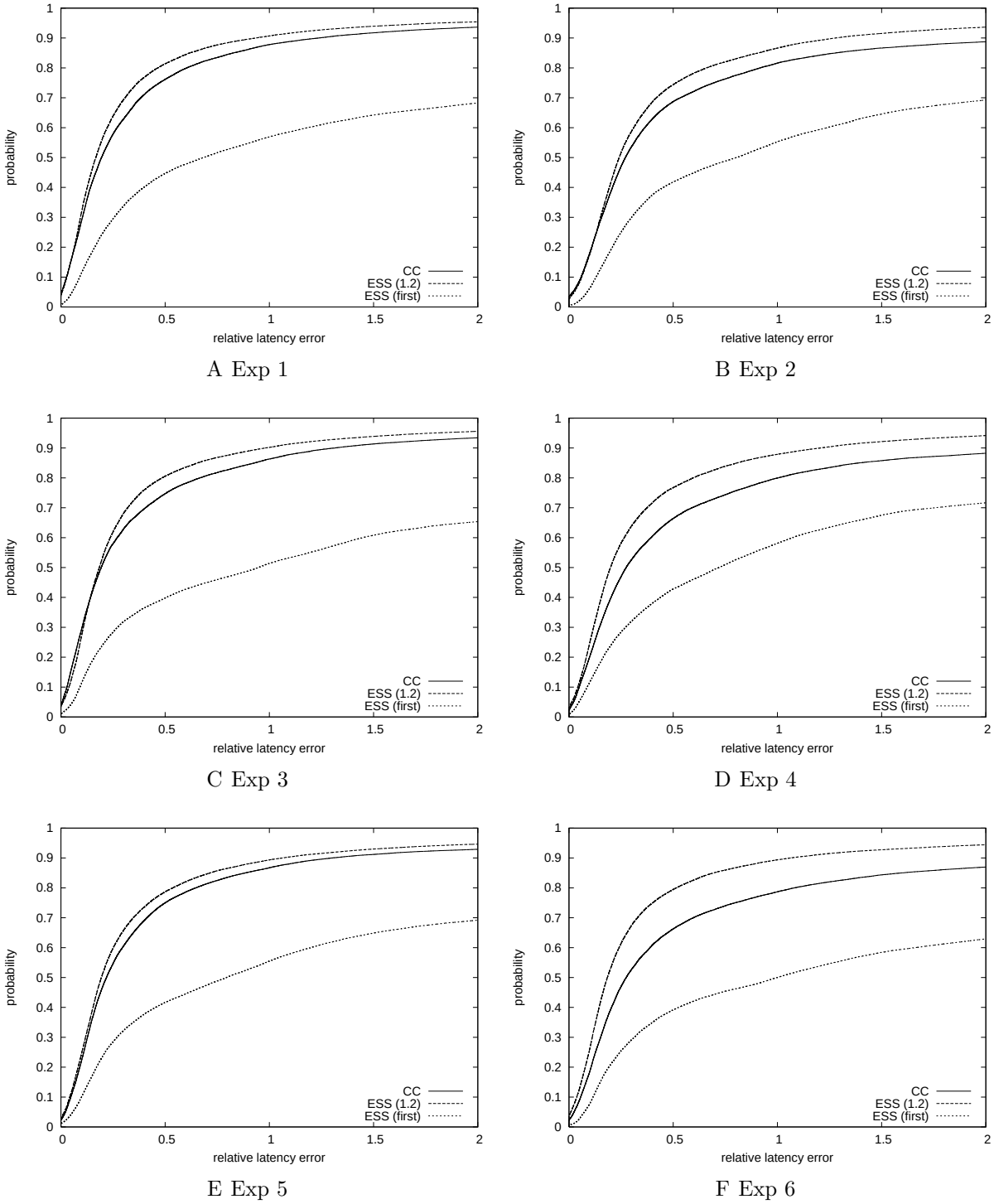


Figure 7-8. Relative penalty (RP) for using a proxy node selected by CC and ESS (1.2), when all nodes in the network can serve as proxies, when only 30 % of the nodes can serve as proxies.

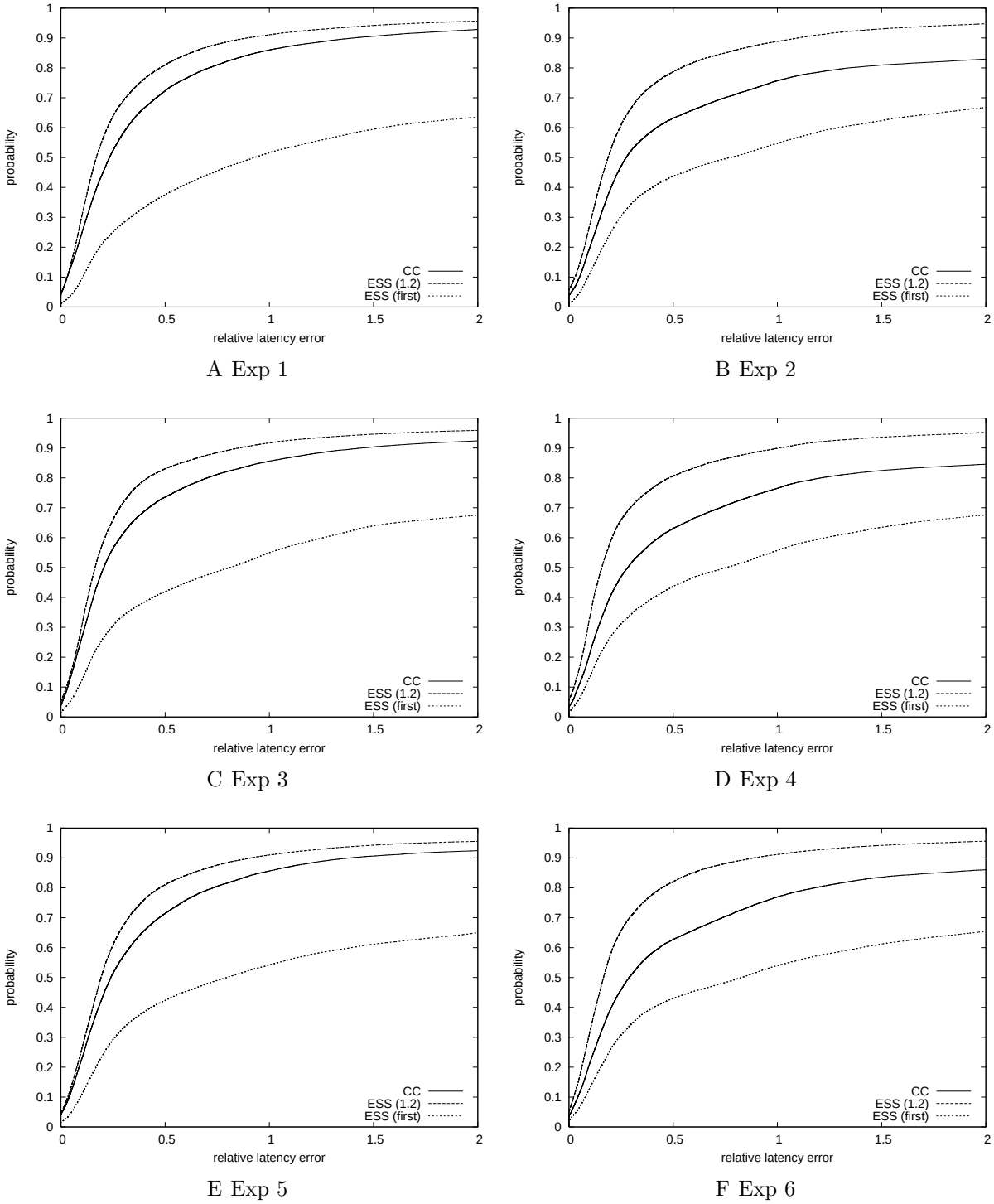


Figure 7-9. Relative penalty (RP) for using a proxy node selected by CC and ESS (1.2), when all nodes in the network can serve as proxies, when only 20 % of the nodes can serve as proxies.

Table 7-3. Relative and absolute penalties of CC and ESS (1.2) for different fraction of proxy nodes

Network		% of proxy nodes					
		100		30		20	
	%-ile	CC	ESS (1.2)	CC	ESS (1.2)	CC	ESS (1.2)
Exp 1 (With PNS)	25	0.076 8.732	0.117 14.296	0.085 11.281	0.079 9.807	0.101 13.116	0.083 10.515
	50	0.166 19.509	0.213 25.787	0.194 25.114	0.167 21.363	0.232 29.639	0.170 21.948
	75	0.337 35.357	0.427 43.622	0.472 49.260	0.368 40.266	0.558 58.390	0.377 40.612
	90	0.742 63.403	1.008 71.118	1.236 92.470	0.934 69.599	1.402 109.546	0.891 77.211
Exp 2 (Without PNS)	25	0.106 12.986	0.133 17.077	0.132 18.623	0.131 16.618	0.124 18.315	0.090 11.555
	50	0.225 26.573	0.245 30.247	0.270 36.405	0.240 31.771	0.274 38.277	0.187 23.158
	75	0.470 49.051	0.537 51.238	0.695 67.040	0.513 52.404	0.965 108.508	0.415 42.939
	90	1.129 92.258	1.373 110.227	2.482 184.208	1.278 113.434	4.210 214.333	1.097 95.353

To improve latency of overlay routing, I implemented Proximity Neighbor Selection (PNS) based on latency estimates from network coordinates. Experiments with IPOP overlays on PlanetLab show up to 30% reduction in average overlay latency The lower overlay latency in turn also results in better response times DHT-based systems that rely on overlay routing of keys.

On the other hand, to provide low end-to-end latency when direct communication is not possible, techniques have been presented to discover suitable proxy nodes that can be used to setup a 2-hop path between the end nodes. It is observed that a low overhead technique based on sampling of existing connections (CC) results in lower end-to-end latency than search based on network coordinates (ESS), when all nodes in the network are willing to serve as proxies. However, practical deployments of IPOP overlays have revealed scenarios where only a fraction of the nodes can be used as proxy nodes based on their capabilities to efficiently route IPOP traffic. In scenarios where less than 30% of the

nodes (in a deployment of 450 nodes on PlanetLab) are potential proxies, ESS provides a better selection of proxies than CC.

CHAPTER 8
CONCLUSION

My research has leveraged a combination of several scientific methods: (1) validation of existing techniques, (2) presenting novel ideas and evaluating their efficacy using simulations, (3) working implementations of research contributions and (4) large-scale experiments to demonstrate the operation of implemented techniques.

I have addressed the problem of providing bi-directional network connectivity among wide-area hosts behind NATs and firewalls, to support unmodified distributed applications on wide-area. I have presented a self-managing virtual network (IPOP) that aggregates wide-area hosts into a private network with decoupled address space management. The virtual network is functionally equivalent to a Local-area network (LAN) environment where a wealth of existing, unmodified IP-based applications can be deployed. The IPOP virtual network tunnels the traffic generated by applications over a P2P-based overlay. IPOP nodes self-configure virtual IP addresses using a DHCP implementation over a Distributed Hash Table (DHT); and self-configure IP tunnels to connect to other nodes on the network. The virtual network provides a mechanism to selectively establish 1-hop overlay links between communicating nodes, which self-optimize the virtual network with respect to overlay link latency and bandwidth.

Together of with VMs for software dissemination, IPOP facilitates creation of homogeneously configured wide-area clusters of Virtual Workstations (called WOWs). These systems can be programmed using existing batch schedulers and middleware, and support checkpoint/migration of distributed applications across domains. WOW distributed systems provide an excellent infrastructure for deployment of desktop grids and cross-domain collaboration, where new nodes can be added by simply downloading a VM image and instantiating it.

The WOW techniques have resulted in an easily deployable and highly usable VM appliance that configures ad hoc Condor pools on w ide-area hosts for high-throughout

computing. Such WOW-based Condor pools have been used to run jobs from nanoHUB [17], and from users at University of Florida.

In support of IPOP overlays for end-users, I have also addressed interesting research problems from different areas, structured P2P systems and wide-area resource discovery, amongst others. Deploying structured P2P systems on wide-area is a well-recognized challenge, connectivity constraints such as symmetric NATs and Internet route outages affect overlay structure maintenance, often leading to inconsistent routing decisions. I have presented generally applicable techniques to improve routability of structured P2P systems, thus benefiting the applications of these systems. In several large-scale distributed systems (including IPOP), when direct communication is not possible between end nodes, it is possible to setup a 2-hop communication through a suitably chosen proxy node. Different techniques have been investigated to discover proxies that minimize end-to-end latency, and their efficacy has been evaluated under different scenarios using experiments on PlanetLab – a testbed that is representative of the Internet.

APPENDIX A
NETWORK ADDRESS TRANSLATORS

Network Address Translators (NATs) first became popular as a way to deal with shortage of IPv4 addresses and also avoid difficulty of reserving IP addresses for building local-area networks. According to RFC 1918, three blocks of IPv4 addresses (10.0.0.0 - 10.255.255.255, 172.16.0.0 - 172.31.255.255 and 192.168.0.0 - 192.168.255.255) have been reserved for private networks, and are not used by hosts on public Internet

NATs are used to provide Internet connectivity to hosts in such private networks. A NAT router has two network interface: one connected to the private network and the other connected to the Internet with one or more public IP address(es). As traffic passes from the private network to the Internet, the source IP/port in each packet is translated to a NAT-assigned public IP/port. The NAT tracks these mapping from internal private IP/port to public IP/port. When a reply returns to the NAT, it uses this tracking data it stored during the outbound phase to re-write the destination IP/port. To a system on the Internet, the router itself appears to be the source/destination for this traffic. NATs have become a standard feature in routers for home and small-office Internet connections, where the price of extra IP addresses would often outweigh the benefits.

Based on the assignment of public IP, port, and treatment to inbound packets the following NAT behavior is observed for UDP traffic:

1. **Full Cone:** A full cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Furthermore, any external host can send a packet to the internal host, by sending a packet to the mapped external address.

2. **Restricted Cone:** A restricted cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Unlike a full cone NAT, an external host (with IP address X) can send a packet to the internal host only if the internal host had previously sent a packet to IP address X.

3. **Port Restricted Cone:** A port restricted cone NAT is like a restricted cone NAT, but the restriction includes port numbers. Specifically, an external host can send a

packet, with source IP address X and source port P, to the internal host only if the internal host had previously sent a packet to IP address X and port P.

4. **Symmetric:** A symmetric NAT is one where all requests from the same internal IP address and port, to a specific destination IP address and port, are mapped to the same external IP address and port. If the same host sends a packet with the same source address and port, but to a different destination, a different mapping is used. Furthermore, only the external host that receives a packet can send a UDP packet back to the internal host.

Since UDP is a stateless, NATs do not do much communication state tracking. Through a carefully crafted exchange of packets two nodes can punch "holes" in their local NATs, and then communicate directly without any external proxying. This hole-punching technique is technique is used in the STUN protocol[21], which we now describe.

A.1 Traversing NATs with UDP

With UDP it is possible to use the same local IP address and port to send packets to different destinations. Furthermore, Cone NATs map a request coming from an internal IP address/port to the same external address and port, irrespective of the destination. A host behind a NAT registers itself with a publicly accessible STUN server. During this process of contacting the STUN server, the local NAT creates a mapping from the internal IP/port to an external IP/port, which it reuses for communication with other public hosts. For each registering node, the public STUN server sees requests coming from a NAT assigned external IP/port, which it then record. The STUN server communicates with the NATed host using this IP/port.

When a node A behind a NAT wants to communicate with a node B behind another NAT, it contacts the STUN server which then communicates back to A the external IP/port that had recorded for node B. At the same time, the STUN server also informs the node B about the external IP/port of node A. The node B then sends a message to A's external IP/port, which punches a hole in B's local NAT that would later allow all packets from A's external IP/port. The packets sent out by A to B's external IP/port create a hole in A's NAT, thus allowing packets coming in from B.

Symmetric NATs are difficult to handle because they choose a different external IP and port for different destinations. The mapping that the STUN server sees, may not be usable by another node to contact the NATed host. However, many symmetric NATs exhibit a simple algorithm for choosing external IP and port, which can be learnt to predict the mapping.

A.2 Traversing NATs with TCP

Since TCP is a connection-based protocol, it has a well-defined handshake that can be used by NATs to track the state of a TCP connection. On an outgoing handshake message, the NAT expects the corresponding reply from the destination. Most TCP protocol messages are generated by the OS kernel that applications cannot control, thus making NAT traversal using TCP difficult.

STUNT protocols [22] exist for NAT traversal using TCP, but require techniques like packet sniffing/spoofing that require superuser privileges.

NAT traversal with TCP is further complicated by the fact that unlike UDP, a different local port is chosen for every outgoing TCP connection. Even though local NAT is of Cone type, it sees a different internal port for each new connection, which it maps differently.

A.3 Problems using STUN/STUNT

An important requirement in the STUN/STUNT protocols is setting publicly available STUN/STUNT servers that can be easily discovered by NATed hosts. There have been proposals to use Domain Name System (DNS) to discover STUN/STUNT servers, something requiring a wide-spread infrastructural support.

APPENDIX B
VIVALDI NETWORK COORDINATES

Synthetic coordinates allow Internet hosts to predict round-trip times (RTTs) to other hosts as follows: hosts compute their coordinates in some space such that the distance between two hosts' synthetic coordinates predicts the RTT between them on the Internet. A coordinate system is particularly useful in server selection to fetch replicated data, especially when the number of servers is large and amount of data is small. In such cases, explicit RTT measurements an easily outweigh the benefits of exploiting proximity information.

Vivaldi [28] is a simple, light-weight and fully distributed algorithm that allows hosts to compute synthetic coordinates by only communicating with only a few other hosts. The algorithm does not require any fixed network infrastructure and no distinguishable hosts.

B.1 Vivaldi Algorithm

The Vivaldi algorithm works as follows. Assume that each pair of nodes (i, j) in the network is connected by a spring. The relaxed length of the spring is equal to the RTT between two nodes it connects. The current length of the spring is equal to the distance using nodes' synthetic coordinates. Difference between the actual RTT and the RTT computer using synthetic coordinates thus corresponds to the displacement of the spring from its relaxed length.

Let L_{ij} be the actual RTT between nodes i and j, and x_i be the coordinates assigned to node i. The total prediction error in the coordinates is given as follows:

$$E = \sum_i \sum_j (L_{ij} - ||x_i - x_j||)^2 \tag{B–1}$$

where $||x_i - x_j||$ is distance between the coordinates of nodes i and j. This error function corresponds to the energy stored in the spring network connecting nodes to each other is equivalent to minimizing the prediction error.

To minimize this prediction error, the algorithm models the motion of a node in direction of total spring forces on it. Define F_{ij} to be the force vector that the spring between nodes i and j exerts on node i. From Hooke's law F is given by:

$$F_{ij} = (L_{ij} - ||x_i - x_j||) \times u(x_i - x_j). \tag{B–2}$$

The scalar quantity $(L_{ij} - ||x_i - x_j||)$ is the displacement of the spring from rest, and $u(x_i - x_j)$ is a unit vector in direction of the force on i.

To simulate the evolution of the spring network, the algorithm considers small intervals of time. At each interval, the algorithm moves each node i a small distance in the direction of each force F_{ij} and then recomputes all forces. The coordinates at the end of the interval are: $x_i = x_i + F_{ij} \times \delta$, where δ is the length of the interval.

Each node the network simulates its movement in the spring system. Each node maintains its current coordinates, starting with coordinates at the origin. Whenever a node communicates with another node, it measures the RTT to that node and also learns that node's current coordinates. In response to such sample, a node pushes itself for a short time in the direction of force computed by Equation B–2; each such movement reduces the node's error with respect to the other node in the system. As nodes communicate with each other, they converge to coordinates that predict RTT well.

B.2 Considerations for Convergence and Accuracy

This section describes the various enhancements that improve the convergence and accuracy of the algorithm. These include:

1. **Adaptive timestep:** The rate of convergence is governed by the δ timestep: large δ values result in oscillations while too small values cause slow convergence. Another issue is handling high error nodes (nodes that are still learning their coordinates), which should not cause big changes in coordinates of ones that predict RTTs well. Each node keep tracks its local prediction error using an exponentially moving average that updates the local error. Each RTT sample bears the prediction error at the remote node. The timestep δ is chosen as:

$$\delta = c_c \times \frac{local\ error}{local\ error + remote\ error} \tag{B–3}$$

where c_c is a constant fraction (< 1). This choice of δ causes only small movements of node with low prediction errors compared to the ones that predict RTT badly.

2. **Sample noise:** Transport level (UDP and TCP) RTT samples between internet nodes exhibit a large variance. Therefore, [114] propose to use non-linear moving percentile within a fixed window of RTT samples. It removes noise and also responds to actual changes in RTT. Overall, this technique provides improved accuracy and stability.

APPENDIX C
RESOURCE DISCOVERY IN IPOP

In the simplest form, resource discovery is possible by querying a subset of nodes in the network for their attributes, gathering all the results at the source node, and then compute an aggregate result. The resource discovery framework in IPOP allows for parallel query execution and result aggregation by building a tree consisting of candidate nodes rooted at the source node, propagate the query down the tree in parallel and as the results propagate up the tree, do an aggregation of results at each intermediate node. Such a query can be described as a map-reduce computation, that executes in the following two phases (Figure C-1):

1. **Map phase**: The query initiates at the root node, and traverses down the tree (see Figure C-1 A). Each node in the tree computes a local result (called *map_result*), and uses it to initialize the *reduce_result*. It then dynamically computes the list of its children and initiates similar map-reduce computations at each child node, with possibly different parameters. The node then starts waiting for child computations to return. In case the node does not have any children, it returns the *reduce_result* to its parent.

2. **Reduce phase**: In this phase, the results of the query traverse up the tree, up to the root node (see Figure C-1 B), and aggregation takes place at each intermediate node. On getting result of a child computation, the parent updates its *reduce_result*. When all child computations have returned their results, the current *reduce_result* is returned back to the parent.

C.1 Application Programmers Interface (API)

The map-reduce computations on the IPOP network can be expressed using the following three methods:

1. *Map(map_args)*: Computes the *map_result*, as the query traverses down the tree.

2. *GenerateTree(gen_args)*: Computes the list of immediate children nodes of the current node, based on the arguments *gen_args*. Returns a list of children, and arguments for their corresponding map-reduce computations. Returns an empty list if the node has no children.

3. *Reduce(reduce_args, current_reduce_result, child_result,* [*out*] *done)*: Invoked on getting a result from a child. Computes an aggregation using the current value of *reduce_result* and the *child_result*. Also returns an *out* parameter if certain

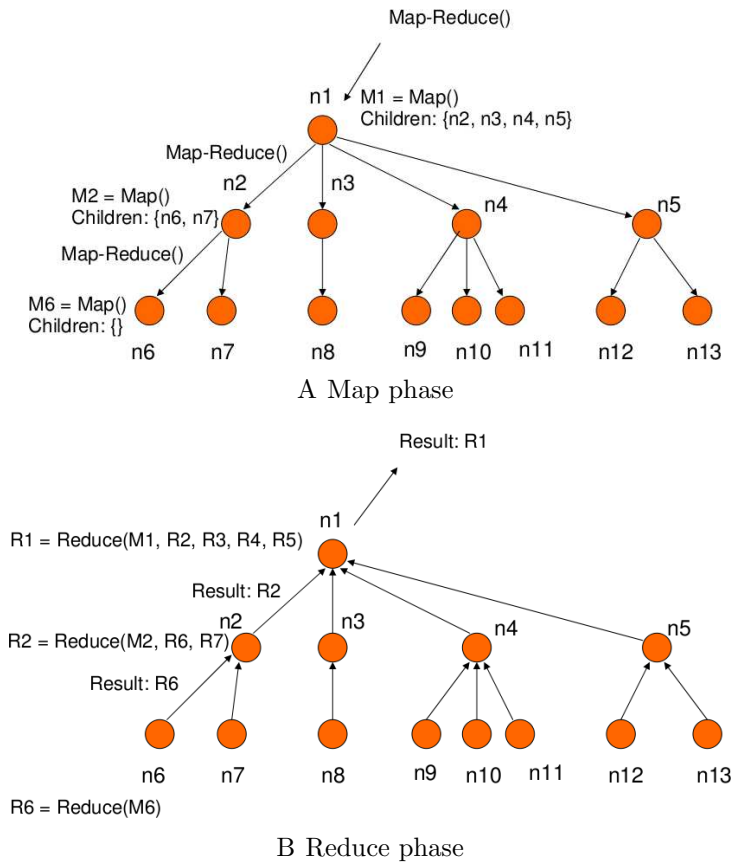


Figure C-1. Execution of a map-reduce computation over a tree of nodes

termination criteria is met, indicating that there is not need to wait for results from other children and the current *reduce_result* can be returned right away to the parent.

C.2 Tree Generation Algorithms

The following tree generation algorithms are used commonly used:

1. **Greedy tree**: Generates a unary tree rooted at the local node and ending at a node closest (in identifier space) to a destination address (specified in *gen_args*). This tree corresponds to the overlay route to a destination. The method returns a list containing single node that is closest to the destination in the local connection table. In case there is no node closer to the destination than the current node, it returns an empty list.
   This tree allows computing statistics about an overlay route between two nodes. For example, it is possible to aggregate all intermediate node addresses into a list, by specifying a Map function that returns a list containing current node address, and a Reduce function that simply does list concatenation. To count the number of tunnel edges in an overlay route, provide a Map function that returns 1 if the

child connection is based on a tunnel edge (and 0 otherwise), while Reduce simply computes a sum.

2. **Bounded broadcast**: This algorithm builds a tree consisting of IPOP nodes within a segment of the P2P ring $[A, B)$ (specified in gen_arg), starting at the current node A (see Figure C-2). To compute the children at each node, the following algorithm is used.
   To broadcast over a region of ring starting at the current node A and ending at a node B, the node determines all its connections in the region $[A, B)$, say $c_1, c_2, c_3....c_m$. The node then assigns to c_i, the segment $[c_i, c_{i+1}]$. The process continues until the current node is the only node in its assigned range. It can be shown that given $O(log(n))$ connections at each node, the maximum depth of the tree is $O(log(n))$, for a range of size n.
   To execute the computation over an arbitrary range $[C, D)$ (C is not the current node), it is required to determine the first node $C\prime$ in this range, which is doable by using greedy routing, and then initiate the computation at that node.

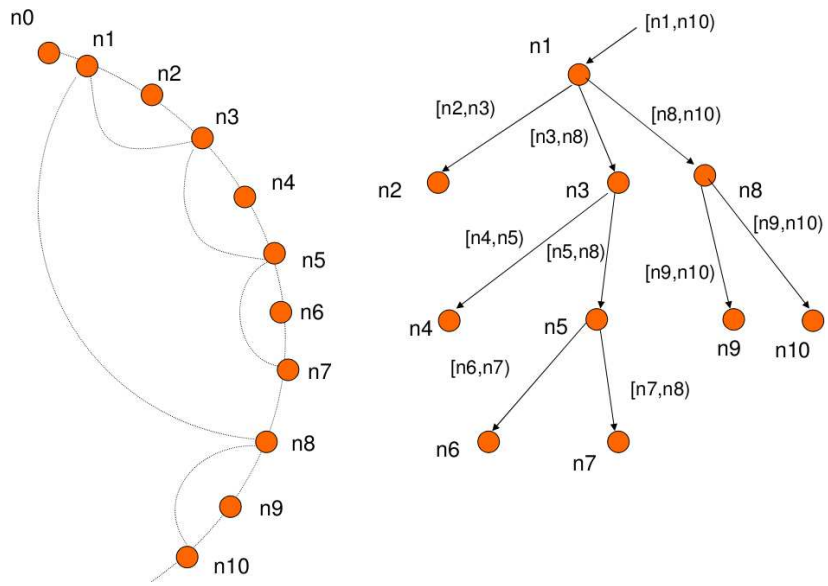


Figure C-2. Bounded broadcast over a segment of the P2P ring

C.3 Applications of Map-Reduce

The framework is extremely flexible and useful for executing different types of queries over the network, such as:

1. **Counting nodes in the range** $[C, D)$: The Map function returns 1. The Reduce function adds the child result to the current reduce result.

2. **Finding nodes that store certain data in the range** $[C, D)$: The Map function returns a list containing the current node if the data is stored locally, an empty list otherwise. The Reduce function does list concatenation.

3. **Finding K best nodes that match a certain job criteria in range** $[C, D)$: The Map function matches the job requirements with the local resources, computes a rank indicating the quality of the match, and generates an initial list containing a single tuple $(rank, local\_identifier)$. The Reduce function merges the child list of tuples with the current list to build a single list of (at most K) tuples with the best ranks.

4. **Execution of workflows:** The Map function does nothing. The GenerateTree computes dependencies for the local task and spawns child computations. Once child results are available, Reduce executes the local task and returns.

REFERENCES

[1] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *Intl. Journal of Supercomputer Applications*, vol. 11, no. 2, pp. 115–128, 1997.

[2] D. P. Anderson, J. Cobb, E. Korpella, M. Lebofsky, and D. Werthimer, ""SETI@Home: An experiment in public-resource computing"," *Communications of the ACM*, vol. 11, no. 45, pp. 56–61, 2002.

[3] M. W. Chang, W. Lindstrom, A. J. Olson, and R. K. Belew, "Analysis of HIV wild-type and mutant structures via in silico docking against diverse ligand libraries," *Journal of Chemical Information and Modeling*, vol. 47, no. 3, pp. 1258–1262, 2007.

[4] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, "Virtual workspaces in the grid," in *Proc. of Europar*, Lisbon, Portugal, Sep 2005.

[5] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo, "VMPlants: Providing and managing virtual machine execution environments for grid computing," in *in Proc. of SC 2004*, Pittsburgh,PA, Nov 2004.

[6] S. Son and M. Livny, "Recovering internet symmetry in distributed computing," in *Proc. of the 3rd Intl. Symp. on Cluster Computing and the Grid*, May 2003.

[7] S. Son, B. Allcock, and M. Livny, "Codo: Firewall traversal by cooperative on-demand opening," in *Proc. of 14th Intl. Symp. on High Performance Distributed Computing (HPDC)*, 2005.

[8] J. Maassen and H. E. Bal, "Smartsockets: Solving the connectivity problems in grid computing," in *Proc. of Symp. on High Performance Distributed Computing Symposium*, Monterey Bay, CA, Jun 2007.

[9] D. Anderson, "BOINC: A system for public-resource computing and storage," in *Proc. of the 5th Intl. Workshop on Grid Computing (GRID-2004)*, Pittsburgh, PA, Nov 2004, pp. 4–10.

[10] P. Barham, B. Dragovic, K. Fraser, and S. H. et.al., "Xen and the art of virtualization," in *Proc. of the 19th ACM symposium on Operating systems principles*, Bolton Landing, NY, 2003, pp. 164–177.

[11] R. Goldberg, "Survey of virtual machine research," *IEEE Computer Magazine*, vol. 7, no. 6, pp. 34–45, 1974.

[12] J. Sugerman, G. Venkitachalan, and B. H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor," in *Proc. of the USENIX Annual Technical Conference*, Jun 2001.

[13] A. Sundararaj and P. Dinda, "Towards virtual networks for virtual machine grid computing," in *Proc. of the 3rd USENIX Virtual Machine Research and Technology Symp.*, San Jose, CA, May 2004.

[14] X. Jiang and D. Xu, "Violin: Virtual internetworking on overlay infrastructure," in *Proc. of the 2nd Intl. Symp. on Parallel and Distributed Processing and Applications*, Dec 2004.

[15] M. Tsugawa and J. A. B. Fortes, "A virtual network (ViNe) architecture for grid computing," in *Proc. of the IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, Rhodes, Greece, Jun 2006.

[16] A. Ganguly, A. Agrawal, P. O. Boykin, and R. J. Figueiredo, "IP over P2P: Enabling self-configuring virtual IP networks for grid computing," in *Proc. of the IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, Rhodes, Greece, Apr 2006.

[17] G. Klimeck, "NanoHUB.org tutorial: Education simulation tools," in *Proc. of IEEE Conference on Nano/Micro Engineered and Molecular Systems (NEMS)*, Bankok, Thailand, Jan 2007.

[18] M. Litzkow, M. Livny, and M. Mutka, "Condor - A hunter of idle workstations," in *Proc. 8th IEEE Intl. Conference on Distributed Computing Systems (ICDCS)*, Jun 1988.

[19] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *Proc. of the 7th IEEE Intl. Symp. on High Performance Distributed Computing (HPDC)*, Chicago, IL, Jul 1998.

[20] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, *Beowulf Cluster Computing with Linux*. The MIT Press, 2002, ch. Condor - A Distributed Job Scheduler.

[21] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "RFC 3489 - STUN - Simple Traversal of User Data Protocol Protocol Through Network Address Translators," Mar 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3489.txt Last accessed: Jul 2008.

[22] S. Guha and P. Francis, "Characterization and measurement of TCP traversal through NATs and firewalls," in *Proceedings of the Internet Measurement Conference*, Berkeley, CA, Oct 2005.

[23] S. Guha, N. Daswani, and R. Jain, "An experimental study of the Skype peer-to-peer VoIP system," in *In Proc. of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, Feb 2006.

[24] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, "Virtual appliances for deploying and maintaining software,"

in *Proc. of the 17th Large Installation Systems Administration Conference (LISA 2003)*, October 2003.

[25] S.-M. Huang, Q. Wu, and Y.-B. Lin, "Tunneling IPv6 through NAT with Teredo mechanism," in *Proceedings of the International Conference on Advanced Information Networking and Applications*, Taiwan, Mar 2005.

[26] M. J. Freedman, K. Lakshminarayanan, S. Rhea, and I. Stoica, "Non-transitive connectivity and DHTs," in *Proc. of the 2nd USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, San Francisco, CA, Dec 2005.

[27] K. Gummadi, R. Gummadi, S.Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," in *Proc. of ACM SIGCOMM*, Karlsruhe, Germany, Aug 2003.

[28] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. of ACM SIGCOMM*, Portland, OR, Aug 2004.

[29] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, 2003.

[30] A. Denis, O. Aumage, R. Hofman, K. Verstoep, T. Kielmann, and H. E. Bal, "Wide-area communication for grids: An integrated solution to connectivity, performance and security problems," in *Proc. of the 13th Intl. Symp. on High Performance Distributed Computing*, Honolulu, Hawaii, Jun 2004.

[31] A. Sundararaj, A. Gupta, and P. Dinda, "Dynamic topology adaptation of virtual networks of virtual machines," in *Proc. Seventh Workshop on Languages, Compilers and Run-time Support for Scalable Systems (LCR)*, Oct 2004.

[32] P. K. Gummadi, S. Saroiu, and S. D. Gribble, "A measurement study of Napster and Gnutella as examples of peer-to-peer file sharing systems," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 1, p. 82, Jan 2002.

[33] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the Gnutella network," *IEEE Internet Communication*, vol. 6, no. 1, pp. 50–57, Feb 2002.

[34] N. Leibowitz, M. Ripeanu, and A. Wierzbicki, "Deconstructing the Kazaa network," in *Proceedings of 3rd Workshop on Internet Applications*, San Jose, CA, Jun 2003.

[35] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *In Proc. of the Intl Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, Cambridge, MA, Nov 2000.

[36] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility," in *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schoss Elmau, Germany, May 2001.

[37] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cooperative file system," in *Proc. of the 18th ACM Symp. on Operating Systems Principles*, Banff, Canada, Oct 2001.

[38] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *Proc. of the ACM SIGCOMM 2001*, 2001.

[39] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proc. of the IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov 2001.

[40] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[41] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A global-scale overlay for rapid service deployment," *IEEE J-SAC*, vol. 22, no. 1, pp. 41–53, Jan 2004.

[42] R. eka Albert, H. Jeong, and A. asl o Barab asi, "Error and attack tolerance of complex networks," *Nature*, vol. 406, pp. 378–381, 2000.

[43] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002, special issue on peer-to-peer networking.

[44] P. O. Boykin, J. S. A. Bridgewater, J. S. Kong, K. M. Lozev, B. A. Rezaei, and V. P. Roychowdhury, "A symphony conducted by Brunet," in *arXiv:0709.4048v1*, Sep 2007.

[45] J. Kleinberg, "Navigation in small world," *Nature*, vol. 406, p. 845, 2000.

[46] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *Proc. of the 2005 USENIX Annual Technical Conference (USENIX '05)*, Anaheim, California, Apr 2005.

[47] S. Guha, Y. Takeda, and P. Francis, "NUTSS: A SIP based approach to UDP and TCP connectivity," in *Proc. of Special Interest Group on Data Communications (SIGCOMM) Workshops, Portland, OR*, Aug 2004, pp. 43–48.

[48] R. J. Figueiredo, P. Dinda, and J. A. B. Fortes, "A case for grid computing on virtual machines," in *Proc. of the 23rd IEEE Intl. Conference on Distributed Computing Systems (ICDCS)*, Providence, Rhode Island, May 2003.

[49] S. Adabala, V. Chadha, P. Chawla, R. J. Figueiredo, J. A. B. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, "From virtualized resources to virtual computing grids: The In-VIGO system," *Future Generation Computing Systems, Special issue on Complex Problem-Solving Environments for Grid Computing*, vol. 21, no. 6, Apr 2005.

[50] A. Shoykhet, J. Lange, and P. Dinda, "Virtuoso: A system for virtual machine marketplaces." Northwestern University, Jul 2004, technical Report NWU-CS-04-39.

[51] J. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, 2005.

[52] I. Foster and A. Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing," in *Proc. of the 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, Feb 2003.

[53] A. S. Cheema, M. Muhammad, and I. Gupta, "Peer-to-peer discovery of computational resources for grid applications," in *Proc. of the 6th IEEE/ACM Workshop on Grid Computing*, Seattle, WA, Nov 2005.

[54] A. Iamnitchi, I. Foster, and D. C. Nurmi, "A peer-to-peer approach to resource location in grid environments," in *Proc. of the 11th Symp. on High Performance Distributed Computing*, Edinburgh, UK, Aug 2002.

[55] J. Cao, O. M. K. Kwong, X. Wang, and W. Cai, "A peer-to-peer approach to task scheduling in computation grid," *Intl. Journal of Grid and Utility Computing*, vol. 1, no. 1, 2005.

[56] N. Therning and L. Bengtsson, "Jalapeno - Decentralized grid computing using peer-to-peer technology," in *Proc. of the 2nd Conference on Computing Frontiers*, Ischia, Italy, 2005.

[57] A. J. Chakravarti, G. Baumgartner, and M. Lauria, "The organic grid: Self-organizing computation on a peer-to-peer network," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 35, no. 3, May 2005.

[58] N. Andrade, L. Costa, G. Germoglio, and W. Cirne, "Peer-to-peer grid computing with the ourgrid community," in *Proc. of the 23rd Brazilian Symp. on Computer Networks*, May 2005.

[59] N. A. Al-Dmour and W. J. Teahan, "Parcop: A decentralized peer-to-peer computing system," in *Proc. of the 3rd Intl. Symp. on Algorithms, Models and Tools for Parallel Computing on Heterogenous Networks*, Jul 2004.

[60] R. Cox, A. Muthitacharoen, and R. Morris, "Serving DNS using Chord," in *Proc. of the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge,MA, Mar 2002.

[61] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet Indirection Infrastructure," *IEEE/ACM Transactions on Networking*, vol. 12, no. 2, pp. 215–218, Apr 2004.

[62] J. Kannan, A. Kubota, and K. Lakshminarayan, "Supporting legacy applications over I3." Computer Science Division, University of California, Berkeley, Jun 2004, technical report no. UCB/CSD-04-1342.

[63] L. Zhou and R. van Renesse, "P6P: A peer-to-peer approach to internet infrastructure," in *Proc. of Peer-to-peer Systems III: Third Intl. Workshop (IPTPS)*, 2004, p. 75.

[64] L. Zhou, R. van Renesse, and M. Marsh, "Implementing IPv6 as a peer-to-peer overlay network," in *21st IEEE Symp. on Reliable Distributed Systems (SRDS)*, 2002, p. 347.

[65] T. E. Anderson, D. E. Culler, and D. A. P. et.al, "A case for network of workstations: NOW," *IEEE Micro*, February 1995.

[66] C. Chiapusio. (2007, May) distributed.net. [Online]. Available: http://distributed.net Last accessed: Jul 2008.

[67] B. Calder, A. A. Chien, J. Wang, and D. Yang, "The entropia virtual machine for desktop grids," in *CSE technical report CS2003-0773, University of California, San Diego*, San Diego, CA, Oct 2003.

[68] X. Zhang, K. Keahey, I. Foster, and T. Freeman, "Virtual cluster workspaces for grid applications," Argonne National Laboratories, Tech. Rep. ANL/MCS-P1246-0405, Apr 2005.

[69] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. 2nd Symp. on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[70] M. Lorch, D. Kafura, I. Fisk, K. Keahey, G. Carcassi, T. Freeman, T. Peremutov, and A. S. Rana, "Authorization and account management in the Open Science Grid," in *Proc. of the 6th IEEE/ACM Workshop on Grid Computing*, Nov 2005.

[71] D. Wolinsky, A. Agrawal, P. O. Boykin, J. Davis, A. Ganguly, V. Paramygin, P. Sheng, and R. Figueiredo, "On the design of virtual machine sandboxes for distributed computing in wide area overlays of virtual workstations," in *Proc. of the 1st Workshop on Virtualization Technologies in Distributed Computing (VTDC), with Supercomputing*, Tampa, FL, Nov 2006.

[72] T. Bailey and C. Elkan, "Fitting a mixture model by expectation maximization to discover motifs in biopolymers," in *Proc. of the Second Intl. Conference on Intelligent Systems for Molecular Biology.* Menlo Park, CA: AAAI Press, 1994, pp. 28–36.

[73] G. Olsen, H. Matsuda, R. Haggstrom, and R. Overbeek, "fastDNAml: A tool for construction of phyllogenetic trees of dna sequences using maximum likelihood," *Comput. Appl. Biosci.*, vol. 10, pp. 41–48, 1994.

[74] C. Stewart, D. Hart, D. Berry, G. Olsen, E. Wernert, and W. Fischer, "Parallel implementation and performance of fastDNAml - a program for maximum likelihood phylogenetic inference," in *Proc. of IEEE/ACM Supercomputing Conference (SC01)*, 2001.

[75] K. Melero, M. Hardy, and M. Lucas, "Open-source software technologies for data archiving and online geospatial processing," in *Proc. of the IEEE Geoscience and Remote Sensing Symposium*, Toulouse, France, Jul 2003.

[76] V. Sunderam, J. Dongarra, A. Geist, and R. Manchek, "The PVM concurrent computing system: Evolution, experiences, and trends," *Parallel Computing*, vol. 20, no. 4, pp. 531–547, Apr 1994.

[77] M. Kozuch and M. Satyanarayanan, "Internet suspend/resume," in *4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002), 20-21 June 2002, Callicoon, NY, USA*. IEEE Computer Society, 2002, p. 40.

[78] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," in *Proc. of USENIX Operating System Design and Implementation (OSDI)*, 2002.

[79] D. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawake, and C. Packer, "Beowulf: A parallel workstation for scientific computation," in *Proc. of the Intl. Conference on Parallel Processing (ICPP)*, 1995.

[80] H. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, T. Kielmann, J. Maassen, R. van Nieuwpoort, J. Romein, L. Renambot, T. Rühl, R. Veldema, K. Verstoep, A. Baggio, G. Ballintijn, I. Kuz, G. Pierre, M. van Steen, A. Tanenbaum, G. Doornbos, D. Germans, H. Spoelder, E.-J. Baerends, S. van Gisbergen, H. Afsermanesh, D. van Albada, A. Belloum, D. Dubbeldam, Z. Hendrikse, B. Hertzberger, A. Hoekstra, K. Iskra, D. Kandhai, D. Koelma, F. van der Linden, B. Overeinder, P. Sloot, P. Spinnato, D. Epema, A. van Gemund, P. Jonker, A. Radulescu, C. van Reeuwijk, H. Sips, P. Knijnenburg, M. Lew, F. Sluiter, L. Wolters, H. Blom, C. de Laat, and A. van der Steen, "The distributed ASCI supercomputer project," *SIGOPS Oper. Syst. Rev.*, vol. 34, no. 4, pp. 76–96, 2000.

[81] O. Aumage, R. F. H. Hofman, and H. E. Bal, "NetIbis: An efficient and dynamic communication system for heterogenous grids," in *Proc. of CCGrid 2005*, Cardiff, UK, May 2005.

[82] D. Xu, P. Ruth, J. Rhee, R. Kennel, and S. Goasguen, "Autonomic adaptation of virtual distributed environments in a multi-domain infrastructure," in *Proc. of IEEE Intl Symp. on High-Performance Distributed Computing (HPDC), Hot Topics Session*, Paris, France, Jun 2006.

[83] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao, "Cluster computing on the fly: P2P scheduling of idle cycles in the internet," in *Proc. of the 3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, CA, Feb 2004.

[84] A. S. Grimshaw and W. A. Wulf, "Legion: Flexible support for wide-area computing," in *Proc. of the 7th ACM SIGOPS European Workshop*, Ireland, 1996.

[85] A. Ganguly, A. Agrawal, P. O. Boykin, and R. J. Figueiredo, "WOW: Self-organizing wide area overlay networks of virtual workstations," in *Proc. of the IEEE Intl Symp. on High-Performance Distributed Computing (HPDC)*, Paris, France, Jun 2006.

[86] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scalable application-level anycast for highly dynamic groups," in *Proc. of the 5th Intl. Workshop on Networked Group Communications (NGC)*, Munich, Germany, Sep 2003.

[87] A. Mislove, A. Post, A. Haeberlen, and P. Druschel, "Experiences in building and operating epost, a reliable peer-to-peer application," in *Proc. of ACM SIGOPS/EuroSys European Conf. on Computer Systems*, Leuven, Belgium, Apr 2006.

[88] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "Opendht: A public DHT service and its uses." in *Proc. of ACM SIGCOMM*, Philadelphia, PA, Aug 2005.

[89] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proc. of USENIX Technical Conference*, Jun 2004.

[90] A. Muthitacharoen, S. Gilbert, and R. Morris, "Etna: A fault-tolerant algorithm for atomic mutable dht data," in *Technical Report MIT-LCS-TR-993, MIT-LCS*, Jun 2005.

[91] Z. Li and M. Parashar, "Comet: A scalable coordination space for decentralized distributed environments," in *Proceedings of the 2nd International Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P)*, San Diego, CA, Jul 2005.

[92] D. Gelernter, "Generative communication in linda," in *ACM Transactions on Programming Language Systems*, vol. 7, no. 1, 1985, pp. 80–112.

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

[94] Z. Xu and Y. Hu, "SBARC: A supernode based peer-to-peer file sharing system," in *Proc. of the 8th IEEE International Symposium on Computers and Communication (ISCC)*, Antalaya, Turkey, Jun 2003.

[95] B. Y. Zhao, Y. Duan, L. Huang, A. D. Joseph, and J. D. Kubiatowicz, "Brocade: Landmark routing on overlay networks," in *In Proc. of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, Mar 2002.

[96] A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting scalable multi-attribute range queries," in *Proc. of ACM SIGCOMM*, Portland, OR, 2004.

[97] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Distributed resource discovery on planetlab with SWORD," in *Proc. of the ACM/USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, San Francisco, CA, Dec 2004.

[98] C. Schmidt and M. Parashar, "Enabling flexible queries with guarantees in p2p systeme," in *IEEE Network Computing (Special Issue on Information Dissemination on the Web)*, vol. 3, Jun 2004, pp. 19–26.

[99] A. Mislove and P. Druschel, "Providing administrative control and autonomy in structured peer-to-peer overlays," in *Proc. of the 3rd Intl. Workshop on Peer-to-peer systems*, San Diego, ca, Feb 2004.

[100] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proc. of the Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, Mar 2002.

[101] M. Castro, M. Costa, and A. Rowstron, "Performance and dependability of structured peer-to-peer overlays," in *Proc. of the Conf. on Dependable Systems and Networks*, Jun 2004.

[102] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Topology-aware routing in structured peer-to-peer overlay networks," in *Microsoft Research MSR-TR-2002-82*, Sep 2002.

[103] J. Liang, R. Kumar, and K. Ross, "The FastTrack overlay: A measurement study," in *Computer Networks (Special Issue on Overlays)*, 2005.

[104] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil, "A performance vs. cost framework for evaluating DHT design tradeoffs under churn," in *Proc. IEEE INFOCOM*, 2005.

[105] S. Gerding and J. Stribling, "Examining the trade-offs of structrured overlays in dynamic non-transitive network," Dec 2003, Class project. [Online]. Available: http://pdos.lcs.mit.edu/~strib/docs/projects/networking_fall2003.ps Last accessed: Jul 2008.

[106] B. A. Miller, T. Nixon, C. Tai, and M. D. Wood, "Home networking with Universal Plug and Play," *IEEE Communications Magazine*, vol. 39, no. 12, pp. 104–109, Dec 2001.

[107] A. Ganguly, D. Wolinsky, P. O. Boykin, and R. J. Figueiredo, "Decentralized dynamic host configuration in wide-area overlays of virtual workstations," in *Proc. of the PCGrid workshop*, Long Beach, CA, Mar 2006.

[108] B. Ford, "Unmanaged Internet Protrocol: Taming the edge network management crisis," in *Proc. of the Workshop on Hot Topics in Networks (HotNets)*, Cambridge, MA, Nov 2003.

[109] W. Chen and X. Liu, "Enforcing routing consistency in structured peer-to-peer overlays: Should we and could we?" in *In Proc. of the Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, Feb 2006.

[110] J. Aspnes, Z. Diamadi, and G. Shah, "Fault-tolerant routing in peer-to-peer systems," in *Proc. of the Symp. on Principles of Distributed Computing (PODC)*, Monterey, CA, Jul 2002.

[111] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topology-aware overlay construction and server selection," in *Proc. of the IEEE INFOCOMM*, New York, NY, 2002.

[112] T. S. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *Proc. of IEEE INFOCOMM*, New York, NY, Jun 2002.

[113] M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti, and T. Harris, "Lighthouses for scalable distributed location," in *Proc. of the Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, Feb 2003.

[114] P. Pietzuch, J. Ledlie, and M. Seltzer, "Supporting network coordinates on PlanetLab," in *Proc. of the 2nd USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, San Francisco, CA, Dec 2005.

[115] P. Pietzuch, J. Ledlie, M. Mitzenmacher, and M. Seltzer, "Network-aware overlays with network coordinates," in *Proc. of the Workshop on Dynamic Distributed Systems (IWDDS)*, Lisbon, Portugal, Jul 2006.

[116] A. Iamnitchi and I. Foster, "On fully decentralized resource discovery in grid environments," in *Proc. of the International Workshop on Grid Computing*, Pittsburgh, PA, Nov 2004.

[117] D. Spence and T. Harris, "Distributed resource discovery in xenoserver open platform," in *Proc. of the IEEE Intl. Symp. on High Performance Distributed COmputing (HDDC)*, Seattle, WA, Jun 2003.

[118] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein, "A case study in building layered dht applications," in *Proceedings of the Conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2005, pp. 97–108.

BIOGRAPHICAL SKETCH

Arijit Ganguly grew up in New Delhi, the capital city of India. He attended the Indian Institute of Technology (IIT), Guwahati (India) for his undergraduate studies (B.Tech.) in computer science. He became interested in computer systems, distributed computing and networks, and later decided to pursue graduate studies. After his graduation from IIT Guwahati, Arijit joined University of Florida in Fall 2002.

In Fall 2003, Arijit started his PhD research under Dr. Renato Figueiredo at the Advanced Computing and Information Systems (ACIS) Laboratory, where he was later appointed as a Research Assistant (RA). At ACIS, he got an opportunity to do cutting-edge research on on virtual machines, networks and P2P systems; and in the process publish papers in highly acclaimed conferences and journals. Together with ACIS researchers, he has also been involved in the development of open-source software called IPOP, which is already being used by researchers at the University of Florida. To complement his acedemic experience, Arijit also got an opportunity to do summer internships at VMware, IBM Research and Microsoft in 2005, 2006 and 2007, respectively. Upon his graduation, Arijit plans to take up a full-time position in industry in the area of computer systems.