Cluster Comput (2009) 12: 239–256
DOI 10.1007/s10586-009-0075-1

Improving peer connectivity in wide-area overlays of virtual workstations

Arijit Ganguly · P. Oscar Boykin · David I. Wolinsky · Renato J. Figueiredo

Received: 1 January 2009 / Accepted: 5 January 2009 / Published online: 17 January 2009

Abstract Self-configuring virtual networks rely on structured P2P routing to provide seamless connectivity among nodes through overlay routing of virtual IP packets, support decentralized hole-punching to establish bi-directional communication links among nodes behind network address translators, and dynamic configuration of virtual IP addresses. Our experiences with deployments of virtual networks in support of wide-area overlays of virtual workstations (WOWs) reveal that connectivity constraints imposed by symmetric NATs and by Internet route outages often hinder P2P overlay structure maintenance and routability, subsequently limiting the ability of WOWs to deliver high-throughput computing through aggregation of resources in different domains.

In this paper, we describe and evaluate two novel approaches which are generally applicable and fully decentralized, and show that they improve routability of structured P2P networks in such connectivity constrained environments: (1) a fault-tolerant routing algorithm based on simulated annealing from optimization theory, and (2) tunneling of connections between adjacent nodes (in the P2P identifier space) over common neighbors when direct communication is not possible. Simulation-based analyses show that (1) when pairs of nodes only have 70% chance of being able to communicate directly, the described approaches improve all-to-all routability of the network from 90% to 99%, and (2) even when only 70% of the nodes are behind NATs that include symmetric NATs, these techniques improve the all-to-all connectivity of the network from less than 95% to more than 99%. We have implemented these techniques in the IP-over-P2P (IPOP) virtual network and have conducted experiments with a 180-node WOW Condor pool, demonstrating that, at 81% probability of establishing a pair-wise connection, annealing and tunneling combined allow all nodes to be connected to the pool, compared to only 160 nodes in the absence of these techniques.

Keywords P2P · DHT · Virtual network · Overlay · Routing

A. Ganguly (✉) · P.O. Boykin · D.I. Wolinsky · R.J. Figueiredo
Advanced Computing and Information Systems Laboratory, University of Florida, Gainesville, FL 32611, USA
e-mail: aganguly@acis.ufl.edu

P.O. Boykin
e-mail: boykin@acis.ufl.edu

D.I. Wolinsky
e-mail: davidiw@acis.ufl.edu

R.J. Figueiredo
e-mail: renato@acis.ufl.edu

1 Introduction

Wide-area overlays of virtual workstations (WOWs) are appealing infrastructures for the creation of high-throughput computing pools and cross-domain collaborative environments [1, 4, 11, 22] due to their ability of self-configuring functionally homogeneous virtual networks of virtual machines on top of a heterogeneous wide-area physical infrastructure [14, 32]. Like several related efforts (such as Chord [30], Kademlia [24] and Pastry [29]), WOWs rely on structured P2P overlays to provide the core service of message routing and additional capabilities such as object storage and retrieval. In the case of WOWs, a structured P2P virtual network (IPOP [13]) provides all-to-all connectivity among nodes, automatic configuration of virtual IP addresses of nodes using a decentralized Dynamic Host Configuration Protocol (DHCP) [15] implementation, and

self-optimization through creation of direct overlay links between virtual IP nodes [14].

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. 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. These constraints can render structured P2P routing inconsistent, negatively affecting routability and services built upon the assumption of consistent routing. In this paper, we describe and evaluate novel techniques that achieve consistent routing in connectivity-constrained environments and demonstrate their applicability in representative WOW-based high-throughput computing environments.

Studies have shown that about 30%–40% [20] of the nodes in a P2P system are behind NATs. Certain kinds of NAT devices can be "traversed" to support bi-directional communication links through UDP-hole punching, a technique which has been implemented in systems such as IPOP and is known to work for a large variety of "cone" NATs. However, certain scenarios often arise where hole-punching is not possible, such as in "symmetric" NATs. In addition, studies have also shown the existence of permanent or transient route outages between pairs of nodes on the Internet; for example, [12] reports 5.2% pair-wise outages among nodes on PlanetLab [8], and an experiment described in this paper reveals 9 broken structured connections in a 420-node P2P ring overlaid on PlanetLab. 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 hinder the correctness of structured routing. As applied to WOWs, they hinder the ability of IPOP to provide all-to-all connectivity among nodes that form a virtual cluster.

Related work has addressed challenges including construction of efficient overlay topologies [17], correct routing of object lookups under churn [5, 27], and proximity-aware routing [6]. However, an implicit underlying assumption common in previous work is of an environment where P2P nodes are able to establish direct connections to one another. Deployments of these systems have recognized the problem of overlay structure maintenance when only a small fraction of pairs (about 4% [19]) cannot communicate with each other [12, 16]. However, from our practical experience with WOW deployments, we have observed 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. To illustrate the negative impact of pair-wise connectivity constraints, results from a simulation-based analysis show that the all-to-all routability of a 1000-node ring structured overlay using a conventional (greedy) structured routing algorithm is less than 90% when there is a 70% chance that P2P nodes will be able to communicate directly using TCP or UDP transports, and less than 95% when 70% of the nodes are behind cone and symmetric NATs.

This paper makes the following contributions. We 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 over common neighbors. These techniques are fully decentralized, self-configuring and generally applicable to any structured P2P system. An implementation of these techniques in IPOP has been demonstrated to operate 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. We demonstrate these techniques significantly improve all-to-all routability (with respect to conventional greedy routing) from 90% to over 99% at 70% probability of successful pair-wise connections. We also simulated a 1000 node network consisting of varying fraction of public nodes and well-studied distributions of NAT types. We observed that when 70% of the nodes are behind different types of NATs (including non-traversable NATs), the described techniques improved routability from less than 95% to over 99%. We also report on results from realistic NAT-constrained environments through experiments which demonstrate the benefit of annealing routing and tunnel edges to improve connectivity within nodes of a 180-node WOW Condor [21] pool, increasing the fraction of reported worker nodes from 88% to 100% at 81% probability of successful pair-wise connections.

The rest of the paper is organized as follows. In Sect. 2, we overview several sources of connectivity constraints found in wide-area environments for deployment of desktop grids and collaborative environments. In Sect. 3 we qualitatively describe the impact of incorrect routing on the functioning of WOW distributed systems. We then describe a fault-tolerant annealing routing algorithm that can route messages to their correct destinations, even in the presence of missing overlay links (Sect. 4.1). In Sect. 4.2, we describe a technique that allows tunneling of overlay links over connections to other nodes and the implementation of this technique in IPOP. In Sect. 5, we quantify the improvement in structured routing using these two techniques through simulations. We describe our implementation of tunnel edges in the IPOP system in Sect. 6. In Sect. 7, we describe experiments that demonstrate the operation of IPOP in connec-

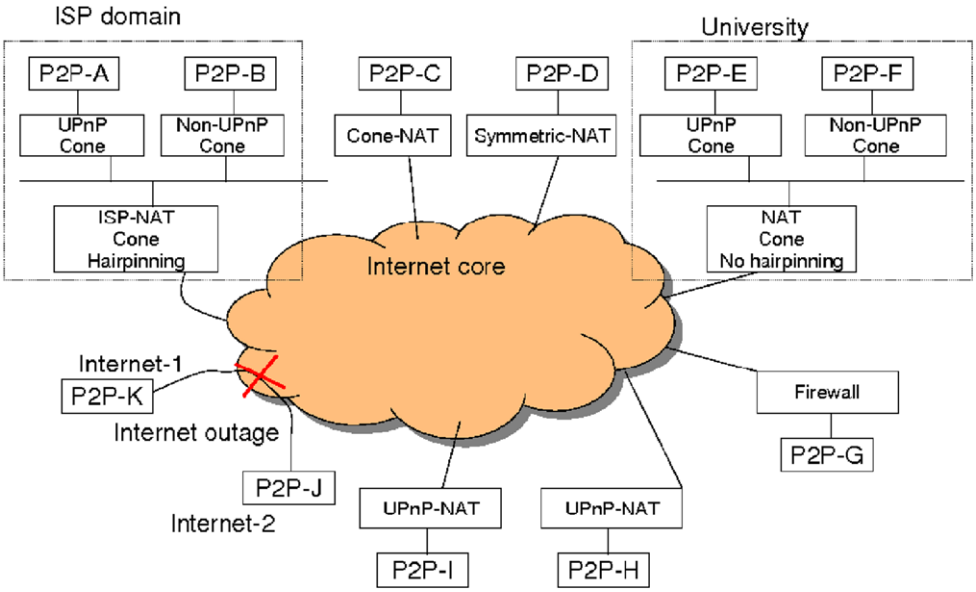

Fig. 1 Illustration of the connectivity-constrained wide-area deployment scenario targeted by deployments of the IPOP P2P system

tivity constrained environments. We discuss related work in Sect. 9, and conclude the paper in Sect. 10.

2 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 [28] 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 Fig. 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.

The scenario which we address in this paper allows the vast majority of P2P nodes to run on hosts that connect to the Internet through one or more levels of NATs. For example, 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 Fig. 1). IPOP supports establishment of UDP communication using hole-punching techniques for "cone" type NATs (e.g. between nodes A and C in Fig. 1), and there is empirical evidence pointing to the fact that these are the common case [10]. Cone NATs consistently map a private IP endpoint to same external IP endpoint, irrespective of the destination. These external IP endpoints can be discovered and subsequently exchanged out-of-band between peers in different private networks for hole-punching.

However, nodes behind "symmetric" 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). Symmetric NATs map a private IP endpoint a different external IP endpoint for each destination. Hole-punching for symmetric NATs rely on the ability to predict the external IP endpoint prior to communication with the destination—such port prediction techniques have not been very effective.

Recognizing the importance of supporting traversal, some of recent NATs have started supporting Universal Plug and Play (UPnP) [31] 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. This problem renders the UPnP approach ineffective outside the domain. For example, although hosts A and E in Fig. 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 Fig. 2. Through communication with nodes on the public Internet, nodes E and F can only learn their IP endpoints assigned by the public NAT-P. In case NAT-P does not support hairpinning, these learned endpoints cannot be used for communication between nodes E and F. Only 24% of the NATs tested in [10] 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 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 [12], 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.

We observe 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 our 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 [16], 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 Fig. 1.

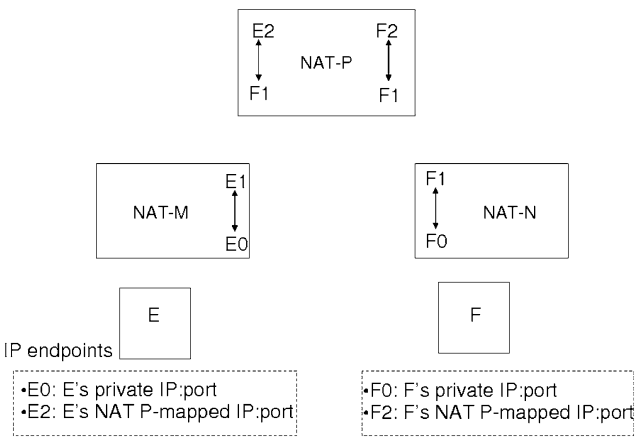

Fig. 2 Nodes E and F behind two different semi-public NATs respectively, which in turn are behind a public NAT-P

3 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 remainder of this section.

3.1 Impact on core structured overlay routing

The IPOP virtual network 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 [18].

Similar to other structured systems, routing in IPOP 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 Fig. 3(a). In this figure, nodes 115 and 110 cannot form a connection. A message is sent to key 112, and the closest node is 110. In one case (a), the message (*Create*) addressed to key 112 arrives at node 115; it believes that it is the closest to the destination and the message is delivered locally (also replicated at node 100).

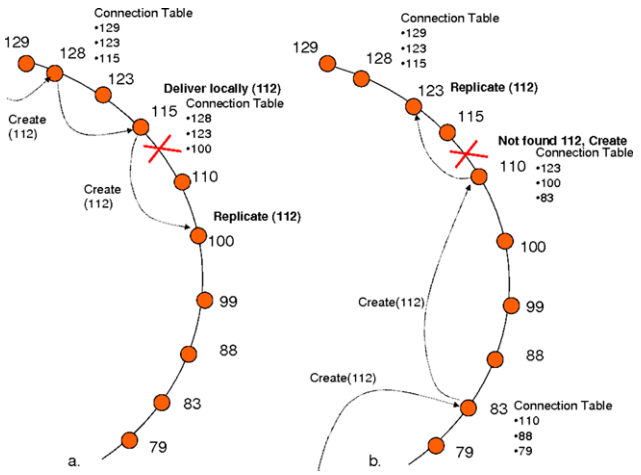

Fig. 3 Inconsistent roots in DHT

In another case (b), a message *Create* addressed to the same key arriving at node 83 is correctly routed to node 110, the key is not found and created again.

3.2 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 [14]. 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.

3.3 Effect on dynamic virtual IP configuration

The structured P2P system in IPOP also provides decentralized object storage and retrieval based on a DHT [15], 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 can cause *Create* messages addressed to the same key from different sources to be routed to different nodes, as shown in Fig. 3(a) and (b). This problem is also identified in [12] and is referred to as inconsistent roots, and can lead to a situation where two WOW nodes claim the same virtual IP address.

3.4 Effect on completion of DHT operations

To reduce the impact of inconsistent roots, the IPOP-DHT internally re-maps each application specified key k to n keys $(k_1, k_2, \ldots, k_n)$, 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 keys 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.

3.5 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.

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 [32], 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 achieve the maximum available throughput because not all nodes can participate in computations.

4 Consistent P2P routing under connectivity constraints

Based on the observations from the previous section, we propose two synergistic approaches to cope with missing

overlay links in a structured P2P network: (1) annealing routing and (2) tunnel edges. These are described in the rest of this section.

4.1 Annealing routing

The first technique we propose 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 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 1 and works as follows.

Lines 1–9 of Algorithm 1 describe the behavior when the message destination matches the current node, or is destined to a node to which the current node is directly connected to. In lines 10–19, 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 21), it finds the two closest nodes to the destination from the connection table, u_{min} and 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_{min}. Otherwise (lines 25–31), until the message has taken MAX_UPHILL hops it is delivered to the closest node u_{min} or the next closest u_{sec} (if it was already received from the closest node u_{min}). Until this point, the algorithm does not check for the forward progress of the message towards the destination in identifier space.

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

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:     Deliver locally.
12:     if (v is to the left of dest) then
13:         w ⇐ v_right (which is on the right of dest)
14:     else
15:         w ⇐ v_left (which is on the left of dest)
16:     end if
17:     if prev ≠ v' then
18:         Send to w (send to other side of dest)
19:     end if
20: else
21:     Find first and second closest nodes u_min and u_sec to dest,
        respectively.
22:     if p.Hops == 0 then
23:         /** Case 2: This is the first hop. Let the packet go to
            closest even if v itself is closest.**/
24:         Send to u_min.
25:     else if p.Hops ≤ MAX_UPHILL then
26:         /** Case 3: Not the first hop. We will do this for up to
            MAX_UPHILL (= 1) hops. **/
27:         if prev ≠ u_min then
28:             Send to u_min.
29:         else
30:             Send to u_sec.
31:         end if
32:     else
33:         /** Case 4: Send only if can get closer than previous
            node.
34:         Find nodes u_min (closest) and u_other (on other side of the
            dest on the ring), respectively.
35:         if prev ≠ u_min then
36:             w ⇐ u_min
37:         else
38:             w ⇐ u_other
39:         end if
40:         d_min ⇐ DIST_ring(w, dest)
41:         if d_min < DIST_ring(prev, dest) then
42:             Send to w.
43:         end if
44:     end if
45: end if
```

Beyond MAX_UPHILL hops (lines 32–45), the node finds the nodes (1) u_{min}: closest to the destination in identifier space, and (2) u_{other}: on the opposite side of u_{min} to the destination. The message is sent to u_{min} or u_{other}, 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 to take one hop that is farther away from the destination than the current node.

Figure 4 illustrates the cases 1 and 4 of the annealing algorithm.

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, this 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 side effect, DHT operations for a key are performed at more than 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. Our next technique is designed to provide a complete overlay structure in face of connectivity constraints that may prevent direct connections among overlay neighbors.

4.2 Tunnel edges

In this section, we describe our 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 Fig. 5. 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. In the IPOP implementation (described in Sect. 6), tunnel edges are functionally equivalent to UDP or TCP edges once they are established, allowing seamless reuse of the code responsible for state maintenance and routing logic in the system.

4.2.1 *Probabilistic analysis of tunnel edges*

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? We address these questions analytically in this section.

Let q be the probability of successful edge setup 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:

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

In Table 1, we show the probability of forming a tunnel edge between unconnected nodes A and B for different values of edge probability q and number of *near* connections m. It should be noted that the 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

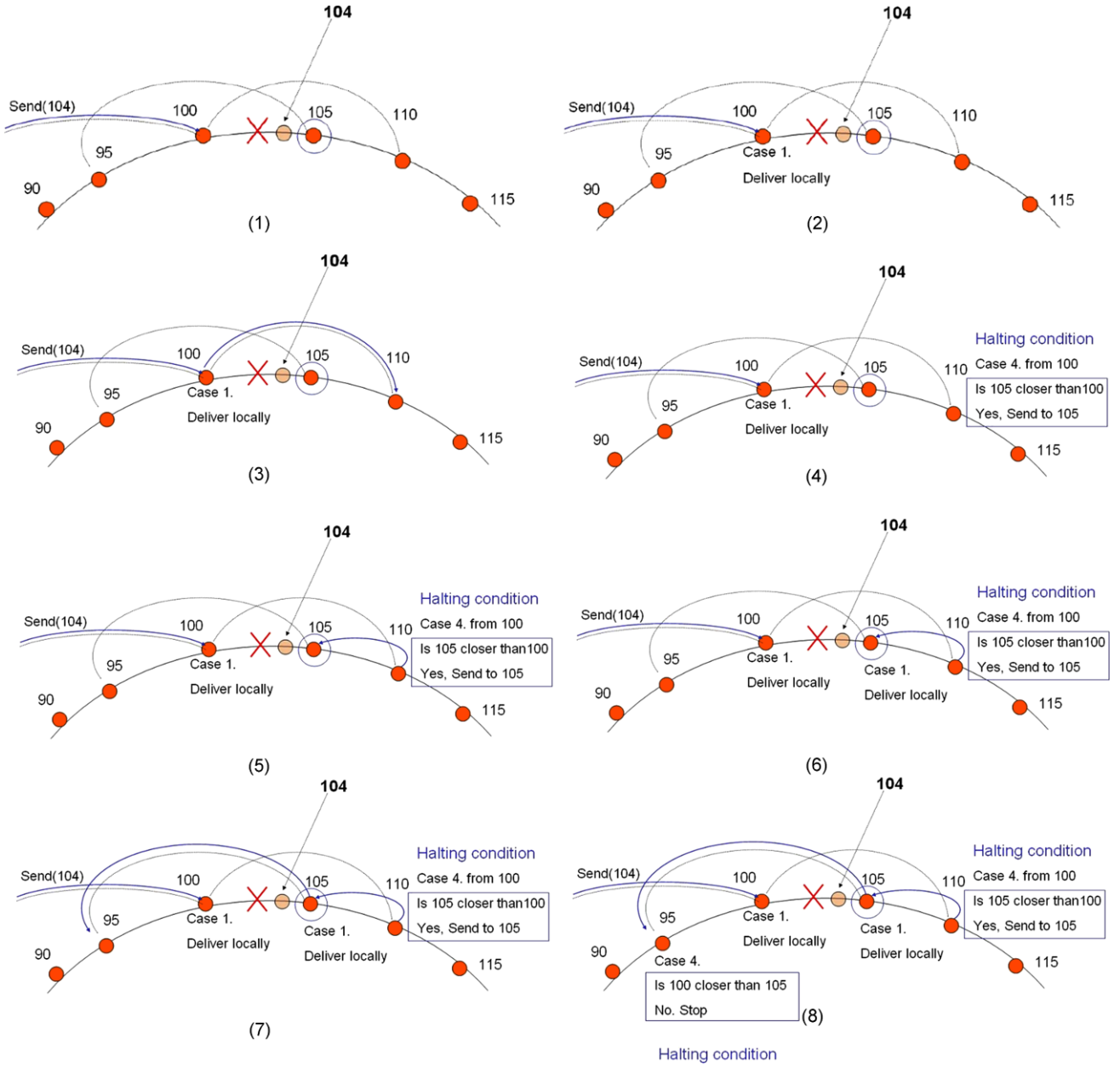

Fig. 4 Illustration of the annealing algorithm. It is assumed that node 100 is unable to create an overlay link with node 105 due to connectivity constraints, as marked by a *red* X in the drawings, and that a message addressed to key 104 is received by node 100. The discussion follows the illustrations from *left* to *right*, *top* to *bottom*. (1) The message addressed to key 104 arrives at node 100. (2) Because node 100's connection table says that it is adjacent to the destination 104, the message is delivered locally. (3) The message is also sent to the node on the other side of the destination: node 110. (4) Node 110 checks for the halting condition (Case 4 of the annealing algorithm); node 105 is indeed closer to the destination than the previous node (100), and hence (5) the message is sent to node 105. (6) Node 105's connection table says it is adjacent to destination 104. Message is delivered locally. (7) Message is sent to the node on the other side, node 95. (8) Node 95 checks the halting condition (annealing algorithm Case 4); its closest node (105) is not closer to destination 104 than previous node 100, hence the message is not forwarded. At the end of the algorithm, both nodes adjacent to node 104 (nodes 100 and 105) are delivered with the message

show that improvements in correctness of routing using tunnel edges are significantly higher when $m \geq 3$. Figure 8 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

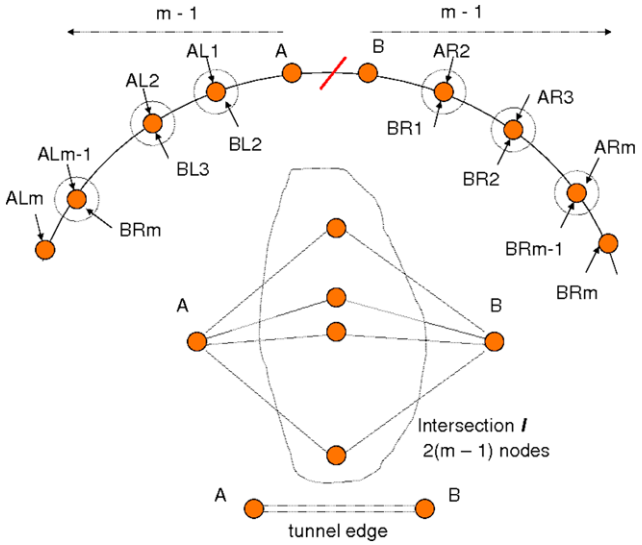

Fig. 5 Tunnel edge between nodes A and B that cannot communicate over TCP or UDP transports

Table 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

Edge prob	Tunnel edge probability			
	$m = 2$	$m = 3$	$m = 4$	$m = 5$
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

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:

$$P[\textit{forwarding set of size atleast } 2] = 1 - \sum_{k=0}^{k=1} P[\textit{forwarding set of size exactly } k] = 1 - \sum_{k=0}^{k=1} \binom{2(m-1)}{k} \cdot (q^2)^k \cdot (1-q^2)^{2(m-1)-k}$$

and the expected size of the forwarding set is given by:

$$E[\textit{expected size of forwarding set}] = \sum_{k=0}^{k=2(m-1)} k \cdot P[\textit{forwarding set of size exactly } k] = \sum_{k=0}^{k=2(m-1)} k \cdot \binom{2(m-1)}{k} \cdot (q^2)^k \cdot (1-q^2)^{2(m-1)-k} = 2(m-1) \cdot q^2$$

For $m = 3$, and $q = 0.9$, 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.

5 Improvements in structured routing

In this section, we quantify 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 IPOP 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 (Fig. 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.

A fault model to capture all such scenarios does not exist in the literature. We investigate the existence of route outages under the following two different models:

1. **Uniform edge likelihood:** Likelihood of an edge between a pair of nodes with a uniform pair-wise edge probability and allow for high probabilities of P2P edges not being able to form—as high as 30%, and
2. **Nodes behind symmetric NATs:** The fraction of public nodes in the network is varied between 70% and 30%. Of the remaining nodes behind NATs, 80% are considered to exist behind cone NATs, while 20% are considered to exist behind symmetric NATs. Nodes behind cone NATs can create direct connections with cone NATed nodes, and also with nodes on Internet. Nodes behind symmetric NATs can only create direct connections with nodes on the public Internet. This distribution is guided by the analysis of different kinds of NATs in [10].

5.1 Simulation methodology

The simulation environment captures the algorithms used in IPOP for structured overlay creation and routing. We create 1000 nodes with randomly generated 160-bit identifiers and model pair-wise outages as follows.

We simulate the uniform edge likelihood model with a configurable probability q. Based on the probability q of any pair of nodes being able to communicate using TCP or UDP, we create a connection matrix that allows/disallows connections between pairs of nodes. To model nodes behind Symmetric NATs, we create a connection matrix that allows all types of connections except the ones involving nodes behind symmetric NATs and other NATed nodes. We then add connections to nodes in the following steps:

1. At each node, attempt to add *near* connections to the immediate m neighbors (on each 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. We then add one *far* connection at each node (that is allowed by the connection matrix). The distances traveled by these connections in the structured ring follow the distribution described in [18].

To study the all-to-all routability of the network, we simulate the sending of a message between each pair of nodes, and count the number of times the message is incorrectly delivered. We conduct this experiment for 200 different randomly generated graphs. To investigate correct routing of keys, we randomly generate 10000 different keys. For each key, we 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 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. We observe 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. Furthermore, keeping more *near* connections at each node only marginally improves the network routability. Similar observations are also made for routing of keys (see Fig. 7)—there is more than 10% chance that a key is wrongly routed as the edge likelihood drops to 70%.

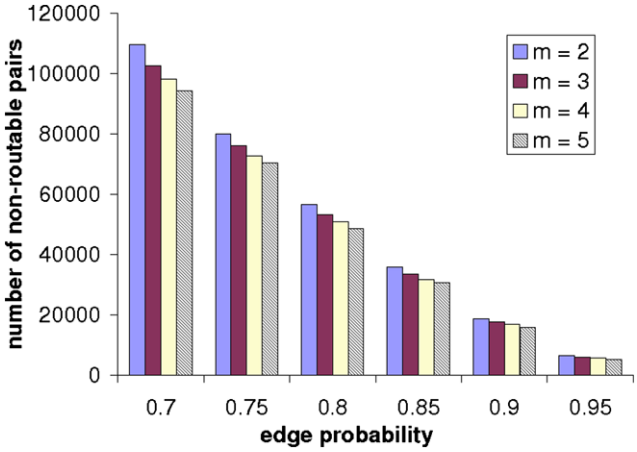

Fig. 6 At edge likelihood of 70% (0.7), the percentage of non-routable pairs varies from 9.5% to 10.9% (the total number of pairs in the simulated network is 1,000,000)

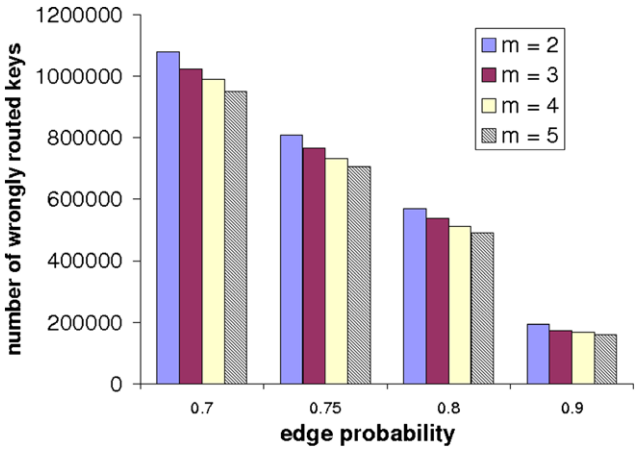

Fig. 7 At edge likelihood of 70%, the percentage of wrongly routed keys varies from 9.5% to 10.7% (the total number of simulated messages is 10,000,000)

5.2 Evaluating the impact of annealing routing

5.2.1 Uniform edge likelihood

In Fig. 9, we show the reduction in average number of non-routable pairs using Algorithm 1 with $m = 3$; tunnel edges are not enabled. We observe that, at an edge 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 these networks of 1000 nodes.

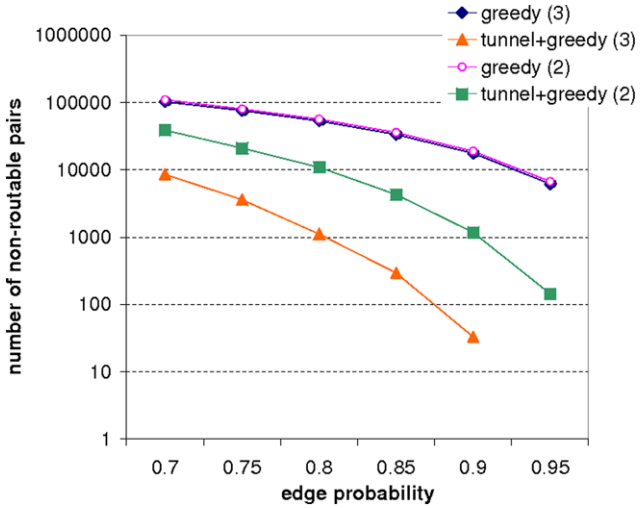

Fig. 8 Comparing greedy routing with tunnel edges for $m = 3$ and $m = 2$. At edge likelihood of 70%, the percentage of non-routable pairs in a network of 1000 nodes is (1) 3.9% for $m = 2$, and (2) 0.86% for $m = 3$

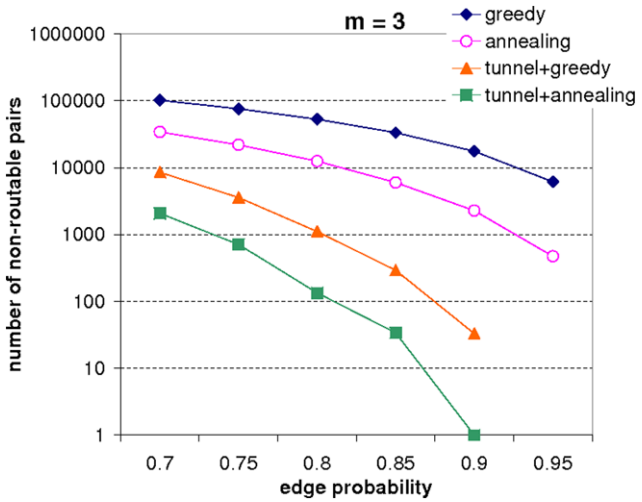

Fig. 9 Average number of non-routable pairs. At edge likelihood of 70%, the percentage of non-routable pairs for greedy and annealing routing is (1) without tunnel edges, 10.26% and 3.4% respectively; (2) with tunnel edges, 0.86% and 0.21% respectively. At edge likelihoods of 95%, there are no non-routable pairs with tunnel edges

We measured the average number of hops taken by a message for both greedy and annealing routing for these cases. 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.

In Fig. 10, we show the average number of wrongly routed key lookups for both annealing and greedy routing

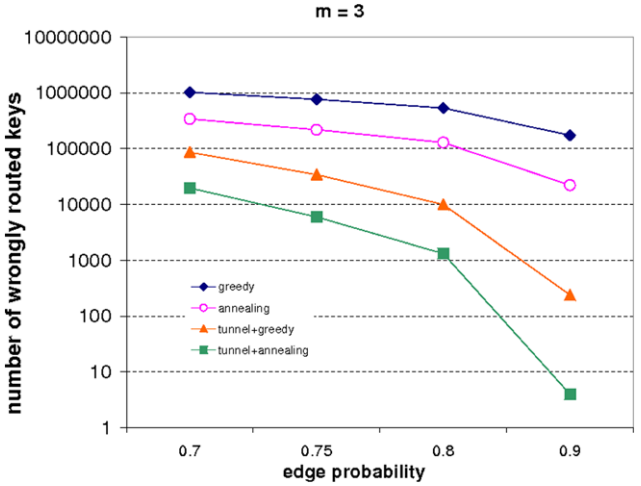

Fig. 10 Average number of wrongly routed keys. At edge likelihood of 70%, the percentage of wrongly routed keys for greedy and annealing routing is (1) without tunnel edges, 10.2% and 3.4% respectively; (2) with tunnel edges, 0.86% and 0.19% respectively

using the methodology as described in Sect. 5.1. We observe 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 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%.

5.2.2 *Nodes behind symmetric NATs*

Figure 11 shows results for the scenario where 20% of simulated NATed nodes are behind symmetric NATs. We observe that even when the number of public nodes drops to less than 30%, the all-to-all routability of the network using annealing routing exceeds 98.12% (as opposed to 94.4% with greedy routing), which is about a 66% reduction in the number of non-routable pairs to that of greedy routing.

5.3 Evaluating the impact of tunnel edges

5.3.1 *Uniform edge likelihood*

Figure 9 shows how the enhanced overlay structure because of tunnel edges can improve the all-to-all routability of the network, for $m = 3$. We observed that even 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. We also recorded the actual number of

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

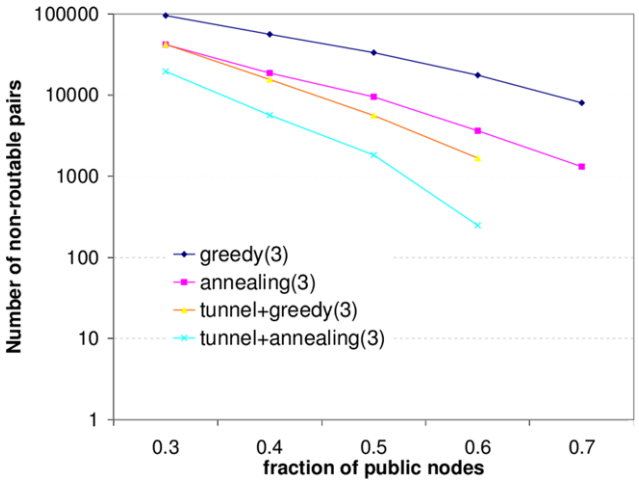

Fig. 11 Average number of non-routable pairs in a scenario where 20% of NATed nodes are behind symmetric NATs. When 70% of nodes are behind NATs, the percentage of non-routable pairs for greedy and annealing routing is (1) without tunnel edges, 5.6% and 1.88% respectively; (2) with tunnel edges, 1.6% and 0.57% respectively. When 70% of nodes are public, there are no non-routable pairs with tunnel edges

hops taken by messages addressed to exact destinations in an overlay that supported tunnel edges. 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 10 also compares how tunnel edges improve the consistency of key routability of the network, for $m = 3$. We 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).

5.3.2 Nodes behind symmetric NATs

Figure 11 shows the improvements in all-to-all routability of the network when the number of public nodes is reduced from 70% to 30%. Even when only 30% of the nodes are public, tunnel edges improve the percentage of non-routable pairs from 1.88% to 0.57% for annealing routing (from 5.6% to 1.6% for greedy routing).

6 Tunnel edge implementation in IPOP

The IPOP P2P system provides extensive support for creating overlay links between nodes over a variety of transports and incorporates decentralized UDP hole-punching. To implement tunnel edges, we extend the existing mechanisms for connection setup to discover suitable proxy nodes for tunnel edges, when direct communication is not possible.

Connection setup between P2P nodes is preceded by a *connection protocol* [14] 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. In the original IPOP system, the *connection protocol* allows nodes to exchange their NAT-assigned IP address/port. In this paper, we use the same mechanism to also exchange information about their connections to near neighbors.

6.1 TunnelEdgeListener

As described earlier, each connection in IPOP 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. An IPOP node can have more than one EdgeListener, and new types can be easily added.

Before we further describe the process of creating a tunnel edge, we overview the functionality that allows each IPOP 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 IPOP node to identify its left and right neighbors in the P2P ring [14].

To enable connections between nodes A and B to be proxied by common neighbors, we have implemented an EdgeListener called TunnelEdgeListener. The TunnelEdgeListener is invoked together with the TCP and UDP EdgeListeners during the process of a connection setup. 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 can compute the forwarding set F which is the intersection of its own connections with those listed in the tunnel URI of B.

Having computed the forwarding set F with remote node B, the node A 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.

Our implementation does not require the nodes in the forwarding set 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.

7 Experiments

In this section, we demonstrate the ability of our 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. We also observe the time it takes 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. Finally, we also study the impact of using annealing routing and tunnel edges on connectivity within a WOW when P2P nodes only had 81% chance of being able to setup connections.

7.1 Structure verification of P2P network

To verify the completeness of the P2P ring, we iteratively "crawl" the IPOP network using the immediate right neighbor information at each node. We check the consistency of its connections with respect to its predecessor. Specifically, for every node, we check if its immediate left *near* connection node identifies it as immediate right *right* 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 that we observe a 100% consistent ring with a few nodes completely missing. These hidden nodes can be detected using information logged by IPOP at each node; knowledge of the number of nodes and their identifiers is also available.

We demonstrate the effect of our techniques with respect to overlay structure, in both a synthetic environment where we artificially create situations where connection setup is not always possible, and also in a large-scale PlanetLab environment, which is known to exhibit route outages between pairs of hosts.

7.1.1 NATed environment

To demonstrate the ability of IPOP to deal with heavily-NATed environments, we deployed a P2P network of 1030 nodes involving 12 private networks (each containing 80 P2P nodes) behind port-restricted cone NATs, and a seed network of 70 P2P nodes that was reachable from all other P2P nodes.

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.

7.1.2 Incomplete underlying network

In this experiment, we built a network of 711 P2P nodes incrementally. Starting with a seed network of 71 nodes, we bootstrapped another 640 P2P nodes on 16 hosts (each running 40 P2P nodes). Each P2P node was configured to use a unique pre-defined UDP port number. Using IPtables, we configured the firewall rules on the hosts to drop UDP packets such that the probability of setting up UDP-based connection between any pair of P2P nodes was 0.95.

Once the P2P structure was formed, we observed there were 35 pairs of adjacent nodes on the P2P ring 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.

7.1.3 Wide-area deployment

In this experiment, we deployed a network of over 420 nodes on PlanetLab with distinct hosts in North America, South America, Asia, Europe and Australia. We observed that as many as 9 adjacent pairs on P2P ring (in North America, Europe and Asia) could not communicate using TCP or UDP

transports. Their inability to connect, which we 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.

To further evaluate the ability of tunnel edges to form, we deployed 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. We observed two instances 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.

We measure 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 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. Our 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, we also observed cases where 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. It does not observe a connection overlap with its neighbor node (with which it has exchanged *Connect To Me* messages) to successfully create a tunnel edge. Similarly, the neighbor node also observes an empty tunnel URI for the new node, and hence cannot create a tunnel edge. The first linkers that are created at these nodes 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.

We have now tuned our implementation of *TunnelEdgeListener*, such that instead of failing immediately when an overlap of connections does not exist with a valid (non-empty) tunnel URI of remote node, it retries after some interval (multiple times) to check if an overlap is created. In the meantime, the new node can acquire connections to its other close neighbors that overlap with those listed in the tunnel URI of its closest neighbor, and tunnel edge creation can succeed. With this enhancement, there is a greater chance for tunnel edge creation to succeed through the first linker that is created at the new node. By not waiting for subsequent linkers to create a tunnel edge, we have been able to reduce the average time to get connected with neighbors from 41 seconds to less than 20 seconds.

7.2 Connectivity within a WOW

In this section, we study the impact of using annealing routing and tunnel edges with respect to improvements in connectivity within a WOW-based Condor pool. Using a bootstrap infrastructure of 20 P2P nodes, we created a WOW consisting of 180 Condor worker nodes and 1 manager. We measure the number of workers reported by the manager (using the *condor_status* command), 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. We also report on the all-to-all connectivity between worker nodes. We send 30 ICMP ping messages from each worker to every other worker, and counted a 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, we configured the UdpEdgeListener at each node 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: $(1 - 0.10)^2 = 0.90^2 = 0.81$.

We then configured the P2P nodes 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, we configured the P2P nodes to use annealing routing but no tunnel edges. The Condor manager reported 177 worker nodes, and there were 859 pairs of workers that could not communicate with each other.

Finally, we configured the P2P nodes to use both annealing routing and tunnel edges. 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. We observed only one inconsistency 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 our 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.

8 Discussion

Our 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 full cone-NATs).

Consider a scenario where we start at a perfectly formed ring of P2P nodes, and there exists a sequence of $2K$ consecutive nodes $(n_i, n_{i+1}, \ldots, n_{i+2K-1}$, K is the number of connections each node keeps on its either side) on the P2P ring that are behind cone NATs. Now a new node N_{new} is bootstrapped with an identifier that lies within this range of nodes (i.e. between n_{i+K-1} and n_{i+K}). Node N_{new} cannot establish a direct connection with any of its K neighbors on either side, as a result an overlap of connections with its neighbors does not exist.[3] Connection setup protocols will hence fail with its immediate neighbors and one of more inconsistencies will appear in the P2P ring.

We now present a protocol for creating tunnel edges that incorporates detection of such cases where a "passive" overlap does not exist. The approach is to discover public nodes in the network, create connections with common sets of such nodes, and then use these connections to form tunnel edges.

8.1 Active tunnel edge creation protocol

A tunnel edge involving a node behind NAT and another node behind a symmetric NAT has to involve a forwarding set consisting of public nodes. Therefore, the process of tunnel edge creation is preceded by an additional step for discovering common public nodes, and both nodes creating connections to them. Each new node in the network is already initialized with URIs of a few public nodes in the seed network. It is possible to use these nodes as the forwarding set for creating tunnel edges. However, in typical WOW deployments, a common configuration file listing the nodes in the seed network is available at each node. If all nodes start using these bootstrap nodes for tunnel edge forwarding, as the WOW grows, tunnel edge forwarding can impose a large load on the bootstrap nodes even if there are other WOW nodes that may be public. Therefore, a mechanism is required to discover other public nodes in the network. The next section describes an efficient mechanism to run queries on the structured P2P network.

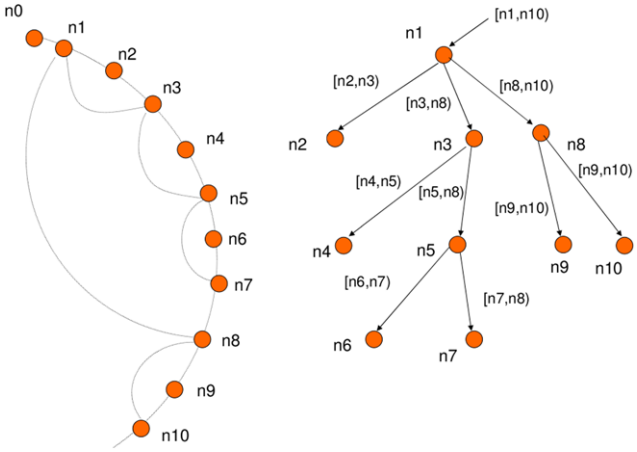

Fig. 12 Illustration of the use of bounded broadcast over a segment of the P2P ring to search for public nodes. *Left*: a segment of the P2P ring. *Right*: the resulting tree for a bounded broadcast on $[n1, n10)$

8.1.1 Discovering public nodes

In the simplest case, it is possible to traverse the P2P ring sequentially until a public node is discovered. For a network of size n, this search runs in $O(n)$ time and is inefficient for large networks. It is possible to carry out the discovery (in sub-linear time) as follows. The search algorithm builds a tree consisting of P2P nodes within a segment of the P2P ring $[A, B)$, starting at the current node A (see Fig. 12). The query is propagated down the tree in parallel all the way to leaf nodes. Each node computes the local result (expressed as a *map* function), and waits for the each child sub-tree to return results. Leaf nodes immediately return their local results. As the results propagate up the tree, an aggregation (expressed as a *reduce* function) is performed at each intermediate node. 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 $c_1, c_2, c_3, \ldots, c_m$. 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. It is possible to terminate

[3] It is possible that the new node can create tunnel edges with nodes at the edge of this range, but since our implementation does not support recursive tunneling, this overlap is not useful for creating a tunnel edge with immediate neighbors, n_{i+K-1} and n_{i+K}.

the query propagation earlier up in the tree if certain criteria are met, expressed in *map* and *reduce* functions. Instead of searching the entire ring, a node initially queries a small segment of the ring, starting at itself. If the search fails, it tries the next segment twice as big. This process is repeated until the entire ring is searched or a set threshold is reached.

To discover public nodes, the *map* function returns a list containing the local node's P2P identifier (if it is a public node), or an empty list otherwise. The *reduce* function simply performs list concatenation of child results. It is also possible to include other criteria such as the CPU load on the host or network proximity to discover a forwarding nodes through appropriately chosen *map* and *reduce* functions.

8.1.2 Tunnel edge creation

Once both nodes discover a set of public nodes, they can exchange this information and create common connections to these nodes. Both nodes create connections to common nodes. These common nodes form the forwarding set of the tunnel edge, and the *EdgeRequest/EdgeResponse* protocol, as described in Sect. 6, can then work over this forwarding set.

9 Related work

In [23], 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. Our 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, our approach is self-configuring and fully decentralized.

Structured P2P systems (Chord [30], Pastry [29], Bamboo [3], Kademlia [24]) have primarily focused on efficient overlay topologies [17], reliable routing under churn [5, 27], and improving latency of lookups through proximity-aware routing [6]. In [12, 16], 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, our focus is to enable overlay structure maintenance when a large majority of nodes are behind NATs, and several scenarios hinder communication between nodes. The techniques described in this paper facilitate correct structured routing, even when many (up to 30%) pairs of nodes cannot communicate directly using TCP or UDP.

In [25], 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 [26], in the context of a P2P-based email system built on top of Pastry. Our work, on the other hand, uses tunneling to improve all-to-all virtual-IP connectivity between WOW nodes. We also quantify the impact of the described techniques on structured routing through simulations, under different edge probabilities between nodes. Unmanaged Internet Protocol (UIP) [9] 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, our focus is to facilitate IP communication between Grid resources in different "managed" Internet domains.

In [7], 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 paper can benefit the implementation of the strongly consistent KBR. Similarly, [2] 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, our work also attempts to correct the overlay structure in presence of link failures.

10 Conclusion and future work

In this work, we describe and evaluate 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. Furthermore, when only 30% of the nodes in a 1000 node network are public, the described techniques improve all-to-all routability of the network from less than 95% to more than 99%. 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 IPOP 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 im-

prove the virtual-IP connectivity within a WOW-based condor pool experimentally.

References

1. Anderson, D.P., Cobb, J., Korpella, E., Lebofsky, M., Werthimer, D.: Seti@home: an experiment in public-resource computing. Commun. ACM **11**(45), 56–61 (2002)
2. Aspnes, J., Diamadi, Z., Shah, G.: Fault-tolerant routing in peer-to-peer systems. In: Proc. of the Symp. on Principles of Distributed Computing (PODC), Monterey, CA, July 2002
3. The bamboo dht—introduction. http://bamboo-dht.org/
4. Calder, B., Chien, A.A., Wang, J., Yang, D.: The entropia virtual machine for desktop grids. In: CSE Technical Report CS2003-0773, University of California, San Diego, San Diego, CA, Oct. 2003
5. Castro, M., Costa, M., Rowstron, A.: Performance and dependability of structured peer-to-peer overlays. In: Proc. of the Conf. on Dependable Systems and Networks, June 2004
6. Castro, M., Druschel, P., Hu, Y.C., Rowstron, A.: Topology-aware routing in structured peer-to-peer overlay networks. In: Microsoft Research MSR-TR-2002-82, Sep. 2002
7. Chen, W., Liu, X.: Enforcing routing consistency in structured peer-to-peer overlays: should we and could we? In: Proc. of the Workshop on Peer-to-Peer Systems (IPTPS), Santa Barbara, CA, Feb. 2006
8. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. ACM SIGCOMM Comput. Commun. Rev. **33**(3), 3–12 (2003)
9. Ford, B.: Unmanaged Internet Protocol: taming the edge network management crisis. In: Proc. of the Workshop on Hot Topics in Networks (HotNets), Cambridge, MA, Nov. 2003
10. Ford, B., Srisuresh, P., Kegel, D.: Peer-to-peer communication across network address translators. In: Proc. of the USENIX Annual Technical Conference, Anaheim, California, April 2005
11. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: enabling scalable virtual organizations. Int. J. Supercomput. Appl. **15**(3), 200–222 (2001)
12. Freedman, M.J., Lakshminarayanan, K., Rhea, S., Stoica, I.: Non-transitive connectivity and DHTs. In: Proc. of the USENIX WORLDS, San Francisco, CA, Dec. 2005
13. Ganguly, A., Agrawal, A., Boykin, P.O., Figueiredo, R.J.: IP over P2P: enabling self-configuring virtual IP networks for grid computing. In: Proc. of Intl. Parallel and Distributed Processing Symp. (IPDPS), Rhodes, Greece, April 2006
14. Ganguly, A., Agrawal, A., Boykin, P.O., Figueiredo, R.J.: Wow: self-organizing wide area overlay networks of virtual workstations. In: Proc. of Intl. Symp. on High Performance Distributed Computing, Paris, France, June 2006
15. Ganguly, A., Wolinsky, D., Boykin, P.O., Figueiredo, R.J.: Decentralized dynamic host configuration in wide-area overlays of virtual workstations. In: Proc. of the Workshop on Desktop Grids and Volunteer Computing Systems, with IPDPS, Long Beach, CA, March 2006
16. Gerding, S., Stribling, J.: Examining the trade-offs of structured overlays in dynamic non-transitive network, Dec. 2003. http://pdos.lcs.mit.edu/~strib/doc/networking_fall2003.ps
17. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In: Proc. of ACM SIGCOMM, Karlsruhe, Germany, Aug. 2003
18. Kleinberg, J.: Nature **406**, 845 (2000)
19. Li, J., Stribling, J., Morris, R., Kaashoek, M.F., Gil, T.M.: A performance vs. cost framework for evaluating DHT design tradeoffs under churn. In: Proc. IEEE INFOCOM, 2005
20. Liang, J., Kumar, R., Ross, K.: The fasttrack overlay: a measurement study. In: Computer Networks (Special Issue on Overlays), 2005
21. Litzkow, M., Livny, M., Mutka, M.: Condor—a hunter of idle workstations. In: Proc. of Intl. Conference on Distributed Computing Systems, June 1988
22. Lo, V., Zappala, D., Zhou, D., Liu, Y., Zhao, S.: 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
23. Maassen, J., Bal, H.E.: Smartsockets: solving the connectivity problems in grid computing. In: Proc. of Symp. on High Performance Distributed Computing Symposium, Monterey Bay, CA, June 2007
24. Maymounkov, P., Mazières, D.: 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, March 2002
25. Mislove, A., Druschel, P.: Providing administrative control and autonomy in structured peer-to-peer overlays. In: Proc. of the Workshop on Peer-to-Peer Systems, San Diego, CA, Feb. 2004
26. Mislove, A., Post, A., Haeberlen, A., Druschel, P.: Experiences in building and operating epost, a reliable peer-to-peer application. In: Proc. of European Conf. on Computer Systems, Leuven, Belgium, April 2006
27. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a DHT. In: Proc. of USENIX Technical Conference, June 2004
28. Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., Opendht, H.Yu.: A public DHT service and its uses. In: Proc. of ACM SIGCOMM, Philadelphia, PA, Aug. 2005
29. Rowstron, A., Druschel, P.: Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: Proc. of the Intl. Conf. on Distributed Systems Platforms (Middleware), Heidelberg, Germany, Nov. 2001
30. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. **11**(1), 17–32 (2003)
31. Universal plug and play in windows xp. http://technet.microsoft.com/en-us/library/bb457049.aspx
32. Wolinsky, D., Agrawal, A., Boykin, P.O., Davis, J., Ganguly, A., Paramygin, V., Sheng, P., Figueiredo, R.: On the design of virtual machine sandboxes for distributed computing in wide area overlays of virtual workstations. In: Proc. of the Workshop on Virtualization Technology in Distributed Computing, with Supercomputing, Tampa, FL, Nov. 2006

Arijit Ganguly received his Ph.D. in Computer Science from University of Florida in 2008, and undergraduate degree (B.Tech.) in Computer Science from Indian Institute of Technology, Guwahati (India) in the year 2002. His research interests include Grid Computing, P2P systems, autonomic computing, virtual machines and networks. He currently works as a Software Development Engineer for the Elastic Compute Cloud (EC2) at Amazon.com in Seattle. Previously, Arijit did summer internships at VMware, IBM Research and Microsoft in 2005, 2006 and 2007, respectively.

P. Oscar Boykin is an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Florida. He received his Ph.D. in Physics from the University of California at Los Angeles. His research interests include complex networks, P2P networking, Grid Computing, and Quantum Information Theory.

David I. Wolinsky received his Bachelors of Science in Computer Engineering in 2005 followed by a Masters of Science in Electrical Engineering in 2007 both from the University of Florida. He is currently working towards a Doctor of Philosophy in Electrical Engineering. He has had the privilege of working on projects that include the Grid Appliance, IPOP, Brunet, and SocialVPN. His current research interests are in distributed security and data structures.

Renato J. Figueiredo is an Associate Professor at the Department of Electrical and Computer Engineering of the University of Florida. Dr. Figueiredo received the B.S. and M.S. degrees in Electrical Engineering from the Universidade de Campinas in 1994 and 1995, respectively, and the Ph.D. degree in Electrical and Computer Engineering from Purdue University in 2001. From 2001 until 2002 he was on the faculty of the School of Electrical and Computer Engineering of Northwestern University at Evanston, Illinois. His research interests are in the areas of virtualization, distributed systems, overlay networks, computer architecture, and operating systems.