PAPER Special Section on Networking Technologies for Overlay Networks

A Dynamic Topology and Routing Management Strategy for Virtual IP Networks

Norihito FUJITA[†a)], ***Member***, **Joseph D. TOUCH**[††], **Venkata PINGALI**[††], ***and*** **Yu-Shun WANG**[††], ***Nonmembers***

SUMMARY This paper describes an architecture for deploying virtual IP networks with P2P-like dynamic topology and routing management. Existing virtual IP network deployment mechanisms do not allow for dynamic topology adaptation and fault-tolerance because provisioning of IP tunnels is performed only once—when a virtual network is deployed. We propose a P2P-XBone, in which a P2P protocol such as DHT drives the topology and the routing table of a virtual IP network consistent with its neighbor node state. We describe how to extend both the existing X-Bone system and P2P mechanisms to achieve interworking between them. The P2P-XBone not only provides P2P's characteristics such as self-organization, fault-tolerance and content-based routing to virtual IP networks but also provides higher forwarding performance and simpler implementation to P2P systems due to the support for the use of existing network services. We also show several results on the evaluation of overhead of P2P-driven provisioning and on forwarding performance.

key words: *virtual networks, peer-to-peer, DHT, dynamic topology control, routing*

1. Introduction

A Virtual Internet (VI) [1] is a kind of overlay network in which an virtual IP network infrastructure is created over an existing IP network. The VI provides all of available IP network capabilities, which can be used by any application that rides on the VI. As a tool to deploy and manage VIs, we are working on the X-Bone [2], which can deploy VIs with hosts and routers logically emulated in physical nodes. It can also create any virtual IP network topology by connecting them by IP tunnels such as IP-in-IP. Existing VI deployment systems including the X-Bone provide static virtual topology and perform provisioning of IP tunnels only once, at the time of deployment of a VI. Dynamic control of a VI is useful to support dynamic node join/departure and state changes in a base network. Although existing VIs support dynamic routing changes using routing protocols such as OSPF, they do not have any mechanism for dynamic node addition/deletion as physical IP networks do. For example, when a node in a VI loses all links to neighbor nodes due to node failure, it cannot communicate with all other nodes without dynamic provisioning of a new tunnel to appropriately selected other nodes. Achieving both dynamic provisioning as well as dynamic routing are important to enable use of a VI for ad-hoc group communication.

The features provided by peer-to-peer (P2P) systems are attractive to supplement what existing VIs are missing because P2P natively supports self-organization and fault-tolerance for dynamic addition or deletion of nodes. P2P networks, unlike regular networks, achieve these promising properties by modifying the topology to reflect the routing changes. Existing P2P protocols [3]–[5] and platforms [6], [7] run at the application layer and use UDP or TCP to interconnect P2P nodes. Therefore, they change only the application-level logical topology over an existing IP network. In a VI, however, because the IP-level topology has to be modified by establishing/releasing IP tunnels (i.e., provisioning), such dynamic topology management cannot be applied in the same way as in the existing P2P systems. Existing P2P systems, by contrast, have performance drawbacks such as low throughput and high latency due to application-level routing [8]. We are motivated to provide a system that has both advantages of VI and P2P and supplements each other's disadvantages.

In this paper, we propose a P2P-XBone that enables a VI to obtain the P2P's attractive properties by achieving P2P-driven dynamic provisioning. In the P2P-XBone, the X-Bone system is extended to have an interface for a user-level daemon running a P2P protocol to control an underlying VI. Also, the P2P protocol is extended to explicitly request IP tunnel creation/release and routing table configuration via a configuration interface. Driven by the extended P2P protocol, IP tunnels are dynamically created/released based on neighbor node changes in the P2P protocol, and a routing table is configured based on a routing rule to the neighbor nodes, which results in a direct and complete mapping of P2P topology and routing from the application layer to the VI layer. This extension cannot be achieved in a straightforward way because the P2P protocol runs over the VI, which is itself controlled by the protocol. The P2P protocol cannot exchange parameters outside of the VI even though it needs to know parameters of the underlying base network to control the VI. We solve this problem without violating the VI's virtualization boundary by introducing a P2P control message that carries an opaque parameter of the base network. The P2P protocol provides the unique property of routing-driven provisioning to a VI due to its ability to control IP tunnels as well as the routing table, even though, traditionally, routing and provisioning have been considered as being independent. We call such a P2P-driven VI simply a P2P-VI.

Manuscript received December 29, 2005.
Manuscript revised April 5, 2006.

[†]The author is with the System Platforms Research Laboratories, NEC Corporation, Kawasaki-shi, 211-8666 Japan.
[††]The authors are with the University of Southern California/Information Sciences Institute, USA.
a) E-mail: n-fujita@bk.jp.nec.com

DOI: 10.1093/ietcom/e89–b.9.2375

The P2P-XBone not only provides P2P's high resilience to a VI but also enables a VI to achieve P2P's unique content-based routing. It is difficult for a normal VI to support data-based forwarding as P2P systems do. P2P systems rely on non-IP-address-based IDs such as URLs and hash values as destinations. The P2P-XBone addresses this by applying the DataRouter string-based forwarding module [9].

Thus the P2P-VI can be an alternative P2P infrastructure achieved at the virtual IP layer. Deploying P2P systems at the virtual IP layer provides several advantages compared with conventional application-level deployment. First, higher forwarding performance can be obtained because forwarding is performed in the kernel, avoiding data copying across the user-kernel boundary. Next, the implementation of P2P applications can be simplified because such fundamental mechanisms as end-to-end reliability and security can be achieved using such existing network services as TCP and IPsec without re-implementing them in the application layer.

The remainder of this paper describes the extensions both to existing P2P protocols and to the existing X-Bone that are required for the P2P-XBone. It also shows the overhead of the routing-driven provisioning and improvement in forwarding performance through simulation and experiments of the implemented system.

2. P2P-XBone

The P2P-XBone is a system to deploy a VI that works in a P2P manner (i.e., P2P-VI). The high-level architecture is shown in Fig. 1. The P2P-XBone is composed of P2P Daemon (P2PD), Resource Daemon (RD) and Overlay Manager (OM), where RD and OM are extended from those of the existing X-Bone. The RD runs on each node participating in a VI (virtual node) and is responsible for configuring IP tunnels and routing tables on the virtual node. In the P2P-XBone, the RD is extended to let these configuration changes be triggered by the P2PD. The details are described in Sect. 2.1. The P2PD is a daemon running a P2P protocol, which has the functionality of sending control messages to the RD to configure a VI corresponding to its neighbor nodes and routing table entries. As many P2PDs as the number of internally emulated virtual nodes are run on a physical node. The P2PD is not necessarily developed from scratch but can easily be extended from existing P2P software without losing those original properties. How to extend DHT protocols will be described in Sect. 2.2.

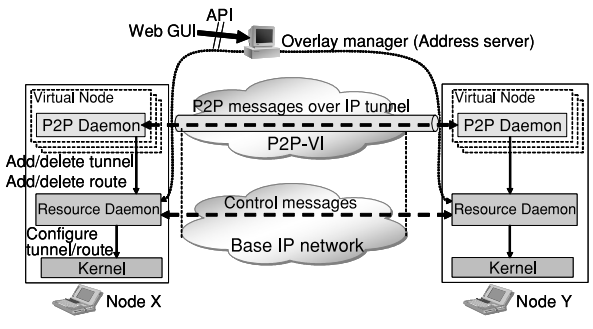


Fig. 1 High-level architecture of the P2P-XBone.

While both RD and P2PD run on nodes participating in a P2P-VI, the OM works outside the nodes to provide functions that are necessary to coordinate a P2P-VI. When deploying a new P2P-VI, an administrator defines the initial topology, node requirements, and services via the API provided by the OM or a Web-GUI to that API. The OM maintains the defined P2P-VI with a set of associated configuration parameters such as an IP tunneling protocol and port number on which RD listens for P2PD. When a new virtual node asks to join the P2P-VI, the configuration parameters are sent to the RD for the virtual node. Whereas one OM in the existing X-Bone manages all parameters on a VI including virtual interfaces and routing entries on the virtual nodes of a single overlay, in the P2P-XBone the OM does not have to maintain any node-specific configuration parameters because the configuration of IP tunnels and the routing tables are managed by the P2P component. Also, the OM in this case contacts each RD only when an associated virtual node joins a P2P-VI. Therefore, P2P network size scalability is not affected by the centrality (per overlay) of the OM. The OM also acts as an address server, which manages IP address blocks available for VIs to configure IP tunnels. It is necessary to coordinate the unique assignment of virtual IP addresses in a VI. The address server dynamically assigns IP addresses to a new tunnel and reclaims them for a released tunnel. This service can be located separately from the OM although it coexists in the figure (and in the current implementation). The address server can be a bottleneck of P2P-XBone especially in cases where virtual nodes frequently join and leave because virtual IP addresses are requested/released corresponding to IP tunnel creations/releases invoked by neighbor node changes in a P2P protocol. However, the load can easily be distributed using multiple address servers. We show how many IP address requests/releases occur in various cases in Sect. 3.

2.1 Extensions to the X-Bone

The P2P-XBone adds three key features: (i) individual join/departure of virtual nodes, (ii) IP tunnel creation/release and (iii) routing table configuration, based on requests from P2PD.

2.1.1 Dynamic Join to/Departure from a VI

Although the existing X-Bone supports only the entire deployment of a VI, a virtual node has to be able to individually join and leave a P2P-VI in the P2P-XBone. When a new virtual node joins a P2P-VI, the OM sends an invite message to the RD with the virtual node. This invite either is requested by the new virtual node itself or is initiated by the administrator of the P2P-VI. If the RD accepts the invitation, a configure message is sent to the RD with a set of

initial parameters, which include the base IP address corresponding to a bootstrap node to which the new node should firstly connect. This bootstrap node is randomly picked from among participating virtual nodes by the OM, as done in existing P2P protocols [10]. In the P2P-XBone, unlike in application-level P2Ps, the new virtual node cannot assume reachability to the bootstrap node at the initial state. The RD therefore establishes an IP tunnel to the bootstrap node using the procedure described in Sect. 2.1.2. After that, the RD launches a P2PD for the virtual node, where the virtual IP address assigned to the bootstrap node in the earlier IP tunnel creation is set to the P2PD as the bootstrap node address. The launch of P2PD is performed using the application deployment functionality [11] in the X-Bone. After the stabilization routine of a P2P protocol begins, the RD configures IP tunnels to neighbor nodes and routing table entries based on control from the P2PD to establish reachability to all other participating nodes. The IP tunnel to the bootstrap node can be released later, when the P2P topology is established, unless it is required for that topology. Node departure procedures are much simpler than node join procedures. In the case of a voluntary node departure, a P2P node needs only to release all IP tunnels and delete routing table entries before departure. Even in the case of silent departure due to node failure, P2PDs on neighbor nodes can detect node death through the keep-alive mechanism of a P2P protocol and unnecessary IP tunnels can be automatically released.

2.1.2 IP Tunnel Configuration

In the P2P-XBone, IP tunnels are dynamically created and released in a P2P fashion based on requests from a P2PD in a given virtual node to the RD on the host. When the P2PD triggers creating a new IP tunnel, it includes the base IP address of the other end of the tunnel in the request. The base IP address can be replaced with other contact information for the other end as described in Sect. 2.2.1. If an IP tunnel is already established to the virtual node for the P2PD, the RD just responds with the virtual IP address pair of the IP tunnel to the P2PD to avoid duplicated tunnel creation. If not, a new IP tunnel is created. Before actually configuring an IP tunnel, the RD obtains a pair of virtual IP addresses (i.e., inner IP addresses) for the IP tunnel from the address server. These addresses are used to configure virtual interfaces in the virtual node. Then it sends a tunnel creation request to the RD at the peer node, where the base IP address given by the P2PD is used. Note that this request is sent over the base network. The RD that received the request configures a virtual interface for the IP tunnel and sends back an ACK message. When the requesting RD receives the ACK message, it configures a virtual interface on the local node as well and then responds with the virtual IP address pair to the P2PD to indicate that the IP tunnel is successfully established.

On the other hand, when the local RD is asked to tear down an existing IP tunnel, it sends a tunnel release request to the RD on the peer node and releases the IP tunnel by unconfiguring the corresponding virtual interfaces at both sides. The virtual IP address pair for the released IP tunnel is returned to the address server. It takes much less time for the P2PD to release an IP tunnel than to create an IP tunnel because the P2PD can proceed to subsequent steps without waiting for a reply message from the RD.

2.1.3 Routing Configuration

The P2PD also works like a routing daemon for a P2P-VI. The P2PD asks the local RD to modify routing table entries for a virtual node when any change in routing entries occurs in the user-level P2P daemon. As mentioned before, forwarding in P2P-XBone happens at the kernel level. By factoring out forwarding functionality from the P2PD, we not only simplify the daemon but also improve forwarding performance. P2P protocols support late binding of the destination through support for content-based forwarding. The P2P-XBone introduces the late-binding property to a VI as well. Because conventional OSes do not support such data-based routing functionality, we support it using kernel extension at nodes participating in a VI. DataRouter [9] is an experimental extension to IP that supports pattern-match-based routing and forwarding at a kernel level. It extends the Loose Source Route option in IPv4 to encode a string destination identifier such as a URL or a hash value. Using IP options elsewhere in the Internet could be impractical because some ISPs have a filtering policy to discard any IP option packets. However, DataRouter packets over a P2P-VI look like normal IP packets in a base network since all P2P nodes are connected via IP tunnels. Therefore, such filtering issues can be avoided.

In our implementation, we use the FreeBSD 5.3 kernel extended with DataRouter functionality to support content-based forwarding at the kernel level and Chord as an example of P2PD. In this case, the RD adds or deletes a routing entry using the `droute` command to set a routing entry to the DataRouter kernel as follows,

`droute (add|del) range` *minid maxid nexthop*

where *minid* and *maxid* corresponds to hash values of both ends in a range of the Chord identifier circle. The DataRouter can support the data routing service to other DHT protocols such as Pastry [4] and CAN [5] in a similar fashion.

2.2 Extensions to P2P Protocols

P2PD is a modified implementation of the existing P2P protocol daemons. Although such a control module can be implemented at the network layer, the network-level implementation has several disadvantages in portability and modularity. As existing routing protocols generally run at the application layer, P2PD takes a similar application-level approach. We use Chord as the target P2P protocol daemon for adapting to the P2P-XBone environment and we discuss the details of the modifications in this section.

2.2.1 General Extensions

The P2PD sends control messages in response to four DHT protocol events: addition/deletion of a neighbor node and addition/deletion of a routing entry. The control messages request creation/release of an IP tunnel and addition/deletion of a routing table entry for the virtual node, respectively. The most challenging and interesting part in designing the P2PD is how to establish IP tunnels corresponding to neighbor nodes in the DHT protocol. This is challenging because, in the P2P-XBone, the nodes that configure the IP tunnel need to know each other's base IP addresses, although the only information available in a regular DHT protocol is the remote end's P2P ID and virtual IP address. Besides, the P2PD cannot directly communicate with the neighbor node over the VI until an IP tunnel is established to that neighbor, and there is no centralized mechanism to map the P2P ID to the base IP address. It would violate the boundary between a VI and a base network (i.e., virtualization boundary) to simply add base IP addresses to parameters treated in DHT. To achieve strict virtualization, nothing within a VI should know any parameters in the underlying network including base IP addresses. We solve this issue by introducing a message carried over P2P-VI using the P2P forwarding in user space or at the kernel-level, in which the contact information of a sender node is included in an opaque manner, to ask the neighbor to create an IP tunnel. Currently, the base IP address of the sender is used as the contact information, while other bits of information such as domain name and URL could be used.

Figure 2 shows the sequence of establishing an IP tunnel in the P2P-XBone. The neighbor connection request is a message to request to establish an IP tunnel to a potential neighbor node. The potential neighbor node is either informed of by existing neighbor nodes via a DHT's self-organization mechanism or is the destination of the P2P message that is not known ahead of time. This message is delivered over the existing P2P-VI and its receiver is identified based on the DHT routing identifier—a hash value in case of Chord. Before actually sending the neighbor connection request message, the contact information of the node itself is obtained via the API between the P2PD and its corresponding RD (*request contact info*) to be included in the message (Step (1)–(2) in Fig. 2). The information type has to be agreed upon by the RD. The obtained information (e.g., base IP address) is treated as an opaque parameter by the P2PD so as not to violate the virtualization boundary. The destination P2PD identified by the P2P routing identifier terminates the message (Step (3)) and requests that the RD establish an IP tunnel to the source node (*add tunnel*; Step (4)). The RD establishes an IP tunnel as described in Sect. 2.1.2 (Step (5)–(8)) and responds with the virtual IP address pair of the established IP tunnel (Step (9)). To deal with an unexpected node failure, the tunnel configuration is reset if an error or timeout is detected during these steps. Finally, the destination P2PD sends an acknowledgment to the source confirming the tunnel creation over the newly established tunnel (Step (10)). After the new tunnel is established, the RD is asked to modify routing table entries on its node kernel if any routing change occurs in the P2PD, as described in Sect. 2.1.3[†]. The above procedure does not necessarily require the addition of a new message to the set of existing DHT messages. In some DHT protocols that have a message to discover appropriate neighbor nodes, that message can be extended instead. We show an example of extending the find successor message in the Chord protocol in Sect. 2.2.2.

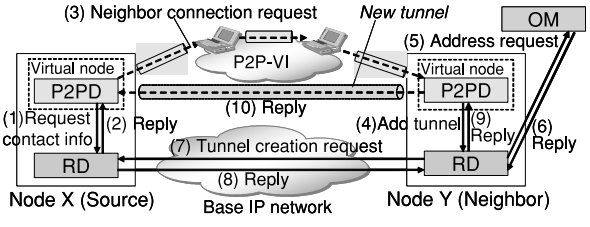


Fig. 2 Sequence of establishing an IP tunnel to a neighbor node.

Because it takes more time for the extended DHT protocols to configure a neighbor node than for existing ones due to the extra cost incurred by the tunnel creation procedure, the convergence performance of a P2P system could be degraded, especially when P2P nodes frequently join and leave. We will show simulation results on convergence times in Sect. 3.1.

2.2.2 Chord-Specific Extensions

We modified i3's Chord implementation [6] for our prototype. The pseudocode of the extended functions are shown in Fig. 3.

The find_successor() function is periodically run to fix

```
//n: ID of which to find the successor
//n′: ID of the node which is responsible for n
//x_b: base IP address of the source node
//x_v1, x_v2: local and remote virtual IP addresses of established IP tunnel

find_successor(n)
  x_b = request_contact_info();
  send_find_successor(x_b, n);

receive_find_successor(x_b, n)
  if (chord_is_local(n)) // check if responsible for n
    (x_v1, x_v2) = add_tunnel(x_b);
    send_find_successor_reply(n′, x_v1, x_v2);
  else
    send_find_successor(x_b, n);
```

Fig. 3 Pseudocode of extended Chord functions.

[†]Because the particular routing algorithm depends on what kind of P2P protocol is used, the establishment of the tunnel between node X and Y does not always mean support direct transfer for data from node X to Y. For example, when proximity route selection described in Sect. 2.3 is used, a multi-hop path could have a priority over a single-hop path.

the finger table (i.e., neighbor node list in Chord). It first calls the request_contact_info() function to obtain the base IP address of the physical node on which the P2PD is running and sends the find successor message to the closest predecessor node to n in the existing neighbors with the obtained base IP address. The receive_find_successor is a function called when a node receives the find successor message. If the node is the immediate successor to n, it terminates the message and calls the add_tunnel() to request IP tunnel creation. These extensions could be implemented with a small amount of extra code: about 1000 lines of C including the APIs within the RD.

2.3 QoS Considerations

The existing X-Bone supports QoS control for VIs, which includes limiting bandwidth, adding latency, etc. for each IP tunnel. In the P2P-XBone, the same QoS control mechanisms can be used. Such QoS-related parameters are configured to the OM by an administrator when a P2P-VI is defined, and are sent to each RD with other kinds of initial parameters when a virtual node joins the P2P-VI. This QoS configuration is applied to every subsequent IP tunnel because IP tunnel creation is performed in a P2P fashion and separate, different-QoS configuration for each IP tunnel is not intended[†].

In DHT protocols, various approaches to support QoS in overlays have been proposed [8], [12]. While there are some approaches including data placement algorithms and transport-layer modifications [8], in terms of enhancing topology and routing, proximity neighbor selection (PNS) and proximity route selection (PRS) [12] can be applied commonly to most DHT protocols only with the small changes in DHT algorithms and the addition of a QoS measurement mechanism. In PNS and PRS, neighbor/route selections depend on monitored QoS parameters such as latency and bandwidth to multiple candidate nodes. In application-level overlays, that is easily achieved because all existing nodes are assumed to be IP reachable with each other. When applying PNS and PRS to the P2P-XBone, the QoS measurement to other nodes would be restricted in PNS while PRS can be applied as it is. This is because, in the P2P-XBone, an IP tunnel has to exist for the measurement over a VI. However, it is not efficient to create IP tunnels to all candidate nodes solely for measurement. In the P2P-XBone, RD can more directly support such measurement by providing APIs for network commands like *ping* and *pathchar*[††]. The P2PD would request the QoS measurement through the API. The RD would then execute a corresponding network system call or command based on the type of the API and returns the results to the P2PD. Note that in this case the measurement is performed on the underlying network. In the API, the base IP address or other contact information of the other end in the measurement has to be given similarly to the case of IP tunnel creation. The information of the other end should be delivered in an opaque manner as described in Sect. 2.2.1. We expect that the P2PD would then use this QoS information to alter the routing table sent to the RD for insertion in the kernel, i.e., that QoS would affect the routing table as a whole. Per-packet QoS decisions may be supported by using more elaborate string matching and substitution capabilities of the DataRouter, but has not been considered in this work.

2.4 Security Considerations

The P2P-XBone supports the same level of security as the existing X-Bone system. That is to say, authentication and encryption are performed using SSL for communication between OM and RDs and using IPsec (transport mode) over IP-in-IP tunnels for virtual links between virtual nodes, respectively. Each RD and OM has its own access control list (ACL), which is used for resource access permissions and restrictions based on usernames. Such resources include number of overlays, number of tunnels, queue limits, bandwidth limits, etc. When a virtual node joins a P2P-VI, it is authenticated based on ACLs in the corresponding OM and RD and is authorized to join if the resource availability meets conditions required for the P2P-VI.

IPsec is configured in the X-Bone using out-of-band secure exchange of shared private keys; this project shifted that effort to the RDs, to prevent the OM from becoming a bottleneck. In the existing X-Bone, IPsec keys are allocated for each IPsec association (one association for each direction) comprising a VI in a centralized manner by the OM when the VI is deployed. In the P2P-XBone, however, IPsec associations are dynamically established in a P2P fashion. Therefore, the keys are calculated by a node initiating an IPsec channel and are shared at both ends via an SSL channel between RDs.

3. Performance Evaluation

The P2P-XBone can provide some advantages compared with conventional application-level P2P deployment, but these advantages cannot be obtained without any tradeoff. Specifically, the P2P-XBone can provide better forwarding performance than application-level P2Ps due to kernel-level processing. To configure IP tunnels and routing tables, however, a P2P protocol has to be extended to support the control messages to the RD as described in Sect. 2.2. This control overhead increases the configuration delay of P2P configuration messages. Therefore, there is a tradeoff between the improvement in forwarding performance and the degradation in routing propagation and provisioning performance. Using IP tunnels instead of application-level channels such as UDP also involves virtual IP address management. The address server can be a system bottleneck due to

[†]Note that these extensions support limit-based QoS, e.g., as used for emulation. Performance-enhancing QoS is not supported in the X-Bone because it is not typically available in the base network, and thus cannot be virtualized.

[††]The implementation of this functionality is future work.

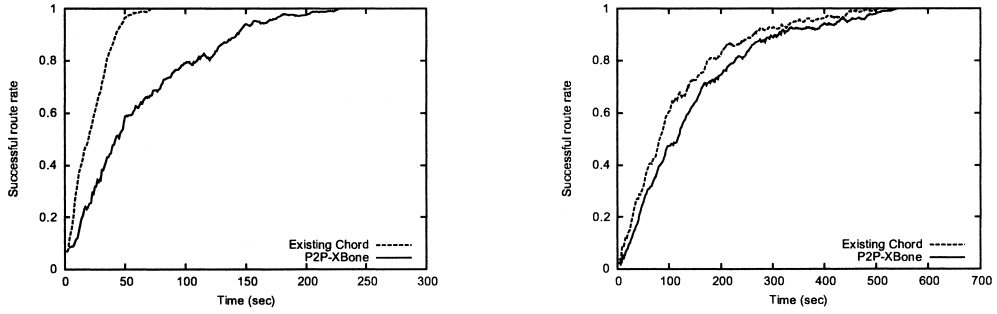
its centralized property as described in Sect. 2. In the following sections, we evaluate our proposed system on these metrics and discuss its tradeoffs.

3.1 Convergence Performance of a DHT Protocol

To evaluate the impact of the overhead in P2P message propagation, the convergence performance in Chord protocol was compared between the original version and the one extended for P2PD using simulation. First, we let 100 nodes join a P2P system at the beginning of the simulation and then measured the rate of available paths in the P2P network against time by continuously generating data transfers between any two nodes to evaluate its route convergence time. We also tested a few values of the stabilization period in Chord protocol, which is a period of the find_successor() function called and corresponds to a keep-alive period in conventional routing protocols. In the P2P-XBone, it takes more time for the P2PD to receive the ACK message (find successor reply) for the find successor message because two extra functions (request_contact_info() and add_tunnel()) are involved as shown in Fig. 3. These overheads were emulated by idling for average time taken for these functions as observed in real experiments. The simulation results in Fig. 4 show that the P2P-XBone increases convergence time compared to the existing Chord for short stabilization periods such as 1 sec. However, as the stabilization period lengthens, the convergence performance difference reduces. This is because the longer stabilization periods mask the time required for the extra procedure calls in the P2P-XBone. In our simulation, 5 sec was long enough to get almost the same stabilization convergence performance even in cases where the P2P-XBone had more participating nodes.

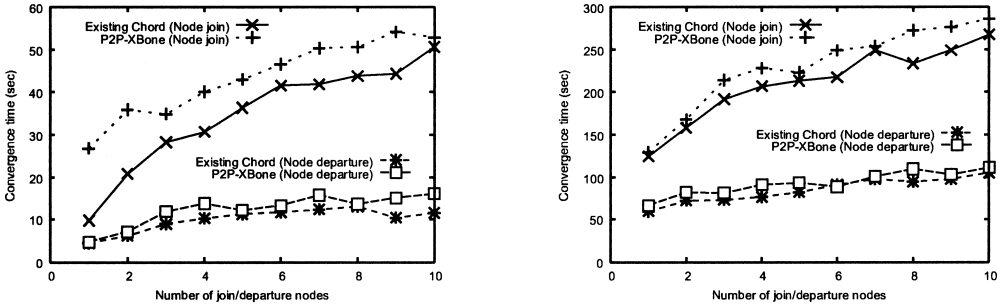
Next, we compared the topology convergence time when a variable number of nodes simultaneously join or leave a system that begins with 100 nodes. The topology convergence means the Chord topology has adapted to corresponding changes in participating nodes. As shown Fig. 5, the topology convergence time increases (roughly) linearly with the number of join/departure nodes. In both the cases of node join and node departure, the P2P-XBone shows longer time due to the overhead of its tunnel creation/release operations. The increased ratio in topology convergence time becomes smaller as the stabilization period is longer just as it did for stabilization convergence. Comparing node joins with node departures, the gap between the existing Chord and the P2P-XBone in node joins is larger than that in node departures. This is because node joins involve tunnel creations where node departures involve tunnel release operations, and the latter imposes much shorter extra delay, as described in Sect. 2.1.2.

We showed the impact of the extra delay in P2P message propagation through the simulations on convergence performance. Although longer stabilization periods reduce the relative degradation in convergence times, they increase the convergence time itself. Therefore, users should choose an appropriate stabilization period based on application requirements. These results suggest that the P2P-XBone is



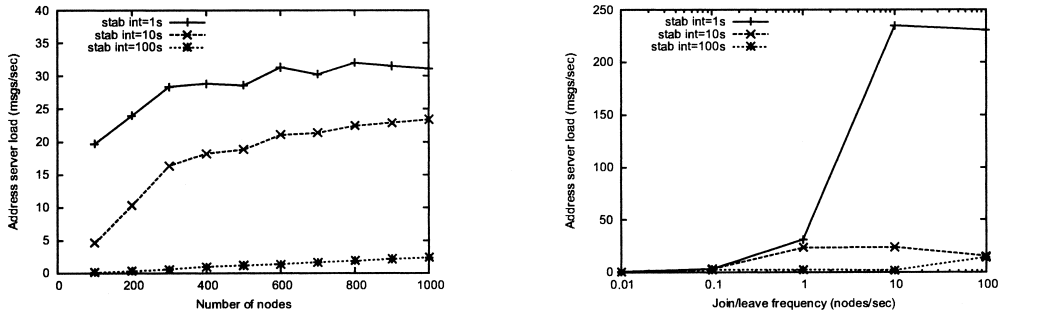
(i) Stabilization period = 1 sec (ii) Stabilization period = 5 sec

Fig. 4 Convergence property in Chord.



(i) Stabilization period = 1 sec (ii) Stabilization period = 5 sec

Fig. 5 Convergence time after node join and departure.



(i) The case of changing the number of nodes (ii) The case of changing the join/departure frequency

Fig. 6 Address server load.

better suited to P2P systems comprised of relatively stable nodes.

3.2 Address Server Load

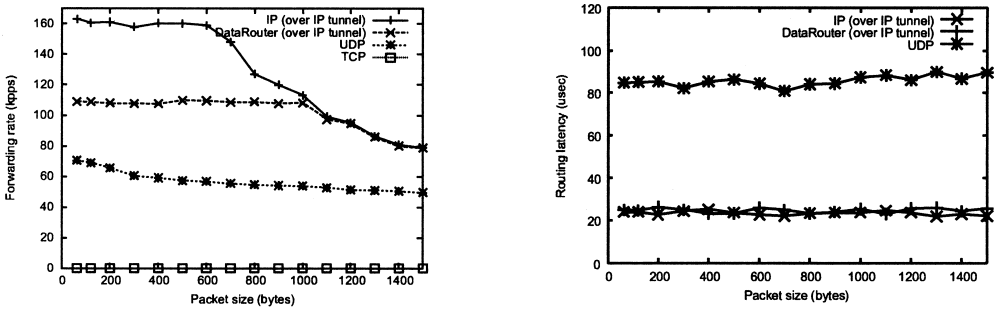
As described in Sect. 2.1.2, in the P2P-XBone, RD needs to obtain and release a pair of virtual IP addresses when establishing and tearing down an IP tunnel, respectively. We measured how many address requests/releases occur for a variable number of nodes and for various join/departure frequencies of P2P nodes. The goal is to determine how many address servers are necessary for load-balancing. In Fig. 6(i), the number of nodes was varied with fixed join/departure frequency (one node join or departure per second). In this simulation, three values of stabilization period were tested. The results show that more address requests/releases occur as more nodes participate in a P2P-VI. This reflects the fact that the number of fingers (routing table entries, as well as tunnels) in a node is proportional to $\log n$, where n is the total number of participating nodes. In terms of stabilization period, larger values reduce the number of address requests/releases. These results can be explained as follows. As the stabilization period is longer, the number of node joins/departures that occur in one stabilization cycle increases. Basically, as more node joins/departures are aggregated into a stabilization routine, the number of tunnel creations/releases for each join/departure triggered by this routine is reduced due to the aggregation effect, compared to when the topology is adapted everytime a node joins/leaves. Therefore, the average number of address requests/releases per unit time decreases as the stabilization period is longer. As observed in Sect. 3.1, a long stabilization period makes convergence performance worse. These results suggest that there is a tradeoff between convergence time of a P2P-VI and the required number of address servers.

Next, the results in which the join/departure frequency is varied with the fixed average number of nodes (1000 nodes) are shown in Fig. 6(ii). These results show that setting stabilization period to a short value such as 1 sec can generate a huge number of address requests/releases under churn (in which 1–10 percent of nodes are turned over per second). Under too much churn, the number of messages peaks without further increasing. This is because node joins and departures are too frequent for stabilization routine to detect them all. Although the address server load should increase in proportion to the join/departure frequency in ideal conditions, these results show that under such churn conditions the processing of tunnel creations/releases cannot catch up with the speed of node changes due to system performance limitations.

In our preliminary implementation, the address server could deliver 67 msgs/sec over UDP, 56 msgs/sec over TCP and 4.2 msgs/sec over SSL (where TCP and SSL deliver one message per connection). The number of address servers required for a P2P-VI should be determined considering the tradeoff with convergence performance, although only one server is sufficient where node joins and departures are not very frequent.

3.3 Forwarding Performance

We implemented DataRouter on FreeBSD 5.3 and measured its forwarding rate and latency on a dual-processor 2.4 GHz Xeon PC with 1 Gbyte main memory. An IXIA hardware-based Ethernet packet generator was directly connected to 64-bit/66 MHz PCI gigabit Ethernet card on the PC, and the number of packets that are forwarded was counted on the generator for various packet sizes. Throughout these experiments, only the IPv4 protocol was used. The results are shown in Fig. 7. DataRouter forwarding based on range-matching of hash values was compared with application-layer forwarding (over UDP and TCP) and normal IP forwarding for reference. In this measurement, conditions for both IP and DataRouter forwarding are different from those in our previous measurement [9] in that here IP tunnels are used to connect nodes. Therefore, decapsulation and encapsulation of an outer IP header are performed before and after forwarding for an inner packet, respectively. In fact, using an IP tunnel achieves only half the forwarding speed as the non-IP-tunneling because the IP forwarding operation is performed twice for each packet. DataRouter still provides much better forwarding rate performance than application-layer forwarding despite this penalty. Although our prototype implementation provides only 70 percent the normal IP forwarding rate, this gap may be further reduced by optimizing the implementation. Both the IP and DataRouter forwarding rates remain almost constant as long as the packet size is not large enough to saturate the link capacity, whereas



(i) Forwarding rate (ii) Forwarding latency

Fig. 7 Forwarding performance.

IP packets with more than 600 bytes and DataRouter packets with more than 1000 bytes make throughput peak at 850–950 Mbps. Unlike kernel-level forwarding, application-layer UDP forwarding performance gets slightly worse as packet size is larger. This is because the time required for recv/send on a socket depends on data size (i.e., packet size). TCP provides even worse performance because a new TCP connection is used for each data transfer, so that the performance is limited by the rate of establishing new connections. As to forwarding latency, DataRouter achieves 1/4 the latency of application-layer UDP forwarding, regardless of packet size. DataRouter also shows comparable latency of normal IP forwarding.

Throughout these results, it was shown that kernel-level data forwarding using an IP option such as DataRouter is effective even when IP tunnels are used between nodes. We plan to optimize the implementation of DataRouter and to extend it to support IPv6, which allows more unconstrained use of IP option field.

4. Discussion on Routing-Driven Provisioning

The P2PD integrates provisioning and routing to provide a unique topology management strategy to a VI. In traditional networks, provisioning is done first, then routing is performed on top of the pre-provisioned topology. This means that existing routing protocols only pick one link from among existing candidates to reach a next hop. However, the P2PD does not assume a fixed set of links but dynamically creates/releases links based on the changes of neighbor nodes, which are selected by the (P2P) routing protocols. This integration of provisioning and routing brings new capabilities such as load-balancing, resilience, robustness, scalability, etc. to a network system, just as an application-level P2P does. Also note that for the P2PD, provisioning and routing are performed at different layers, i.e., provisioning is performed underneath the layer where routing is performed. Therefore, an interworking mechanism between modules at different layers is needed to achieve the routing-driven provisioning. Section 2 shows how this is achieved through the dialogue between the P2PD and the RD in a VI's operation. The P2PD could be further generalized to provide topology management and routing strategy to other types of networks than virtual IP networks, such as a layer-2 network over optical paths.

When the routing-driven provisioning is generalized to any networks, how provisioning is performed at the underlying layer has to be taken into account. Provisioning can be classified into two ways based on whether the path that a virtual link goes through at the underlying layer is fixed: (i) tunnel-based provisioning and (ii) path-based provisioning. The former includes PPP, GRE and IP tunnels, in which configuration is performed only at the ends of a link. The latter includes ATM-VCs and label switched paths in MPLS, where link configuration has to be performed at intermediate switches as well as at both ends. The P2P-XBone relies on tunnel-based provisioning. To support the path-based provisioning, the RD needs to be modified to interwork with such a signaling mechanism as RSVP-TE [13] to facilitate provisioning of switched paths. In either case, the P2PD should not be aware of the difference in provisioning style at the underlying layer but should be able to configure a network with common APIs [14].

5. Related Work

IP-based overlays are widely used to deploy new and experimental protocols over existing IP networks. M-Bone, A-Bone [15], and 6-Bone [16] are some of the well-known testbeds. The Virtual Internet (VI) is a generalized architecture for such IP virtual networks that supports recursion (i.e., stackable virtual networks) and re-visitation (multiple virtual nodes in a single base node). The X-Bone [2] is a system to enable automated deployment of VIs. GX-Bone [17] extends the X-Bone by adding a global LDAP directory infrastructure for resource registration and discovery. UMU-PBNM [18] addresses automated VPN deployment, though it focuses on standardized XML-based configuration and PKI-based authentication. Although both X-Bone and UMU-PBNM can create/delete VIs and VPNs in an on-demand fashion, neither supports individual node join/departure and self-organization.

In the P2P area, there have been several proposals which apply the characteristics of DHT to such lower layer stack as Layer 2 and 3. P6P [19] provides a scheme to connect isolated IPv6 sites by tunneling IPv6 packets between

edge routers in an IPv4 network. It uses a DHT lookup mechanism to resolve ingress and egress routers and then establishes tunnels in between. There is no P2P routing involved. ELA [20] is a distributed VPN system with a two-layer hierarchy which creates a fully-meshed topology when traffic occurs among all edge nodes. By using a full mesh, ELA avoids the need for routing or incremental provisioning, but this assumes a topology which is inherently unscalable. PeerNet [21] is an augmented routing scheme in a wireless ad-hoc network, in which a location-aware ID is assigned to each node to enable DHT-like routing. PeerNet assumes only physical links but not tunnels for connecting nodes and does not involve dynamic provisioning as the P2P-XBone does.

6. Conclusion and Future Work

This paper describes the P2P-XBone which deploys P2P systems as virtual IP networks instead of application layer overlays. The provisioning of virtual links (IP tunnels) in P2P-XBone is driven by the neighbor state changes according to the P2P routing protocol used. The DataRouter is also added to support in-kernel, string-based forwarding within the P2P virtual networks. This combination provides the core characteristics of common P2P systems including self-organizing topology, fault tolerance, and content-based routing. A prototype based on existing X-Bone software was developed, and compared with the Chord implementation. The benchmarks show significantly better forwarding performance than Chord, albeit with slightly more signaling overhead and convergence time. The modular nature of the P2P-XBone also makes it simpler to implement or plug-in new P2P protocols, as well as supporting the use of conventional transport protocols for P2P transfers, because it re-uses existing IP network services rather than duplicating the functionality at the application layer.

The prototype was implemented on FreeBSD 5.3 and is freely available as part of the X-Bone software distribution for use in other P2P systems. Future work includes generalization of APIs to support broader types of P2P protocols and framework for recursive P2P virtual networks.

Acknowledgement

The authors thank the numerous contributors to the code and architecture of the X-Bone project, both within projects of USC/ISI (X-Bone, DynaBone, NetFS, DataRouter) as well as in collaboration.

This work was partly supported by the US NSF STI-XTEND (ANI-0230789). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

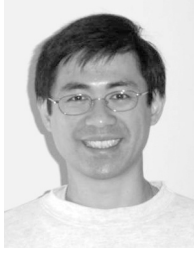
References

[1] J. Touch, "Dynamic Internet overlay deployment and management using the X-Bone," Comput. Netw., vol.36, no.2-3, pp.117–135, July 2001.
[2] X-Bone URL—http://www.isi.edu/xbone/.
[3] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," Proc. ACM SIGCOMM, pp.149–160, San Diego, CA, Aug. 2001.
[4] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," Proc. Middleware, pp.329–350, Heidelberg, Germany, Nov. 2001.
[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," Proc. ACM SIGCOMM, pp.161–172, San Diego, CA, Aug. 2001.
[6] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," Proc. ACM SIGCOMM, pp.73–86, Pittsburgh, PA, Aug. 2002.
[7] 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," Proc. ACM SIGCOMM, pp.73–84, Philadelphia, PA, Aug. 2005.
[8] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," Proc. 1st USENIX Symposium on Networked Systems and Implementation (NSDI'04), pp.85–98, San Francisco, CA, March 2004.
[9] J. Touch and V. Pingali, "DataRouter: A network-layer service for application-layer forwarding," Proc. International Workshop on Active Networks (IWAN), pp.113–124, Kyoto, Japan, Dec. 2003.
[10] K. Shen, "Structure management for scalable overlay service construction," Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'04), pp.281–294, San Francisco, CA, March 2004.
[11] Y. Wang and J. Touch, "Application deployment in virtual networks using the X-Bone," Proc. DARPA Active Network Conference and Exposition (DANCE), pp.484–493, May 2002.
[12] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," Proc. ACM SIGCOMM, pp.381–394, Karlruhe, Germany, Aug. 2003.
[13] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP tunnels," RFC 3209, Dec. 2001.
[14] R. Kompella, A. Greenberg, J. Rexford, A. Snoeren, and J. Yates, "Cross-layer visibility as a service," Proc. Fourth ACM Workshop on Hot Topics in Networks (HotNets), College Park, MD, Nov. 2005.
[15] A-Bone URL—http://www.isi.edu/abone/.
[16] 6-Bone URL—http://6bone.net/.
[17] J. Touch, Y. Wang, V. Pingali, R. Zhou, G. Finn, and L. Eggert, "A global X-Bone for network experiments," Proc. IEEE Tridentcom 2005, pp.194–203, Trent, Italy, March 2005.
[18] F. Clemente, G. Millan, J. Re, G. Perez, and A. Gomez-Skarmeta, "Deployment of a policy-based management system for the dynamic provision of IPsec-based VPNs in IPv6 networks," Proc. 2005 Symposium on Applications and the Internet (SAINT) Workshops, pp.10–13, Trento, Italy, Jan. 2005.
[19] L. Zhou and R. Renese, "P6P: A peer-to-peer approach to Internet infrastructure," Proc. International Workshop on Peer-to-Peer Systems (IPTPS'04), San Diego, CA, Feb. 2004.
[20] S. Aoyagi, M. Takizawa, M. Saito, H. Aida, and H. Tokuda, "ELA: A fully distributed VPN system over peer-to-peer network," Proc. 2005 Symposium on Applications and the Internet (SAINT), pp.89–92, Trento, Italy, Jan. 2005.
[21] J. Eriksson, M. Faloutsos, and S. Krishnamurthy, "Scalable ad hoc routing: The case for dynamic addressing," Proc. IEEE INFOCOM, pp.1108–1119, 2004.

Norihito Fujita received the B.E. and M.E. degrees in electrical engineering from Kyoto University, Japan, in 1996 and 1998, respectively. He joined NEC Corporation in 1998 and is an assistant manager at System Platforms Research Laboratories, NEC Corporation. He was a visiting researcher in the X-Bone group at the Univ. Southern California/Information Sciences Institute from 2004 to 2005. His research interests include routing and security in computer networks. He received the IEICE Switching Systems Engineering Technical Group Research Award and the IEICE Young Investigators Award in 2000 and 2004, respectively.

Joseph D. Touch received a B.S. with Honors in biophysics and computer science from the Univ. of Scranton in 1985, an M.S. in CS from Cornell Univ. in 1987, and a Ph.D. in CS from the Univ. of Pennsylvania in 1992. He joined the Univ. Southern California (USC)/Information Sciences Institute (ISI) in 1992 and is Director of the Postel Center in the Computer Networks Division of USC/ISI. He is also a Research Associate Professor in USC's Computer Science and EE/Systems Departments. His interests include Internet protocols, network architecture, high-speed & low-latency nets, network device design, and experimental network analysis. He is a member of Sigma Xi (A'84-F'93), IEEE (S'83-M'92-SM'02), and ACM (S'83-M92), and is currently ACM SIGCOMM's Conference Coordinator, IEEE Infocom 2006 Program Chair, a member of numerous conference steering and program committees, and is active in the IETF. He also serves on the editorial board of IEEE Network.

Venkata Pingali received the B.Tech. and M.S. degrees from IIT Bombay, India and University of Utah, respectively. He is currently a Ph.D. student at the Univ. Southern California/Information Sciences Institute. His research focuses on automatic network configuration.



Yu-Shun Wang received the B.S. degree in Electronic Engineering from National Chiao Tung University, Taiwan and the M.S. degree in Computer Engineering from the University of Southern California (USC). He is currently a Ph.D. candidate at USC/Information Sciences Institute. His research focuses on network architecture, security and virtual networks.