# P2P-XBone: A Virtual Network Support for Peer-to-Peer Systems

USC/ISI Technical Report ISI-TR-2005-607

Norihito Fujita [1] Joseph D. Touch [2] Venkata Pingali [2] Yu-Shun Wang [2]

[1] System Platforms Research Laboratories, NEC Corporation (Visiting Scholar of USC/ISI)

[2] USC/Information Sciences Institute

September 27, 2005

***Abstract*– An architecture to deploy virtual IP networks with P2P-like dynamic topology management is described. Existing virtual IP network deployment mechanisms do not allow for dynamic topology adaptation and fault-tolerance since 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 protocols 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 availability of existing network services. We also show several evaluation results on the overhead of P2P-driven provisioning and on forwarding performance.**

## I. 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], in which hosts and routers are logically emulated in physical nodes and any virtual IP network topology can be created in a manner of connecting them by IP tunnels such as IP-in-IP and GRE. Existing VI deployment systems including the X-Bone provide static virtual topology and perform provisioning of IP tunnels only once at deployment of a VI. Therefore, the VI has no mechanism for dynamic node addition/deletion as physical IP networks do, which constrains the usability and capability of a VI compared with more dynamic application-level overlays. 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. P2P networks, unlike regular networks, achieve these promising properties by modifying the topology to reflect the routing changes. Existing P2P protocols [3][4][5] and platforms [6][7] run at the application layer and use UDP or TCP to connect P2P nodes. Therefore, they only change the application-level logical topology over an existing IP network. In a VI, however, since 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 the exisiting P2P.

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 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. The P2P protocol provides a unique property of routing-driven provisioning to a VI due to its ability to control IP tunnels as well as a routing table, while traditionally routing and provisioning have been considered as being independent. We call such a P2P-driven VI simply a P2P-VI. The P2P-XBone not only provides P2P's high resilience to a VI but also enables a VI to achieve content-based routing that is another promising property of P2P. Although it is difficult for a normal VI to

support such data-based forwarding as existing P2P systems do because they usually use such non-IP-address-based IDs as URLs and hash values for destination IDs, the P2P-XBone addresses the issue by involving a kernel extension module that supports string-based forwarding. In our implementation, the DataRouter module [8] is used to support the kernel-level application forwarding. Thus the P2P-VI can be alternative P2P infrastructure that is achieved at the virtual IP layer. Deploying P2P systems at the virtual IP layer provides a couple of advantages for themselves compared with conventional application-level deployment. Firstly, higher forwarding performance can be obtained because forwarding is performed at the kernel level. 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 for the implemented system.

## II. P2P-XBone

### A. Overall architecture

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 comprised 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 constituting a VI (virtual node; VN) and is responsible for configuring IP tunnels and routing tables on the VN. In the P2P-XBone, the RD is extended to let these configuration changes be triggered by the P2PD. The details are described in Sec. II-B. 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 VNs 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 Sec. II-C.

While both RD and P2PD run on nodes participating in a P2P-VI, Overlay Manager (OM) works outside the nodes to provide functionalities that are necessary to coordinate a P2P-VI. When deploying a new P2P-VI, an administrator defines it via API provided by 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 VN requests to join the P2P-VI, the configuration parameters are sent to the RD for the VN. While the OM in the existing X-Bone manages all parameters on a VI including virtual interfaces and routing entries on VNs in a centralized manner, that in the P2P-XBone does not have to maintain any node-specific configuration parameters since the configuration of IP tunnels and a routing table is performed completely in a P2P fashion after a VN joins a P2P-VI. Also, the OM contacts to each RD only when an associated VN joins a P2P-VI. Therefore, scalability is not degraded by the existence of the OM. The OM also has the functionality of address server, which manages IP address blocks available for VIs to configure IP tunnels. It is necessary to uniquely coordinate virtual IP addresses in a VI. The address server dynamically assigns IP addresses to a new tunnel and reclaims them for a released tunnel. It can be located separately from the OM although it coexists in the figure. The address server can be a bottleneck of P2P-XBone especially in cases where VNs 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 Sec. III.

### B. Extensions to the X-Bone

The P2P-XBone adds three key features: (i) individual join/departure of VNs, (ii) IP tunnel creation/release and (iii) routing table configuration,

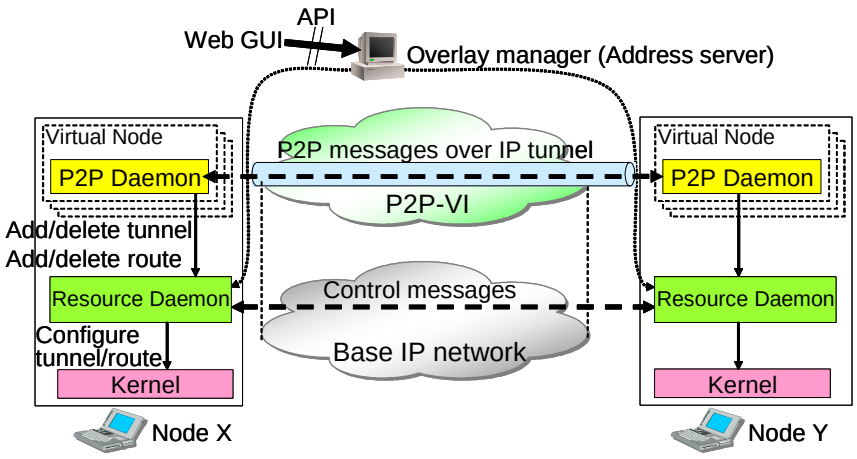



Fig. 1. High-level architecture of the P2P-XBone.

based on requests from P2PD.

### B.1 Dynamic join to/departure from a VI

While the existing X-Bone supports only whole-scale deployment of a VI, a VN has to be able to individually join and leave a P2P-VI in the P2P-XBone. When a new VN joins a P2P-VI, the OM sends an invite message to the RD with the VN. This invite either is requested by the new VN itself or is initiated by the administrator of the P2P-VI. If the RD accepts the invite, a configure message is sent to the RD with a set of initial parameters, which include the base IP address corresponding to a VN to which the new node should firstly connect (a bootstrap node). In the P2P-XBone, unlike application-level P2P, the new VN does not get 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 Sec. II-B.2. After that, the RD launches P2PD for the VN, 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 [9] in the X-Bone.

Node departure procedures are much simpler than node join. In the case of voluntary node departure, a P2P node only releases all IP tunnels and deletes routing table entries before departure. Even in the case of silent departure due to node failure, P2PDs on neighbor nodes can detect the node death through the keep-alive mechanism of a P2P protocol and unnecessary IP tunnels are automatically released.

### B.2 IP tunnel configuration

In the P2P-XBone, IP tunnels are dynamically created and released in a P2P fashion triggered by requests from P2PD to RD. When P2PD triggers a new IP tunnel creation, it includes the base IP address of the other end in the request. While the base IP address can be replaced with other contact information for the other end as described in Sec. II-C.1, we describe the simplest case. If an IP tunnel is already established to the VN 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, new IP tunnel creation is initiated. 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. This is a VI-specific behavior because a virtual IP address has to be set to each virtual interface on a VN. [1] 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

[1] In the X-Bone, an address block with /30 and /126 prefix length is obtained for IPv4 and IPv6, respectively, to configure a subnet for a virtual link.

pair to the P2PD to indicate that the IP tunnel is successfully established.

On the other hand, when RD is asked to tear down a existing IP tunnel, it sends a tunnel release request to 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.

### B.3 Routing configuration

P2PD also works like a routing daemon for a P2P-VI. It asks RD to set consistent routing table entries for a VN when any change in routing entries occurs in the user-level P2P daemon. By factoring out forwarding functionality from P2PD, we not only simplify the daemon but also improve the performance. In addition, by changing the P2P protocol running in the P2PD, the routing strategy of the P2P-VI can be customized with modularity. In P2P protocols, resource identifiers such as URLs (or its hash values) that do not explicitly identify the location of final destination are used for routing, which brings late-binding property to P2P systems. The P2P-XBone introduces the late-binding property to a VI as well. Since normal OSes do not support such data-based routing functionality, we support it using kernel extension at nodes participating in a VI. DataRouter [8] is an experimental implementation that supports pattern-match-based routing and forwarding at a kernel level. It extends the Loose Source Route option in IPv4 to encode a destination identifier such as a URL and a hash value. Using IP options 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 implemented system, we used DataRouter and Chord for an extended kernel module and a P2P protocol, respectively. In this case, 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.

## C. Extensions to P2P protocols

P2PD can be extended from an existing P2P protocol in a systematic fashion. As the first target of P2P protocols, we describe how to extend DHT protocols.

### C.1 Extension principle

The basic extension principle is that P2PD sends control messages to RD in response to four events in the running DHT protocol: 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 VN, respectively. The most challenging and interesting part in achieving P2PD is how to establish IP tunnels corresponding to neighbor nodes in the DHT protocol. This is not so easy because, in the P2P-XBone, the nodes that configure the IP tunnel need to know each other's base IP addresses, while 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 the P2P-VI, 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

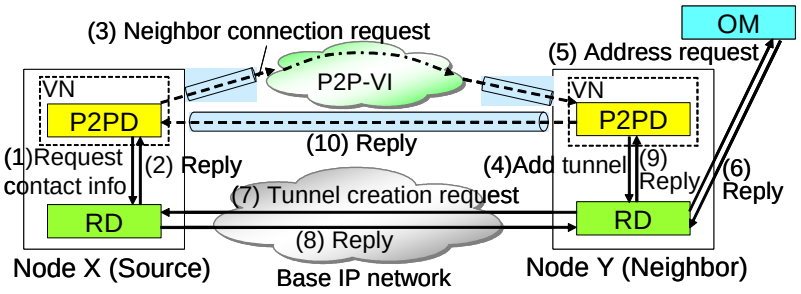



Fig. 2. Sequence of establishing an IP tunnel to a neighbor node.

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 discovered by the message itself. The receiver of this message is designated with a hash value so that the message can be delivered over the existing P2P-VI. Before actually sending the neighbor connection request message, the contact information of the node itself is obtained via the API between P2PD and 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 P2PDs not to violate the virtualization boundary. The P2PD that is either corresponding or responsible to the hash value terminates the message (Step (3)) and requests to the RD to establish an IP tunnel to the source node (*add tunnel*; Step (4)). The RD establishes an IP tunnel as described in Sec. II-B.2 (Step (5)-(8)) and responds with the virtual IP address pair of the established IP tunnel (Step (9)). Finally, the P2PD that terminated the neighbor connection request message sends a reply message back to the source node over the newly created IP tunnel (Step (10)). This procedure does not necessarily have to be extended by adding a new message to existing DHT messages. In some DHT protocols that have a message to discover appropriate neighbor nodes, it can be substituted by extending the existing message. We show an example of extending the find successor message in Chord protocol in Sec. II-C.2.

Since it takes more time for the extended DHT protocols to set a neighbor node than for existing ones due to the extra IP tunnel creation procedure, convergence performance of a P2P system would be degraded especially when P2P nodes frequently join and leave. We will show simulation results on convergence time in Sec. III-A.

### C.2 Extensions to Chord protocol

We implemented P2PD by extending Chord protocol. The Chord implementation in i3 [6] was used as a base code. We extended mainly the *find successor* message. The pseudocode of the extended functions are shown in Fig. 3.

```
//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.

The find_successor() function is run periodically to fix 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 that is called when a node receives the find successor message. If the node is the immediate successor to $n$, it terminates the message and calls

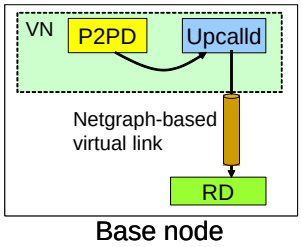



Fig. 4. Crossing VN boundary using upcall daemon

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 programs including the APIs with RD.

### *D. Crossing virtualization boundary*

There are some mechanisms, such as jails [10], vserver [11] and VMware [12], to isolate VN resources from the outer world. In P2P-VI, these mechanisms can be used for VNs to get stronger resource separation. In the architecture described above, we assumed that P2PD and RD can communicate directly with each other. However, these mechanisms would sometimes restrict such a direct communication across the VN boundary. We address this problem by introducing an Upcall Daemon (Upcalld) that relays control messages from applications in a VN to RD out of the VN through a logical internal network shared between a VN and a base node[2]. The module map is shown in Fig. 4. The internal network is created by the netgraph module [13] in FreeBSD, which is the similar scheme as used to connect virtual images in the clonable stack [14]. In this approach, P2PD sends control messages to the Upcalld, and the messages are transparently forwarded to RD at the undelying level.

### *E. QoS consideration*

The existing X-Bone supports QoS control for VIs, which includes limiting bandwidth, adding delay, etc. for each IP tunnel. In the P2P-XBone, the same QoS control mechasnisms 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 RD with other kinds of initial parameters when a VN joins the P2P-VI. This QoS configuration is applied to every IP tunnel since IP tunnel creation is performed in a P2P fashion and individual configuration for each IP tunnel is impossible.

In DHT protocols, meanwhile, some approaches to improve QoS in overlays have been proposed. While there are various approaches including data placement algorithms and transport-layer modifications [15], in terms of enhancing topology and routing, proximity neighbor selection (PNS) and proximity route selection (PRS) [16] 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 are performed based on the results in measuring QoS parameters such as latency and bandwidth to multiple candidate nodes. In application-level overlays, that is available 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 is necessary to perform the measurement over a VI. However, it is not efficient to create IP tunnels to all candidate nodes only for the measurement. In the P2P-XBone, RD provides an API for network commands like ping and pathchar although this API has not been implemented yet. The P2PD asks the QoS measurement through the API. The RD executes a corresponding network system call or command based on the type of the API and returns the results to the P2PD. Note that 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 is delivered in an opaque manner as described in Sec. II-C.1.

### *F. Security consideration*

The P2P-XBone basically provides the same security functionalities as the existing X-Bone does. That is to say, authentication and encryption are performed using SSL for communication between

[2] It may be an immediately outer VN when VNs are recursively stacked.

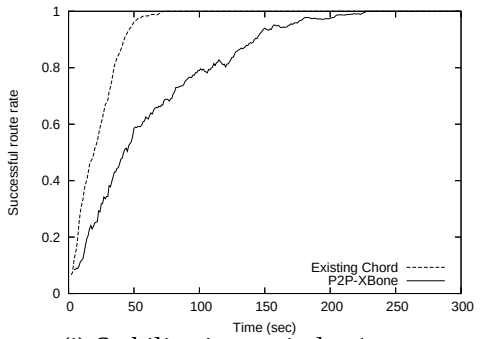
OM and RDs and using IPsec (transport mode) over IP-in-IP tunnels for virtual links between VNs, respectively. Each RD and OM have its own access control list (ACL), which is used for resource access permissions and restrictions based on user's names as well as for authentication. Such resources include number of overlays, number of tunnels, queue limits, bandwidth limits, etc. When a VN joins a P2P-VI, it is authenticated based on the corresponding ACL and is authorized to join if the resource availability meets conditions required for the P2P-VI.

While the same security mechanisms as in the existing X-Bone are available in the P2P-XBone, the P2P-based IP tunnel creation makes a difference in a way to allocate the keys for IPsec channels. In the existing X-Bone, IPsec keys are allocated for each IPsec channel (one channel for each direction) comprising a VI in a centralized manner by the OM when the VI is deployed. In the P2P-XBone, however, since IPsec channels are dynamically established in a P2P fashion, such a simultaneous key allocation is not available. Therefore, the keys are calculated by a node initiating an IPsec channel and are shared at both ends via an SSL channel between RDs.

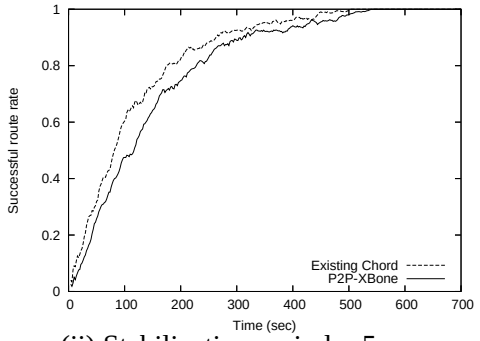
## III. Performance Evaluation

### A. Convergence time of a DHT protocol

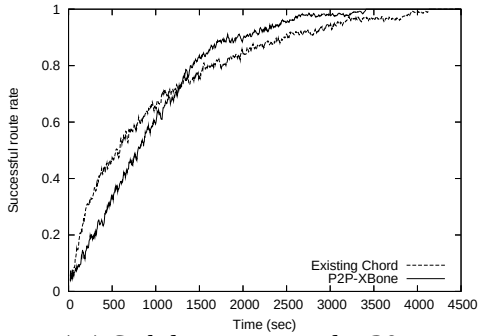
Convergence time in Chord protocol was compared between the original version and the one extended for P2PD by simulation. In this simulation, we let 100 nodes join a P2P system at the beginning of the simulation and then measured the rate of available paths in the P2P network against time by continuously generating data transfers between any two nodes. We also tested a couple values of stabilization period in Chord protocol, which is a period of the find_successor() function called. 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_baseaddr() and create_tunnel()) are involved as shown in Fig. 3. These overheads were emulated by idling for average time taken for these functions that was observed in real experiments. The simulation results in Fig. 5 show that the P2P-XBone makes




(i) Stabilization period = 1 sec




(ii) Stabilization period = 5 sec




(iii) Stabilization period = 30 sec

Fig. 5. Convergence time in Chord.

convergence time longer than the existing Chord in cases of short stabilization period such as 1 sec. However, as stabilization period is longer, convergence performance gets closer. This is because, in the case of a long stabilization period, such extra procedures as IP tunnel creation can be completed before the next stabilization routine is called. In our simulation, 5sec was long enough to get almost the same convergence time even in cases where there were more participating nodes. Of course, since longer stabilization period makes convergence time longer, an appropriate stabilization period should be determined based on system requirements. These results suggest that the

P2P-XBone is better suited to a P2P system that is comprised of relatively stable nodes and is used as an infrastructure rather than to a P2P system that is comprised of fickle end nodes and requires quick convergence.

### *B. Address server load*

As described in Sec. II-B.2, in the P2P-XBone, RD needs to request 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 various number of nodes and for various join/departure frequencies of P2P nodes to figure out how many address servers are necessary for load-balancing. In this simulation, the same extended Chord protocol as in the previous simulation was used for P2PD. 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 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 because the creation and release of IP tunnels are triggered by the stabilization routine. 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 1sec can generate huge number of address requests/releases under churn in which 1-10 percent of nodes are turned over per second. Under too much churn, it is observed that the number of messages peaks out without ever increasing. This is because node joins and departures are too frequent for stabilization routine to detect all of them.

In our preliminary implementation, the address server could transact 67 msgs/sec for UDP, 56 msgs/sec for TCP and 4.2 msgs/sec for SSL. The number of address servers required for a P2P-VI should be determined considering a tradeoff with convergence performance while only one server works well as long as node joins and departures are not so frequent.

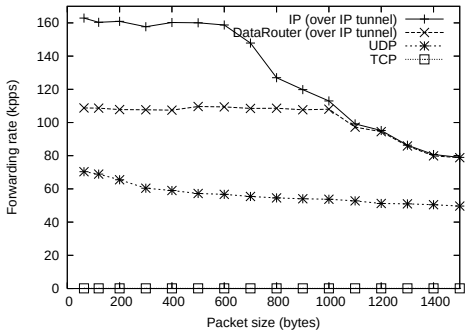
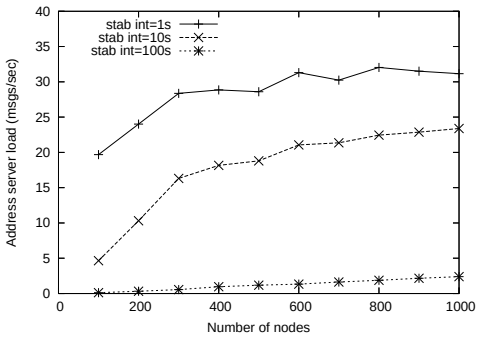


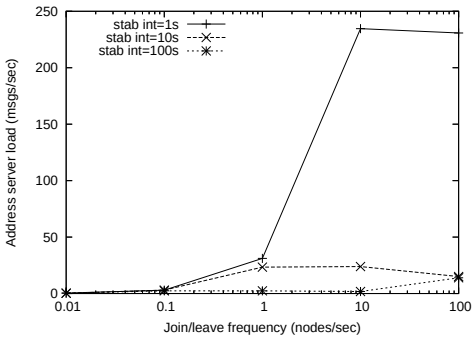

Fig. 7. Forwarding performance.

### *C. Forwarding performance*

We implemented DataRouter on FreeBSD 5.3 and measured its forwarding performance on a dual-processor 2.4GHz Xeon PC with 1Mbyte main memory. An IXIA hardware-based packet generator was connected to 64-bit/66MHz PCI gigabit Ethernet card on the PC, and the number of packets that are forwarded is counted on the generator for various packet sizes. Throughout the experiments, only 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 (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 [8] in that 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 makes almost half as fast forwarding speed as non-IP-tunneling because IP forwarding routine is performed twice for each packet. DataRouter still provides much better performance than application-layer forwarding despite its halved performance. Although our preliminary implementation provides 70 percent as much as normal IP forwarding does, the gap may possibly be reduced by optimizing the implementation. In IP and DataRouter forwarding performance, they keep almost constant forwarding




(i) The case of changing the number of nodes.




(ii) The case of changing the join/departure frequency.

Fig. 6. Address server load.

rate as long as packet size is not large enough to saturate link capacity, while IP packets with more than 600 bytes and DataRouter packets with more than 1000 bytes make throughput peak out at 850-950 Mbps. Unlikely kernel-level forwarding, 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 much less performance than the other kinds of protocols since a TCP connection was assumed to be closed for each data transfer, where the performance is limited to the rate of establishing new connections.

Through these results, it was shown that kernel-level data forwarding using IP option such as DataRouter is effective even when IP tunnels are used between nodes. We plan not only to optimize the implementation of DataRouter but also to extend it to support IPv6, which allows more unconstrained use of IP option field.

## IV. 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 and routing are considered as being independent. In other words, provisioning is performed first, and then routing is performed over the pre-provisioned topology. In that sense, existing routing protocols can be regarded as a mechanism to pick one link from multiple candidates to reach a next hop. However, P2PD does not assume a fixed set of links but dynamically creates/releases links based on the changes of appropriate neighbor nodes, which are selected out of the whole set of nodes in the network. This integrated work between provisioning and routing brings some promising properties such as load-balancing, resilience, robustness, scalability, etc. to a network system as application-level P2P does. What is taken care here is that provisioning and routing are performed at different layers, i.e., provisioning is performed under a layer where routing is performed. Therefore, an interworking mechanism between modules at different layers is needed to achieve the routing-driven provisioning. We showed how they can interwork through the dialogue between P2PD and RD in a VI's operation in Sec. II. If P2PD could be generalized as an alternative routing protocol, it would provide the unique topology management and routing strategy to other kinds of networks than virtual IP networks, such as a layer-2 network over optical pathes.

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 and IP tunnels, in which configuration is performed only at both ends of a link. The latter one includes MPLS and optical links, in which link configuration has to be performed at intermediate switches as well as at both ends. While the P2P-XBone architecture described above can be applied to the former case in a relatively straightforward way, it is necessary for

RD to interwork with such a signaling mechanism as RSVP-TE [17] to support provisioning of switched paths. In either case, 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.

## V. Related work

IP-tunnel-based overlay has been widely used to deploy virtual experimental networks as well as conventional VPNs over the existing IP network. Although the M-Bone, A-Bone [18] and 6-Bone [19] are well-known as globally-deployed testbeds, they involve static and manual deployment. Virtual Internet (VI) is generalization of such IP-level virtual networks. Recursion (i.e., stackable virtual networks) and revisitation (multiple virtual nodes in a single base node) are VI's key features. The X-Bone [2] is a system to enable automated deployment of VIs. We also have extended it to be a global infrastructure as GX-Bone [20], in which an LDAP-based registry for globally-distributed VI resources is introduced. In VPN's point of view, UMU-PBNM [21] addresses automated VPN deployment, though it focuses on standardized XML-based configuration and PKI-based authentication rather than resource discovery and node-specific script generation that the X-Bone focuses on. However, such virtual network deployment systems as X-Bone and UMU-PBNM creates/deletes VIs and VPNs simultaneously and support neither individual node join/departure nor self-organization.

On the other hand, DHT has been used for various purposes such as distributed database and augmented routing. There have been a couple of approaches which apply the promising properties of DHT to lower layer networks such as Layer 2 and 3. P6P [22] provides a scheme to resolve an IPv6 site to which a IPv6 packet is forwarded at an edge router in a core existing (e.g.,IPv4) network using a DHT lookup mechanism. Although IPv6 packets are encapsulated with IP or UDP in the core network, tunnels are established directly between concerned edge routers with no P2P routing involved. Therefore, a DHT protocol is used only for resolving base addresses for tunneling but not for organizing a virtual network. In PeerNet [23], the authors propose 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.

## VI. Conclusion and future work

We described the P2P-XBone to enable deployment of a P2P system, which was conventionally deployed at the application layer, at the virtual IP layer. In P2P-XBone, configuration of IP tunnels for a virtual node is driven by neighbor state changes in a P2P protocol (i.e., routing-driven provisioning). Also, the kernel module for data-based forwarding such as DataRouter is introduced, and routing entries consistent with those in a P2P protocol are configured to the kernel. The P2P-XBone not only provides P2P's promising characteristics such as self-organization, fault-tolerance and content-based routing to a VI but also provides higher forwarding performance and simpler implementation to a P2P system due to the availability of existing network services. Although the P2P-XBone has the overhead to control a VI, our evaluation results for Chord protocol showed that it provides almost the same convergence property in a scalable manner as long as stabilization period is not set too short. Therefore, the P2P-VI is promising as an alternative of P2P infrastructure with relatively stable nodes.

The proposed system was implemented on FreeBSD 5.3. We plan to distribute the source code for experimental use in other P2P researchers. We also plan to enhance the system, which includes more generalization of APIs between RD and P2PD to support any kinds of P2P protocols and the support of recursive P2P-VIs (i.e., stackable P2P-VI).

## References

[1] J. Touch, "Dynamic Internet Overlay Deployment and Management Using the X-Bone," Computer Networks, Vol. 36, No. 2-3, pp. 117-135, Jul. 2001.

[2] X-Bone URL - http://www.isi.edu/xbone/.

[3] I. Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," ACM SIGCOMM, 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 et al., "A Scalable Content-Addressable Network," ACM SIGCOMM, Aug. 2001.
[6] I. Stoica et al., "Internet Indirection Infrastructure," Proc. ACM SIGCOMM, pp. 73-86, Aug. 2002.
[7] S. Rhea et al., "OpenDHT: A Public DHT Service and Its Uses," Proc. ACM Sigcomm, Aug. 2005.
[8] J. Touch and V. Pingali, "DataRouter: A Network-Layer Service for Application-Layer Forwarding," Proc. International Workshop on Active Networks (IWAN), Kyoto, Japan, Dec. 2003.
[9] 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.
[10] Jails URL - http://docs.freebsd.org/44doc/papers/jail/jail.html.
[11] Linux-VServer URL - http://linux-vserver.org/.
[12] VMWare URL - http://www.vmware.com/.
[13] All About Netgraph URL - http://ezine.daemonnews.org/200003/netgraph.html.
[14] Marko Zec, "Implementing a Clonable Network Stack in the FreeBSD Kernel," Proc. 2003 USENIX Annual Technical Conf., FreeNIX track, San Antonio, TX, Jun. 2003.
[15] F. Dabek et al., "Designing a DHT for low latency and high throughput," Proc. 1st USENIX Symposium on Networked Systems and Implementation (NSDI '04), San Fransisco, CA, Mar. 2004.
[16] K. Gummandi et al., "The impact of DHT Routing Geometry on Resilience and Proximity," ACM SIGCOMM, Aug. 2003.
[17] D. Awduche et al., "RSVP-TE: Extensions to RSVP for LSP Tunnels," RFC 3209, Dec. 2001.
[18] A-Bone URL - http://www.isi.edu/abone/.
[19] 6-Bone URL - http://6bone.net/.
[20] J. Touch et al., "A Global X-Bone for Network Experiments," Proc. IEEE Tridentcom 2005, pp. 194-203, Trent, Italy, Mar. 2005.
[21] F. Clemente et al., "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), Trento, Italy, Jan. 2005.
[22] 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.
[23] J. Eriksson, M. Faloutsos and S. Krishnamurthy , "Scalable Ad Hoc Routing: The Case for Dynamic Addressing," Proc. IEEE INFOCOMM, Mar. 2004.