Dynamic Internet Overlay Deployment and Management Using the X-Bone

Joe Touch
USC / Information Sciences Institute
touch@isi.edu

Abstract[1]

The X-Bone dynamically deploys and manages Internet overlays to reduce their configuration effort and increase network component sharing. The X-Bone discovers, configures, and monitors network resources to create overlays over existing IP networks. Overlays are useful for deploying overlapping virtual networks on shared infrastructure and for simplifying topology. The X-Bone extends current overlay management by adding dynamic resource discovery, deployment, and monitoring, and allows simultaneous participation in multiple overlays. Its two-layer IP in IP tunneled overlays support existing applications and unmodified routing, multicast, and DNS services in unmodified operating systems. This two-layer scheme uniquely supports recursive overlays, useful for fault tolerance and dynamic relocation. The X-Bone uses multicast to simplify resource discovery, and provides secure deployment as well as secure overlays. This paper presents the X-Bone architecture, and discusses its components and features, and their performance impact.

1. Introduction

The X-Bone [31] is a system for the dynamic deployment and management of Internet overlay networks. Overlay networks are used to deploy infrastructure on top of existing networks, to isolate tests of new protocols, partition capacity, or present an environment with a simplified topology. Current overlay systems include commercial virtual private networks (VPNs) [27], IP tunneled networks (M-Bone [10], 6-Bone), and emerging research systems providing quality-of-service guarantees. These systems require OS and/or application modifications, restrict the number of overlays a router or host can participate in, or require manual component configuration. The X-Bone provides automated deployment of overlays, coordinates their sharing of network components, and monitors deployed overlays. The X-Bone requires no OS or application modifications and only basic IP in IP encapsulation, and uses existing implementations of dynamic routing, name service, and other infrastructure. Finally, the X-Bone is a uniform extension of the network to support overlays, and supports stacking (recursion) of overlays for fault tolerance and capacity sub-provisioning for experiments.

The X-Bone uses a two-layer tunnel mechanism, rather than the single layer in conventional overlays. It is this two-layer scheme which supports stacked overlays, as well as permitting use of unmodified applications and network services (routing, DNS, IPSEC) inside a deployed overlay. The X-Bone is the only known overlay system that integrates both IPSEC support and dynamic routing. This particular aspect of the X-Bone is covered in detail in Sections 2.1 and 3.

The X-Bone system provides a high-level interface where users or applications request DWIM (do what I mean) deployment, e.g.: *create an overlay of 3 hosts connected to each of 6 routers in a ring*. The X-Bone automatically discovers available components, configures, and monitors them.

This paper presents an overview of the X-Bone architecture, and discusses the particular techniques required to provide an IP layer overlay using existing protocols to support existing implementations of operating systems, applications, or network services. The paper builds on our earlier discussion of the coarse architecture and goals [31], by presenting the details of the X-Bone's two-layer encapsulation, which resolves support for dynamic routing, the use of a node multiple times in a single overlay, and the use of IPSEC to secure the tunnels of an overlay. This paper adds performance analysis, an extended and updated discussion of related work, and presents our vision of the utility of the X-Bone to support networking research, networking education, dynamic service deployment, and fault tolerance.

[1] This work is partly supported by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-98-1-0200. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government.

1.1. What is an overlay?

An overlay network is an isolated virtual network deployed over an existing network. It is composed of hosts, routers, and tunnels. Tunnels are paths in the base network, and links in the overlay network. Hosts are packet sources or sinks, and routers are packet transits, as in conventional networks. Individual components (routers or hosts) can participate in more than one overlay at a time or in multiple ways (router, host) in a single overlay. Figure 1 shows an IP network (left); on that network, the X-Bone can deploy a ring (center) or star (right), by using various subsets of the nodes of the base network, connected by a set of tunnels. These tunnels determine the overlay topology, and may traverse multiple links in the base network, or a single link multiple times.

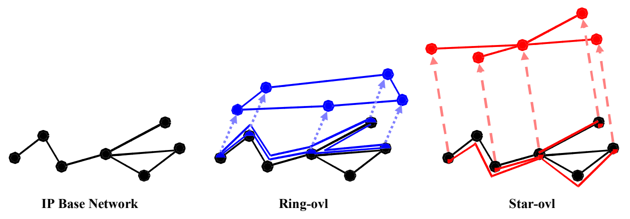


Figure 1. A ring (center) and a star (right) overlay deployed on a base network (left)

Overlays have three primary uses: containment, provisioning, and abstraction. Containment is the ability of an overlay to restrict the visibility of its contents. Tunneling encapsulates the packets of new protocol so it can be tested in a controlled environment. Containment was one of the first uses of overlays in the early 1980's [20], and motivated their re-emergence in the early 1990's for the M-Bone and later 6-Bone [10]. Tunnels allow incremental deployment, where (primarily) routers lacking new protocol capabilities can be skipped over (or through), avoiding the need for contiguous availability.

Provisioning uses reservation of components and capacity along tunnels to provide service guarantees to the overlay. Provisioned overlays can be used during emergencies to create virtual infrastructure when it is not feasible to deploy new physical resources. They can also be used to limit the scope and impact of network experiments, e.g., to nominal use of surplus capacity.

Abstraction is a new use of overlay networks. Both provisioning and containment imply the interim the use of overlays that are supplanted by advanced hierarchical reservation in the former case, or more sophisticated dynamic services deployment in the latter [28]. In these cases, overlays are a way to provide such capabilities without requiring contiguous deployment; once a new protocol or service is ubiquitous, tunnels (and thus overlays) can be avoided. However, abstraction remains a useful tool for education (networking classes), deploying testbeds, and simplifying applications. For example, a single lab can support a large number of concurrent experiments, each using a different topology. A testbed can be configured using a graphical user interface, in *do what I mean* style. Applications can request a deployed topology (e.g., ring) without needing to incorporate network management. In each case, manual intervention by a network manager is avoided, and applications and tools can be simplified.

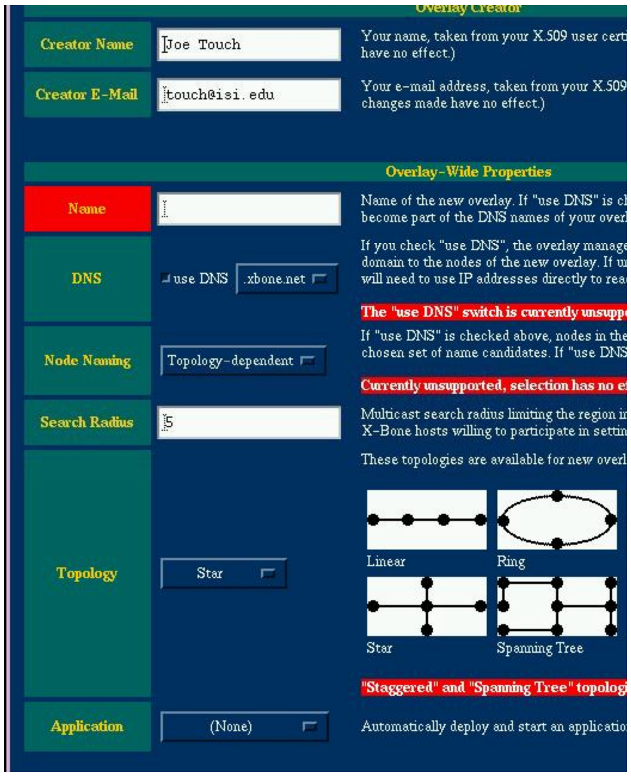
1.2. Deploying an overlay

Conventional overlay deployment is a multi-stage process, involving manual intervention at every step. Components in the network (routers, hosts) are selected according to some criteria, e.g., operating system, protocol capability, or permissions. The desired topology (e.g., ring) must be mapped to the available components and parameters such as addresses, network masks, and routes determined. For each component, secure remote access is required, typically via SSH/telnet, and then each component is manually configured. This includes setting tunnel endpoints, configuring interfaces, setting link encryption or authentication keys, and configuring routes. Each of these steps is manual, often requiring out-of-band communication (telephone, e-mail) to locate available resources or initiate access. Each of these steps also requires external mechanisms for coordination, such as a reservation web page or e-mail system.

Once an overlay is deployed, there is no assurance it remains available. Both in-band (over the overlay) and out-of-band (in the base network, or via telephone or e-mail) methods may be required to confirm the state of the overlay. Current overlays lack mechanisms for monitoring, for repairing an erroneous component, or for signaling for attention. Modern dynamic routing protocols are typically not available within an overlay, so they are susceptible to single-tunnel failures. When an overlay is no longer of use, it must be dismantled. This is requires a tedious recapitulation of installation steps in reverse.

The key problems with the current method of overlay deployment are manual intervention, the excessive need for out-of-band communication, the lack of monitoring, and the necessity of separate dismantling procedures. The X-Bone is designed to reduce deployment effort, involving manual interaction only at the initial request phase, e.g., in a graphical user interface, or programmatic API. Resource discovery is automatic, such that any sufficient available resources can be used to satisfy a request. Resource sharing is managed so many overlays can simultaneously share the use of a single component. An X-Bone overlay can use features of the existing Internet, including dynamic addressing (DHCP), dynamic routing, and diagnostic tools (*traceroute*, *ping*, etc.) without modification. The X-Bone also supports existing operating systems and applications, without modification, provided they support basic IP in IP encapsulation.

The X-Bone extends the current Internet network architecture to include support for overlay networks. It provides stackable overlay networks, where control can be via a web-based GUI (Figure 2) or a program-controlled API. An X-Bone overlay is an integrated end-to-end solution, including host configuration, router configuration, and support for DNS.



Overlay Creator

Creator Name	Joe Touch	Your name, taken from your X.509 user cert have no effect.)
Creator E-Mail	touch@isi.edu	Your e-mail address, taken from your X.509 changes made have no effect.)

Overlay-Wide Properties

Name		Name of the new overlay. If "use DNS" is c become part of the DNS names of your over
DNS	use DNS .xbone.net	If you check "use DNS", the overlay manag domain to the nodes of the new overlay. If u will need to use IP addresses directly to rea The "use DNS" switch is currently unsupp
Node Naming	Topology-dependent	If "use DNS" is checked above, nodes in the chosen set of name candidates. If "use DNS Currently unsupported, selection has no e
Search Radius	5	Multicast search radius limiting the region i X-Bone hosts willing to participate in settin
Topology	Star	These topologies are available for new overl "Staggered" and "Spanning Tree" topolog
Application	(None)	Automatically deploy and start an applicatio

Figure 2. X-Bone graphical user interface

This paper presents the X-Bone and discusses its components and features, their performance impact, and the effect of overlays on the Internet architecture. The architecture section presents the X-Bone's components and features, including its use of two-layer tunnels to avoid OS and application customization and to support recursion. The evaluation section discusses the system's new capabilities, security, and performance. Related efforts and future work are discussed, including extensions for fault tolerance and the merging and splitting of deployed overlays.

2. Architecture

The X-Bone is a distributed system composed of Resource Daemons (RDs) and Overlay Managers (OMs) , with a graphical user interface (GUI) and a more direct API. These components are shown in Figure 3.

OMs deploy overlays. A user creates an overlay by sending a request to an OM, either via a web-based GUI (Figure 2) or by sending a message to the OM API. Each overlay is coordinated by a single OM; large overlays are created by divide-and-conquer, where a single OM forks sub-overlay requests to other OMs. Fault tolerance can be achieved by replicating state in multiple backup OMs.

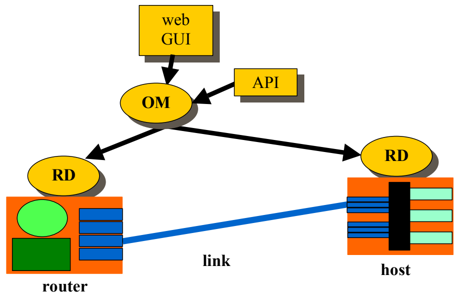


Figure 3. X-Bone architectural components

An OM creates an overlay in phases, using multicast to discover available resources and TCP/SSL [16] to configure and monitor resources. The overlay request is translated to an invitation, and the invitation is multicast using UDP. An invitation indicates a set of simple conditions, e.g., a specific operating system, bandwidth requirements, etc. Invitations currently fit in a single UDP packet; where they do not, IP's automatic fragmentation and reassembly is utilized. Invitations are repeated with increasing TTLs until a sufficient number of invitees respond, or until a preset search limit is exceeded: i.e., an expanding ring search (Figure 4) [21].

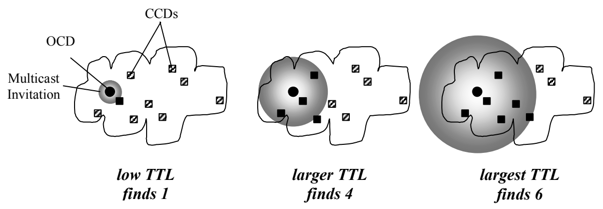


Figure 4. Resource discovery using increasing-TTL multicast invitations

RDs are daemons that configure and monitor the resources of routers and hosts. RDs listen for multicast invitations, and respond when their available resources and permissions match. Responses are in the form of TCP/SSL (X.509 encrypted) connections back to the source OM, where each RD indicates its particular capabilities (Figure 5). The OM selects an appropriate from among the responding RDs. The OM determines configuration information, such as tunnel endpoint addresses and routing table entries, and sends specific configuration information to each RD. Once an overlay is deployed, the TCP/SSL connections are released and the overlay is up. Subsequent overlay actions initiated by the

OM include keep-alive pings, liveness and status requests, and modifying or dismantling configurations.

TCP/SSL [16] is used for secure configuration to take advantage of TCP's reliable channel, and reduce the number of different security schemes required. The X-Bone uses a web-based GUI; web browsers already support SSL, so the user's request is secure on the path to the OM. For simplicity, the same mechanism is used between the OMs and RDs. Other schemes, such as PGP, would require multiple solutions.

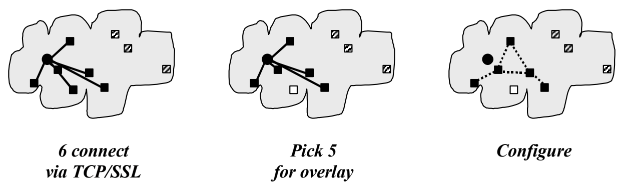


Figure 5. Responding to invites, selecting, and configuring the overlay

This architecture utilizes a single, well-known multicast channel for invitation announcements, and separate reliable channels for configuration and monitoring. It is based on the multicast announcements in M-Bone teleconferencing; in fact, the X-Bone deploys an overlay as if it were a teleconference between its OM and the RDs of its router and host components.

2.1. Functions

There are several key functions performed by the X-Bone. Primary among these is resource discovery, in which an expanding-ring search over a well-known multicast address [21] replaces rendezvous or registry systems. The X-Bone's invitation-based system promotes privacy and security, because participating components (hosts and routers) need not publicly post their availability or configuration. The invitation itself is public or encrypted to be private to a pre-arranged subset of components. Components each decide for themselves whether to respond, based on a match between their capabilities, availability of resources, and permissions.

The X-Bone uses two-level tunnels (Figure 6). Each overlay IP packet is wrapped in two additional IP headers. The innermost (overlay) header indicates the endpoints in the overlay. The next layer acts as a link layer in the overlay, and indicates the endpoints of the tunnel over which the packet is currently traversing. Overlay link addresses are a separate set of IP addresses, also internal to the overlay. The final header indicates the tunnel endpoints in the base network. The base network can itself be an overlay, providing stacking (recursion).

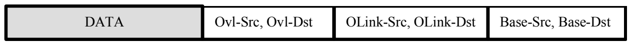


Figure 6. Double tunneling results in three headers

The additional tunnels are required to allow multiple tunnels between two components, even within the same overlay. Such doubly connected components are useful to emulate systems with larger numbers of components, i.e., 50-node rings simulated by using 10 router nodes. The additional layer also permits the use of multicast and dynamic routing algorithms inside the overlay, because such systems effectively operate on the link IP layer. Without that layer, it would be impossible to decouple intra-overlay routing from base-layer routing.

The two layers of the encapsulation change at every overlay hop, as shown in Figure 7. Note the hosts, indicated by their single overlay interface and overlay link addresses, and the router, indicated by its pair of overlay interface and overlay link addresses. Each component is shown as using a single, canonical base address for base-layer routing; this can be relaxed for multihomed systems. The X-Bone requires that routers are multihomed inside the overlay, according to the standard Internet practice.

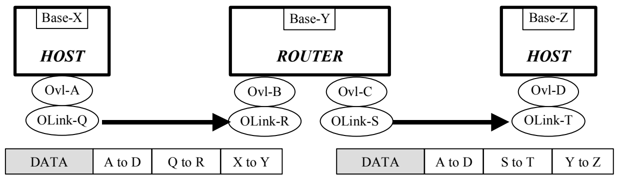


Figure 7. A single packet traverses the overlay – modifying both outer IP headers at each hop

The X-Bone is currently implemented using separate IP address spaces both for the overlay endpoint addresses and the overlay link addresses. The use of separate address spaces effectively encodes the overlay identifier inside the IP addresses, allowing conventional dynamic routing and forwarding at the routers, and conventional IP demultiplexing at the destination host. This can be relaxed to allow address reuse, provided the decapsulation steps in routers (for forwarding) and end hosts (for demultiplexing) keep sufficient context of the discarded layers of IP headers. Current implementations discard this state, requiring global addresses. Overlay addresses can be reused among overlays that do not overlap, as can be determined during the negotiation process. These issues are covered further in Section 3's discussion of IPSEC.

The OM emits heartbeat pings to refresh the state of the RD components. When a RD no longer hears from an OM, all overlays of that OM are released from the RD state. Both RD and OM state are kept on disk, and reloaded after reboots or restarts.

3. Features

The X-Bone exhibits unique overlay capabilities, largely due to a combination of its focus on IP, and the use of two IP in IP encapsulation tunnels for each overlay link. The X-Bone can be deployed on unmodified operating systems, requiring only support for IP encapsulation. For FreeBSD, this requires the KAME IPSEC [18] or CAIRN IPSEC patches [4]. Alternately, DIVERT sockets in standard FreeBSD can be used in conjunction with the X-Bone project's user-level IP in IP encapsulation daemon, *ip-tun* [9]. Linux supports similar tunnels, and support for these variants is included in the current software release (http://www.isi.edu/xbone).

The X-Bone allows applications to be used unmodified inside overlays, by virtue of its use of two-layer IP encapsulation. On a host, an overlay is selected either directly by IP address, or indirectly by overriding the DNS resolver parameters of a process environment. A deployed overlay includes dynamically configured DNS entries for variants of the names of the participating components. For example, if *blue.abc.com* belongs to an overlay called *apple*, then a DNS near the OM (part of the X-Bone deployment) is updated with the name *blue.apple.diode.net* as part of the overlay configuration. Both FreeBSD and Linux support the use of per-process overrides to the resolver default suffix; setting this parameter allows the name *blue* to resolve to either *blue.abc.com* or *blue.apple.diode.net*, depending on the process setting. Different processes on the same host can easily refer to different overlays, even using the same endpoint names. An example of how the overlays and base network from Figure 1would appear is shown in Figure 8. Base component names (here only hosts are shown named) remain the same; the DNS suffix in each window differs. A standard network mapping utility can thus show different network views in different windows.

The X-Bone's dual-layer tunnels allow existing dynamic routing and network diagnostic tools to be used inside an overlay, transparent to the base network. This has been used to deploy dynamic routing across non-cooperating administrative domains, where only the hosts involved need participate in the routing algorithms. This has been demonstrated in the X-Bone system, and dynamic routing using RIP (via *gated*) and multicast (*mrouted*) are supported inside deployed overlays.

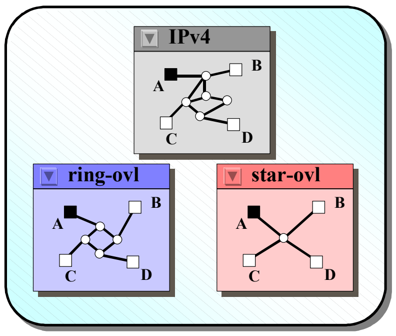


Figure 8. User views of a network mapping utility; different views in different windows

The use of multicast for invitations provides privacy and avoids the need for preconfiguration of the OMs or RDs. A single channel can be used for all invitations, because invitations are not expected to produce significant traffic. Resources in the current implementation are centered on the OM; alternately, loose source route [22] or an explicit proxy to a remote OM can center the invitation wherever useful. Invitations can be general (5 routers and 15 hosts), system or capability specific (FreeBSD/KAME, IPSEC/3DES), permission-based (userid=jones), or specific down to the site (loc=blue.abc.com). Topologies can be selected from a generic set (ring, line, and star are currently implemented), or provided by a netlist to the API.

The X-Bone also supports IPSEC in the overlay [19]. Again, the use of two layers of IP encapsulation simplifies the architecture. The overlay IPSEC parameters are attached to the overlay link IP header, according to the IPSEC protocol. This allows separate IPSEC associations to exist between base network hosts (or in the underlying overlay, if the base is itself an overlay), as well as allowing IPSEC end-to-end by applications in the overlay. It also allows applications to benefit from a secure overlay network without requiring specific application support for IPSEC, assuming the components (hosts, routers) in the overlay are reasonably secure.

IPSEC in an X-Bone overlay is configured out-of-band, via the OM using TCP/SSL. Keys are exchanged over these secured channels, rather than via IPSEC key exchange protocols. The X-Bone uses explicit key distribution for simplicity; IPSEC key exchange mechanisms are not widely available, and are currently in a high state of flux. The X-Bone uses transport mode IPSEC on an IP in IP encapsulated overlay link packet, then wraps the result with the outermost base layer IP in IP encapsulation. This is simpler to manage, because tunneling is independent of whether IPSEC is enabled on a particular overlay hop.

Dynamic routing in an overlay network can interfere with the use of IPSEC to secure overlay links [30]. IPSEC authenticates or encrypts links in an X-Bone overlay. IPSEC can interfere with forwarding decisions in overlay routers, however. Consider a packet P entering router A, destined ultimately for host Z (Figure 9). There are two possible paths to Z, one through B, the other through C. The B path begins with an overlay link keyed with K1; the C path, with K2. Per-link keys are required for robustness, to avoid needlessly compromising keys. In an implementation where IPSEC processing precedes forwarding decisions, Router A must decide which key to use (K1 or K2) before it has decided which path to take (via B or via C). Some of the forwarding decisions (i.e., routing table) must then be represented in the IPSEC rule base, so that packets destined for Z are tagged to use K1. The IPSEC rules must reflect the current routing table, imposing configuration and synchronization effort on the routing protocol implementation. Current routing protocols do not support synchronous IPSEC rule updates.

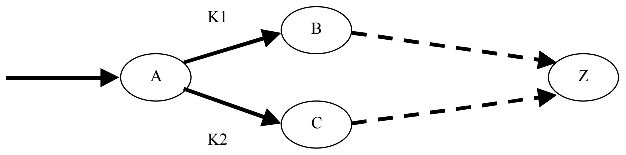


Figure 9. Dynamic routing interferes with per-hop IPSEC

IPSEC relies on policy databases to determine key usage and requires that keying precedes forwarding [19]. This is not consistent with the use of per-hop keys and dynamic routing protocols. An alternative to binding keys to rules is to bind keys to virtual interfaces, as in the NIST Linux implementation. Keys are bound to links by conventional routing rules, rather than policy-based rules in a separate key database. This allows the key decision to come after forwarding. A forwards via B by using virtual interface V1; everything from V1 is encrypted with K1, then sent to B Figure 10.

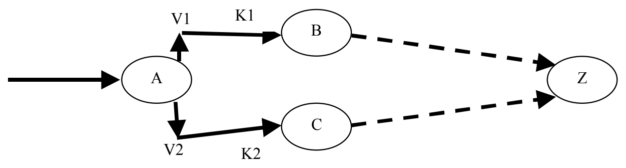


Figure 10. Binding keys to virtual interfaces allows per-hop IPSEC

The X-Bone takes advantage of this scheme, even in systems that bind keys to IPSEC rule bases. In the X-Bone, tunneling is decoupled from keying, and tunneling is always performed first [30]. E.g., V1 performs the link-layer encapsulation, and K1 would add the link key. This allows the IPSEC rules to remain static, as in "encrypt everything wrapped in this overlay link header." Dynamic routing algorithms update the routing table, and determine which virtual interface, and, by consequence, which key. The X-Bone is the only known overlay system that integrates both IPSEC support and dynamic routing.

This example highlights the issue of lost context. When an encapsulated packet is received, it is unwrapped, and forwarded by the router or demultiplexed to endpoint connections in the host. Forwarding and demultiplexing decisions do not depend on the state of the additional encapsulation headers; this state is discarded as it is removed, so is not available anyway. This means that the interior packet addresses must be globally unique, unless host kernel and router firmware modifications are made to support retaining this state. Uniqueness is per-component. Addresses can be reused on overlays that do not share components, i.e., that participate in both overlays. Routers that provide tunneling only (i.e., intermediate on the tunnel path) do not count as part of an overlay.

4. Performance

The performance of the X-Bone has been measured in a lab testbed using 300 MHz Pentium II PCs running FreeBSD 3.2 with KAME IPSEC extensions, FreeBSD 2.2.5 with CAIRN IPSEC extensions, and Linux RedHat 6.0 with NIST IPSEC extensions. These PCs were connected using a private, switched 100 Mbps Ethernet. The primary focus of overlay deployment is connectivity, but it is useful to consider the performance of this implementation using untuned tunneling and IPSEC code.

The primary performance impacts are an increase in per-hop latency and a decrease in end-to-end bandwidth. Because modest (300 MHz) hosts and a modest network (100 Mbps) were used, processing overheads were not measured at this time, though we will have sample numbers by publication. The X-Bone's two-layer tunneling adds 30% to per-hop latency and decreases bandwidth similarly, compared to the base network. Compared to M-Bone-style single-layer tunnels, the X-Bone's additional tunnel layer adds 6% to the per-hop latency, and 20% to the end-to-end bandwidth decrease. Limited processing capability of our current hosts is the likely reason for the substantial bandwidth impact.

Figure 11 shows the per-hop latency increases, measured using ICMP ping messages. The first three bars (from the left) indicate the per-hop latency in the base network with a single-layer tunnel, and with the X-Bone's two-layer tunnel. Subsequent pairs compare the base network and two-layer solutions for IPSEC authentication (AH), encryption (ESP), and combined (AH/ESP) processing. Where IPSEC is used, it is performed on only one tunnel layer.

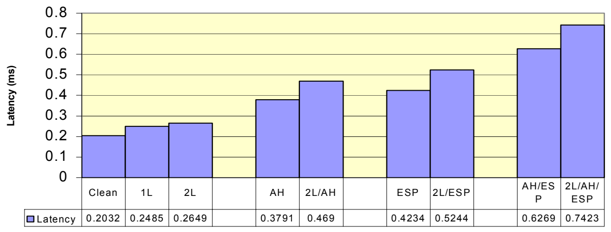
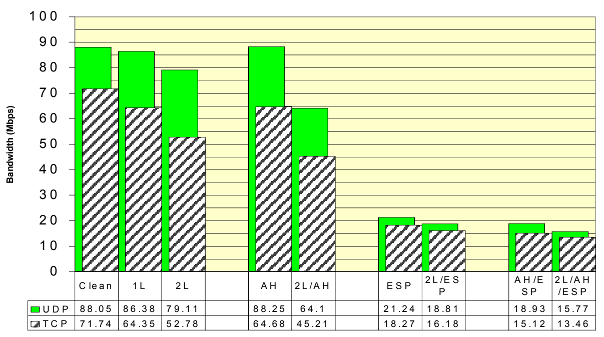


Figure 11. Per-hop increases in latency using single and two-level tunnels

Figure 12 compares end-to-end throughput of TCP and UDP streams, in similar three- and two-way comparison. Note that the effects of multiple tunnels are masked where encryption is used, because encryption processing dwarfs the overhead of additional encapsulation and decapsulation processing.



	Clean	1L	2L		AH	2L/AH		ESP	2L/ESP		AH/ESP	2L/AH/ESP
🟩 UDP	88.05	86.38	79.11		88.25	64.1		21.24	18.81		18.93	15.77
▨ TCP	71.74	64.35	52.78		64.68	45.21		18.27	16.18		15.12	13.46

Figure 12. End-to-end decrease in throughput using single and two-level tunnels

The X-Bone also decreases the effective MTU (maximum transmission unit) or packet size in the overlay network. On multihop paths in the base network an MTU of 576 bytes is required by IP, of which 20 bytes are IP and another 20 bytes by the transport layer (typically TCP or UDP). This leaves 536 bytes for application data, although many implementations round this down to a power of two (512 bytes), for efficiency. This leaves a slack of 24 bytes that can be used by additional encapsulation layers without significant impact on applications. Note that these limits can be overcome using path MTU discovery, but successful discovery depends on contiguous deployment of P-MTU, not currently the case.

Each layer of tunneling adds an additional IP layer, which consumes 8-20 bytes, when using minimal-encapsulation [26] or standard IP in IP encapsulation tunnels [25]. The X-Bone uses the standard IP in IP tunnels in the FreeBSD/CAIRN, FreeBSD/KAME, and Linux/NIST stacks. As a result, our effective MTU is 576-40-40 = 496 bytes. The additional two headers increase packet overhead by 7%. More importantly, they reduce the effective application MTU to a non-power-of-two, which may have more significant effects. If IP in IP encapsulation were replaced with minimal encapsulation IP, the overhead would be cut nearly in half, and 512-byte MTUs would be supported again.

5. Related Work

The X-Bone is related to other overlay networks and overlay deployment systems, as well as to the abstraction of network components. Other manually deployed overlay systems include the M-Bone [10], which first used IP in IP encapsulation for tunnels and the more recent 6-Bone (for IPv6), both used to test new protocols. The M-Bone was developed to incrementally deploy a testbed for multicast IP. Not all systems under test were directly connected; tunnels were used to avoid this need for contiguity. The first M-Bone tunnels used IP's loose source route (LSR) option [22], but this was replaced by IP in IP tunneling [25], because the former is more computationally intensive and requires contiguous deployment of loose source routing in intermediate routers along a tunnel path, as noted in Section 4. IP in IP tunneling presents a conventional IP packet to intermediate router hops, so takes the *fast path*, and does not stress these routers' implementation of infrequently used options. IP in IP tunneling requires new software at both tunnel endpoints, whereas LSR needs encapsulation software only at the source end of the tunnel, but also relies on proper option processing at all intermediate steps, as well as at the destination end of the tunnel.

Both M-Bone and 6-Bone are manually-deployed overlays, requiring network managers to design, deploy, and monitor network configuration. There are a number of systems for automatic deployment of overlays as well. Argonne's MORPHnet [1] is an overlay system that supports virtual networks at all layers, from virtual physical, to link, to network, on up to application. MORPHnet was designed for use in supercomputer networks, where performance requirements necessitate low- and multi-layer solutions. CRATO's Supranet [8] extends this multi-layer notion with multi-layer optimizations. Columbia's Virtual Active Networks (VANs) are part of the Netscript project [32] and deploy link-layer virtual networks. These systems focus on multi-layer or low-layer virtual network support; the X-Bone [31] has more in common with Cornell's VON [23], focusing on IP. The X-Bone's IP focus supports stackable networks and the use of standard network protocols and applications within an overlay. It differs from application solutions, e.g., Yallcast [12], and pseudo-network overlays, e.g., the A-Bone [2]. In both cases virtual networks exist inside application environments interconnected by UDP or TCP tunnels.

Several overlay systems focus on QoS support for overlay deployment, e.g., Supranet and DARWIN [7], the latter of which includes a subcomponent called VNS [8], addressing dynamic overlay deployment. The X-Bone does not require QoS support, though there are hooks to use standard QoS mechanisms, such as RSVP [33] and tunnel mode RSVP [29], where available.

The X-Bone focuses on end-to-end deployment of an entire overlay, including end host configuration, router configuration, and network services such as DNS. This is similar to the goal of VONs, but differs from the partial deployment of Darwin/VNS [7], Detour [24], and VANs [32]. In Darwin, the overlay is deployed among a set of routers via tunnels, and end hosts are attached via filter-based translators, stationed upstream of the end-hosts. This supports unmodified end-host applications, even though VNS requires OS modifications in the remainder of its deployment. Detour [24] deploys individual tunnels to override inefficient or inoperative routing, rather than deploying an entire overlay network. VANs [32] deploy only links, inside an active networks layer. Even VONs

[23], though end-to-end, do not address the issue of automation of the deployment process. Both the X-Bone and VONs allow access of different overlays via dynamic, partitioned namespaces, but in VONs the namespace is per-login, whereas in the X-Bone it is per-process.

The X-Bone differs from commercial VPNs [27] by supporting components being shared by multiple overlays, and multiple times in a single overlay. VPN components are typically a member of only one VPN at a time, and VPN deployment is an increment to an existing, deployed network. VANs and VONs support components shared in multiple overlays, but do not address single components appearing multiple times in a single overlay. This latter use enables testbeds to emulate larger networks, such as 5 routers emulating a 50-router ring, enabling large-scale experiments using limited resources.

The X-Bone avoids OS and application modifications, and supports the use of existing dynamic network services, such as routing, multicast, and nameservers, inside the overlay. Supranet, MORPHnet, VONs and VNS require OS modifications for custom tunneling and QoS. VANs push these modifications to an application layer emulation of the OS, in an Active Networks environment. In each case some network services can be reimplemented to operate within the overlay, though only VONs and VANs purport to support dynamic routing.

The X-Bone uses multicast for resource discovery, avoiding explicit configuration. Yallcast [12] and some other overlay systems (e.g., USC/ISI and SRI's A-Bone [2]) rely on central registries or rendezvous points, which must be configured explicitly. Zeroconf [14], a recent IETF effort at specifying zero-configuration network deployment, focuses on the base network, and relies on broadcast. This limits Zeroconf to a single LAN, as with BOOTP and DHCP protocols. The X-Bone uses multicast, which is not limited to a single network.

The X-Bone presents an IP overlay built on an IP base network, and is intended to be recursive, or stackable. Stackability is a feature of the X-Bone and VONs, though VONs use of global identifiers, e.g., the IETF's VPN ID [11] limits the scope of the recursion. The X-Bone differs from the IETF's VPN and VONs by allowing non-overlapping reuse of global addresses, rather than requiring VPN ID [11] and protocol modifications [13] to support their use. This contrasts to inherently single-level solutions, such as the M-Bone and A-Bone, where recursion is not feasible due to the tunneling mechanisms used. Genesis supports a retrograde variant of recursion – deploying parent overlays, where each parent can *spawn* multiple child overlays. Genesis goes up two levels, and back one, allowing testbed overlays to deploy subset overlays to coordinate and separate multiple concurrent experiments in each testbed. The X-Bone supports arbitrary recursion, due to its use of two-level tunnels, thus allowing testbeds on testbeds ad infinitum.

The X-Bone uses two-level tunneling and global address space to abstract its hosts and routers. Address partitioning allows a single routing table to contain non-interfering entries for multiple overlays as well as a base network. Preprocessed routing configuration scripts provide partitioned dynamic routing and multicast without OS or router modification. Competing proposals support partitioned routing tables without modification, including policy routing, multi-table *gated* and *mrtd* host-based router routing protocol systems. The X-Bone explicitly configures both ends of a tunnel; this can be replaced with single-ended tunnel deployment mechanisms, such as Ascend's Tunnel Management Protocol (TMP) [15]. Future versions of the X-Bone are expected to replace scripting with advanced variants of automated configuration, such as MPLS, DHCP, and SNMP. MPLS [5] will allow fine-grained control over the path a tunnel uses. DHCP will allow standard configuration of an end-host, but must be modified to allow the DHCP server to initiate the reconfiguration of the host, rather than supporting only client-initiated transactions. SNMP is a reasonable replacement to our explicit scripting mechanism, but was not necessary for a proof-of-concept.

The X-Bone supports security at multiple levels, allowing encrypted or authenticated invitations with private response, using TCP/SSL for configuration, and supporting existing IPSEC to secure the deployed overlay links. IPSEC is supported, but not strictly required. There may be cases where, for performance reasons, a secure tunnel is neither required nor desired, such as for lab testbeds. Other overlay systems do not address security, or use custom integrated packet security, e.g., VONs [23].

The X-Bone system shares much in common with the IETF's emerging VPN framework, and with the goals of VONs. All three abstract network infrastructure for the purposes of simplicity, scalability, provisioning, and containment. Like VONs [23] (and Detour [24]), the X-Bone supports fault tolerance. The X-Bone uniquely uses the ability to deploy existing dynamic routing protocols to support fault tolerance within an overlay, and its capability to support stackable overlays to support more advanced fault tolerance (see future work, below).

Finally, the X-Bone is currently implemented and available, and is being deployed in a number of regional and national testbeds, as well as used in several networking courses. Many other proposals, such as MORPHnet [1], VON [23], and Genesis [6] are only at the planning stages thus far.

6. Current Status and Future work

We released our first distribution of the X-Bone, including source code, in Feb. 2000; it’s most recent release is v1.3 (Aug. 2000). The latest distribution supports FreeBSD 3.4/KAME, FreeBSD4.1[2], Linux RedHat 6.0/NIST (kernel 2.2.5) (later versions also supported); indicated patches (KAME, NIST) are optional and required only to support IPSEC (excepting FreeBSD4.1). FreeBSD pre-4.x requires use of available DIVERT sockets together with a user-level IP in IP tunnel daemon (ip-tun [30]), developed as part of the project. Our current release uses non-encrypted multicast invitations, and configures components securely via TCP/SSL. It supports dynamic DNS with per-overlay namespaces, and currently requires configuration via the GUI from among a fixed set of topologies, including ring, star, and line. The current release also supports static intra-overlay routing, either via conventional static routes or via explicit entries in the *gated* configuration of the dynamic routing of the base network. The X-Bone has multiple levels of logging, and includes heartbeat refresh and timeout, as well as state recovery on restart or reboot. Support for dynamic intra-overlay routing via gated using the RIPv2 protocol, and support for intra-overlay multicast have been developed, and are currently undergoing tests for robustness.

Future releases over the next year are expected to include richer topologies and the explicit API. A system for deploying applications, e.g., Squid proxy cache or *anetd* (Active Nets) daemons, is also under current testing. We are also developing an explicit recursive X-Bone, deploying OMs and RDs inside an overlay. Support for other dynamic routing systems, namely *mrtd*, as well as an interface to resource reservation (RSVP) is underway. The X-Bone's out-of-band configuration will eventually be replaced with emerging standards for in-band control, such as tunnel configuration (TMP, including MPLS tunnel pinning), IPSEC key exchange, dynamic DNS, and SNMP. Security of the invitations is also under investigation, including authentication, privacy, and traffic confidentiality of invitation activity.

The X-Bone is currently used for networking research and networking education. Various research groups are using the X-Bone to facilitate concurrent overlapping virtual testbeds on a shared, interdomain infrastructure (CAIRN, including a total of 30 nodes). The X-Bone is being augmented to assist in the deployment of the A-Bone, and is being deployed in several advanced showcase testbeds. It is also being used for USC's graduate networking laboratory class (a 24 node lab), for overlapping concurrent student experiments.

The X-Bone has been especially affected by the dearth of support for multihoming. Hosts in an overlay system are necessarily multihomed [3], belonging to both the base network and perhaps several overlays. Multihoming requires context-sensitive demultiplexing, such that daemons not attach to every incoming packet addressed to a particular protocol's port. While this is supported in current operating systems, most protocol daemons are not written to bind to a subset of addresses. In addition, applications need control of the source IP address of a packet, i.e., to indicate which of a host's multiple addresses is to be used as the source (not currently implemented). Multihomed hosts often require support for an internal, virtual router, such as support for dynamic routing protocols, support for proxy ARP, etc.

Routers are similarly *multirouted* (our term for a multihomed router), needing similar partitioning. In a router, this translates to context-sensitive forwarding, and context-sensitive routing algorithms. Packet processing and routing packet exchanges need to be predicated on the address of the incoming packet and its interface. In both routers and hosts this context is both address, and overlay identifier specific. For overlays, this means that the IP decapsulation must retain portions the outer headers, to be used as context for further processing. Other host services, such as DNS resolution, require this context to differentiate namespaces among overlays. Our future work includes these extensions.

We are investigating extensions for fault tolerance and optimization. It would be useful to avoid deploying multiple overlays over the same physical link, or to provide redundant links within a single overlay. It would also be useful to map requested ring overlays onto rings in the base network, somewhat matching topologies. Strictly, such overlay optimizations are graph embedding problems, which are difficult to optimize efficiently. The Internet's strict layering further complicates redundancy detection; even purchasing separate physical links from different networks providers can result in unintentional fate-sharing. We are investigating protocols for voluntary labeling to enable automated fate-sharing detection.

We are investigating many of the features described in the architecture as possibilities, such as proxy-based resource discovery, and divide-and-conquer deployment. These features determine the scale of an overlay that the X-Bone can deploy; the current system has been tested for tens of nodes, and while larger scale tests are underway, it is not realistic to expect a single OM to coordinate thousands or tens of thousands of nodes. A related issue is fusion and fission, the ability to split an existing overlay, or merge two overlays into a single meta-overlay, which are useful for policy-based coordination, where organizations create their own overlays and subdivide them for internal use (ala Genesis), or merge them for inter-organization testbeds (e.g., CAIRN).

Fault tolerance is a specific focus of future development. The X-Bone's support of stackable overlays supports dynamic relocation of a running overlay, without renumbering that overlay, which can remap a faulty underlying overlay to a working overlay. Consider our

[2] FreeBSD 4.0 has a bug in the IPSEC *setkey* command that affects the X-Bone. It is fixed in FreeBSD 4.1 and higher.

first example, of a base network on which various overlays are deployed (Figure 1). The X-Bone can deploy stacked overlays, such as three ring networks on the base network, and a star on one of those rings (Figure 13). When a fault is detected in one ring, the star can be remapped to a different ring. The challenge is deploying multiple rings that are known not to share physical resources. The X-Bone's layering provides a level of indirection to IP addressing in the star overlay, which allows it to be renumbered with respect to the base network, without renumbering within the star (the virtual network equivalent of virtual memory paging).

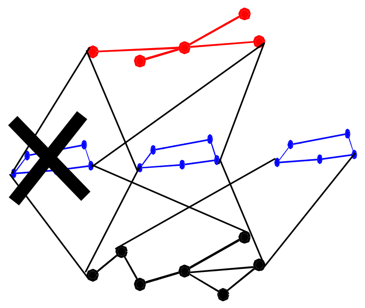


Figure 13. Multi-layered overlays allows dynamic re-mapping, supporting fault tolerance

Current members of the X-Bone project include Gregory G. Finn and graduate students Amy S. Hughes, Lars Eggert, Yu-Shun Wang, and Ankur Sheth. The author wishes to acknowledge Steve Hotz, Anindo Banerjea, Wei-Chun Chao, Oscar Ardaiz, and Stephen Suryaputra for their earlier contributions, as well as Ted Faber and the anonymous reviewers of ICNP.

7. References

[1] Aiken, R., et al., "Architecture of the Multi-Modal Organizational Research and Production Heterogeneous Network (MORPHnet)," ANL-97/1, Argonne National Lab, IL, Jan. 1997.

[2] Braden, B., "A Plan for a Scalable ABone - A Modest Proposal," (work in progress), July 1999.

[3] Braden, R., ed. "Requirements for Internet Hosts -- Application and Support," RFC-1123, Oct. 1989.

[4] CAIRN IPSEC patches, http://www.cairn.net

[5] Callon, R., Viswanathan, A., Rosen, E. "Multiprotocol Label Switching Architecture," (work in prog.), Aug. 1999.

[6] Campbell, A., et al., "Spawning Networks," IEEE Network, July/Aug. 1999, pp. 16-29.

[7] Chandra, P., et al., "Darwin: Resource Management for Value-Added Customizable Network Service," Sixth IEEE Int'l Conference on Network Protocols (ICNP'98), Austin, Oct. 1998.

[8] Delgrossi, L., Ferrari, D., "A Virtual Network Service for Integrated-Services Internetworks," 7th Int'l Workshop on Network & OS Support for Digital Audio & Video, May 1997.

[9] Divert sockets man pages, FreeBSD http://www.freebsd.org

[10] Eriksson, H., "MBone: The Multicast Backbone," Communications of the ACM, Aug. 1994, pp.54-60.

[11] Fox, B., Gleeson, B., "Virtual Private Networks Identifier," RFC-2685, Sept. 1999.

[12] Francis, P., "Yallcast: Extending the Internet Multicast Architecture," (work in prog.) Sept. 1999.

[13] Gleeson, B., et al., "A Framework for IP Based Virtual Private Networks," (work in prog.), Feb. 1999.

[14] Hattig, M. (ed), "Zeroconf Requirements," (work in prog.), Jan. 2000.

[15] Hamzeh, K., "Ascend Tunnel Management Protocol - ATMP," RFC-2107, Feb. 1997.

[16] Hickman, Kipp, "The SSL Protocol," Netscape Communications Corp., Feb. 1995.

[17] Ip-tun man pages http://www.isi.edu/xbone.

[18] KAME IPSEC patches, http://www.kame.net

[19] Kent, S., Atkinson, R., "Security Architecture for the Internet Protocol," RFC-2401, Nov. 1998.

[20] MacGregor, W., Tappan, D., "The Cronus Virtual Local Network," RFC-824, Aug. 1982.

[21] Moy, J., "Multicast Extensions to OSPF," RFC-1584, March 1994.

[22] Postel, J., "Internet Protocol," RFC-791, Sept. 1981.

[23] Rodeh, O., Birman, K., Hayden, M., Dolev, D., "Dynamic Virtual Private Networks," TR98-1695, Dept. of Computer Science, Cornell University, Aug. 1998.

[24] Savage, S., Anderson, T., et al., "Detour: a Case for Informed Internet Routing and Transport," IEEE Micro, V19, N1, Jan. 1999, pp. 50-59.

[25] Perkins, C., "IP Encapsulation within IP," RFC-2003, Oct. 1996.

[26] Perkins, C., "Miminal Encapsulation within IP," RFC-2004, Oct. 1996.

[27] Scott, C., Wolfe, P., Erwin, M., *Virtual Private Networks*, O'Reilly & Assoc., Sebastapol, CA, 1998.

[28] Tennenhouse, D., et al., "A Survey of Active Network Research," IEEE Comm. Mag., Jan. 1997, pp. 80-86.

[29] Terzis, A., Krawczyk, J., Wroclawski, J., Zhang, L., "RSVP Operation Over IP Tunnels," RFC-2746, Jan. 2000.

[30] Touch, J., Eggert, L., "Use of IPSEC Transport Mode for Virtual Networks," (work in progress), Mar. 2000.

[31] Touch, J., Hotz, S., "The X-Bone," Proc. Global Internet Mini-Conference / Globecom, Nov. 1998.

[32] Yemini, Y., da Silva, S., "Towards Programmable Networks," IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, L'Aquila, Italy, Oct. 1996.

[33] Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D., "RSVP: A New Resource ReSerVation Protocol," IEEE Network, Sept. 1993.