ELSEVIER

Computer Networks 36 (2001) 117–135

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Dynamic Internet overlay deployment and management using the X-Bone ☆

Joe Touch *

Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, CA 90292-6601, USA

Abstract

The X-Bone dynamically deploys and manages Internet overlays to reduce 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 network components (hosts, routers) to participate simultaneously in multiple overlays. Its two-layer IP in IP tunneled overlays support existing applications and unmodified routing, multicast, and DNS services in unmodified host 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. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: VPNs; Overlay networks; Virtual networks; Internet architecture; Network management; Tunnels; Encapsulation

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. The X-Bone system provides a high-level interface where users or applications request do what I mean (DWIM) 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 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 author 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. This is an extended and reorganized version of a paper of the same title that appears in the Proceedings of ICNP, Osaka, November 2000, pp. 59–68.

* Tel.: +1-310-448-9151.

E-mail address: touch@isi.edu (J. Touch).

PII: S1389-1286(01)00172-4

Other overlay 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 specific 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 used in conventional overlays. It is this two-layer scheme which supports stacked overlays, as well as permitting use of unmodified applications and network services inside a deployed overlay. It also permits network resources (hosts, routers) to participate multiple times in a single overlay, and is the only known overlay system that integrates both IPsec support and dynamic routing.

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 host and router 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, and includes performance analysis measurements. It also presents an extended and updated discussion of related work and our vision of the utility of the X-Bone to support networking research, networking education, dynamic service deployment, and fault tolerance.

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. Fig. 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.

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 a new protocol so it can be tested in a controlled environment. Containment was one of the first uses of overlays in the early 1980s [20], and motivated their re-emergence in the early 1990s 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

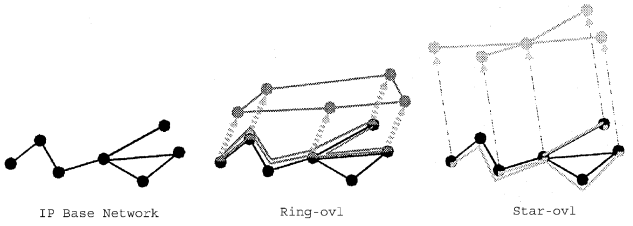

Fig. 1. A ring (center) and a star (right) overlay deployed on a base network (left).

infrastructure when it is not feasible to deploy new physical resources. They can also limit the scope and impact of network experiments, e.g., limiting them to nominal use of surplus capacity.

Abstraction is a new use of overlay networks. Both provisioning and containment imply the interim use of overlays that are supplanted by advanced hierarchical reservation in the former case, or more sophisticated dynamic service 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. Application 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 that 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 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 host operating systems and applications, assuming only basic support for IP in IP encapsulation.

The X-Bone extends the current Internet network architecture to include support for overlay networks. It supports stackable overlay networks, where control can be via a Web-based GUI (Fig. 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.

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.

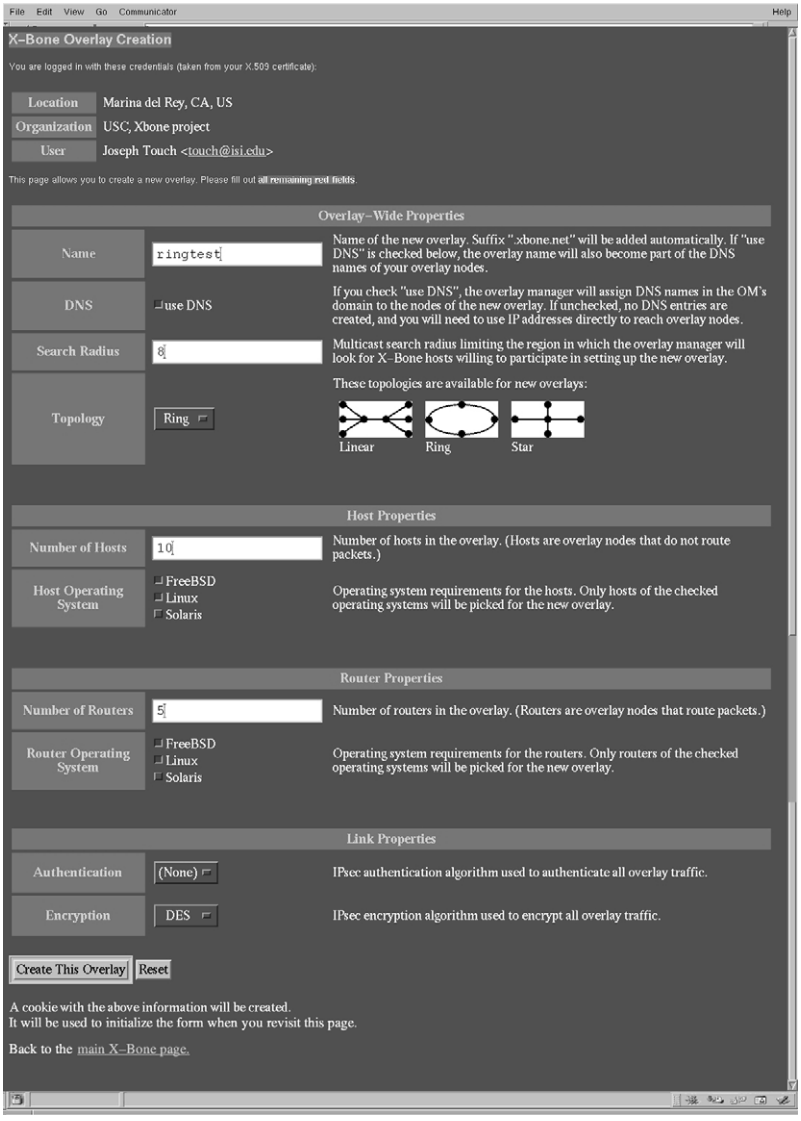

Fig. 2. X-Bone graphical user interface.

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.

Before proceeding, it is important to address capabilities outside the scope of the X-Bone. The primary purpose of the X-Bone is to examine the capability of automated overlay deployment. As such, a number of optimizations are not addressed, trading completeness for expedience. The X-Bone maps overlay components to available network components using a replaceable, arbitrary algorithm. Optimization of component placement, e.g., to avoid redundant tunnels, or to match the overlay topology to the base network topology is computationally intractable, and not the focus of this effort. Similarly, the X-Bone provides hooks to reserve network resources, including capacity, but QoS, network bandwidth, and router queues

are not currently reserved. The current Internet architecture for reserving such resources, RSVP, has only preliminary support for a single layer of tunnels, and does not support reservations on multilayer tunnels. OS modifications for such QoS support are part of other projects; the X-Bone focuses on using existing host and router operating systems.

2. Architecture

The X-Bone is a distributed system composed of daemons and control processes, much like SNMP. The components of the X-Bone system are discussed in Section 2.1. The system architecture is enabled by the use of two layers of tunnels, and by transport-mode IPsec over these tunnels. These techniques are described later in this section, in Sections 2.2 and 2.3, respectively.

2.1. X-Bone system components

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 Fig. 3.

OMs deploy overlays. A user creates an overlay by sending a request to an OM, either via a Web-based GUI (Fig. 2) or by sending a message directly to the OM API. Each overlay is coordinated by a single OM. Large overlays can be created by divide-and-conquer, where a single OM will fork suboverlay requests to other OMs. Fault tolerance can be achieved by replicating state in multiple backup OMs. Both of these latter capabilities (recursion, fault tolerance) are supported in the X-Bone architecture, though not implemented in current releases.

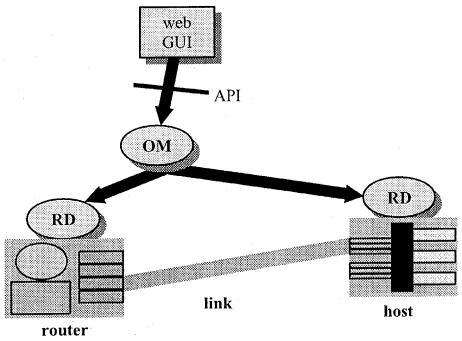

Fig. 3. X-Bone architectural components.

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 set of host operating systems, bandwidth requirements, etc. Invitations currently fit in a single IP 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 (Fig. 4) [21]. TTLs are the IP "time-to-live" fields; i.e., the TTL is effectively a hopcount that limits the number of routers a packet passes through.

This search over a well-known multicast address replaces rendezvous or registry systems. The 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, signed (authenticated), or encrypted to be private to a prearranged subset of components. Components each decide for themselves whether to respond, based on a match between their capabilities, availability of resources, and permissions.

RDs are daemons that configure and monitor the resources of routers and hosts. RDs listen for multicast UDP invitations, and respond when their capabilities, available resources and permissions match. The RDs respond with unicast UDP messages, indicating their willingness to participate in an overlay, and their capabilities (protocol version, OS type, etc.). The OM selects a (currently arbitrary) subset from among the responding RDs, [1] and opens TCP/SSL (X.509 encrypted) connections to each chosen RD. The OM deter-

[1] An arbitrary selection algorithm again replaces an intractable optimization that is outside X-Bone's scope.

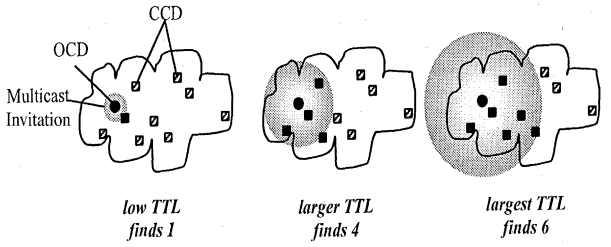

Fig. 4. Resource discovery via expanding-ring search.

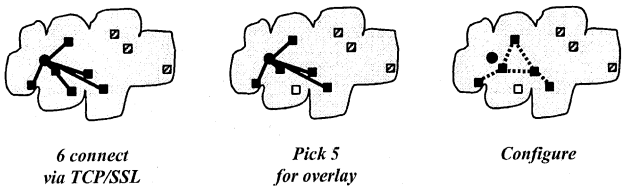

Fig. 5. Responding to invites, selecting, and configuring the overlay.

mines 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.

TCP/SSL secures the reliable configuration channel only; other mechanisms are needed to secure the multicast UDP invitations and unicast UDP responses. The X-Bone is currently applying S/MIME authentication to invitations, and S/MIME encryption to invitation responses to secure these UDP messages.

This architecture utilizes a single, well-known multicast channel for invitation announcements, and separate secure, 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 (see Fig. 5).

2.2. Two-level tunnels

The X-Bone uses two levels of IP encapsulation tunnels for each level of overlay, resulting in a total of three IP headers for an overlay on the base network (Fig. 6). Each overlay IP packet is

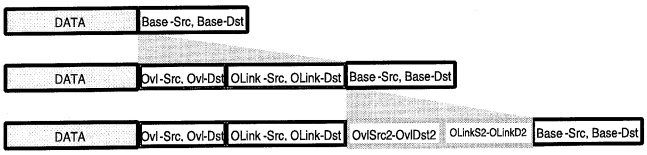

Fig. 6. Double tunneling results in three headers. Shown (top to bottom) are packets on the base network, an overlay, and an overlay *on* an overlay.

wrapped in two additional IP headers for each overlay layer; the first is the overlay link, and the next is for the endpoint of the next layer 'down'. The innermost 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), as also shown in the figure.

The additional layers of encapsulation are required to allow multiple overlay links between the two components, within the same overlay. This allows network components to participate multiple times in a single overlay (Fig. 7). Such multiply connected components are useful to emulate systems with larger number of components, i.e., 50-node rings simulated by using 10 router nodes.

In Fig. 7, an overlay is created that passes from host A → router B → router C → router B → router C → router B → host D, thus router B is visited three times. Consider a packet addressed from A to D, as shown. The packet enters B three different times – once via link U, once via link W, and once via link Y. All these addresses are overlay addresses. The difficulty occurs in distinguishing entry via link W and entry via link Y. In both cases, the base source and destination addresses are the same – the base network addresses of routers C and B. The overlay endpoint addresses do not change (from A to D). Without additional overlay link addresses, router B cannot determine whether to forward over link X (if entering from link W) or over link Z (if entering from link Y). Existing Internet routing lacks the context of received link; packets are routed solely on their IP addresses. The overlay link addresses encode these link contexts, and suffices in cases of multiple visitation.

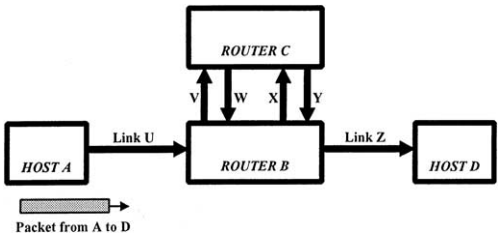

Fig. 7. A single component (router B) participates multiple times, using multiple overlay links between the same two components.

The additional layer also permits the use of multicast, dynamic routing algorithms, and IPsec 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. Again, the context of link address is critical, and the additional IP encapsulation layer provides this context.

It is also desirable for an overlay to secure its links. Link-level security in an overlay protects the routing and management of the overlay itself, as well as providing some protection when neither applications nor the base network provides security. In this case, the additional IP layer provides a place for overlay link IPsec, independent of base network IPsec (which may be incomplete) or application IPsec (which may not be available). Further, by providing a different header where overlay IPsec occurs, the X-Bone's IPsec allows either (or both) base or application IPsec independent of overlay IPsec. Fig. 8 shows these three places for IPsec – the dashed indicator shows where X-Bone deploys IPsec, when requested.

The two layers of the encapsulation change at every overlay hop, as shown in Fig. 9. Note the hosts, indicated by their single overlay endpoint and overlay link addresses, and the router, indicated by its pair of overlay endpoint and overlay link addresses. In this case, because the router is not an endpoint for overlay communication, its overlay endpoint addresses are shown in dashed ovals. 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 (having multiple overlay link and endpoint addresses), according to the standard Internet practice, though this can be supported even through a single (as shown) or multiple base interfaces.

The X-Bone is currently implemented using separate IP address spaces both for the overlay

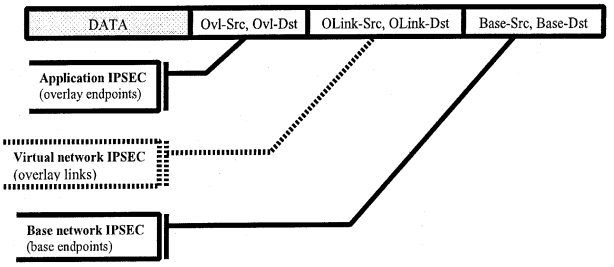

Fig. 8. Three headers = three places to do IPsec.

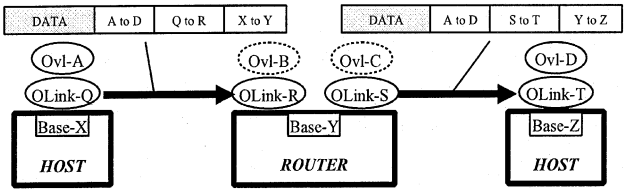

Fig. 9. A single packet traverses the overlay – modifying both outer IP headers at each hop.

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 routes (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. [2] Overlay addresses can be reused among overlays that do not overlap, as can be determined during the negotiation process.

The X-Bone adds two IP headers for each overlay layer. The use of IP at both layers supports the use of unmodified host operating systems, user applications, and network services. The X-Bone could equivalently add a new overlay identifier layer that could combine these two headers, e.g., as in the VPN-ID [11], requiring substantial programming without additional benefit.

[2] Again favoring design simplicity over efficiency.

2.3. IPsec issues

The X-Bone supports IPsec in the overlay [19]. The X-Bone uses transport-mode IPsec, rather than tunnel-mode IPsec, as would be expected in an overlay. The combination of transport mode IPsec and an IPIP link tunnel allows the X-Bone to support dynamic routing over IPsec'd links, which requires extraordinary coordination of key databases and routing tables if done in conventional (tunnel-mode) IPsec overlays. This section discusses the X-Bone's use of IPsec to secure the overlay links, and describes its use of transport-mode IPsec on IPIP tunnels in detail.

The use of two layers of IP encapsulation simplifies IPsec support. 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, as shown in Fig. 8. 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. This security comes at the expense

of hop-by-hop IPsec, whose performance is shown in Section 4.

IPsec in an X-Bone overlay is configured out-of-band, via the OM using TCP/SSL. Keys are generated by the OM, and sent to the RDs over these secured channels, rather than via IPsec key exchange protocols. The X-Bone uses explicit key distribution for simplicity, but can use in-band key exchange (e.g., IKE) when available; such in-band key exchange mechanisms are not currently used because they are not widely available, and because they 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 traditional 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 (Fig. 10). 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.

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 (Fig. 11).

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

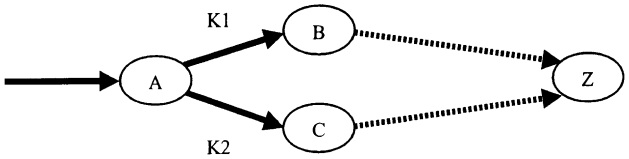

Fig. 10. Dynamic routing interferes with per-hop IPsec.

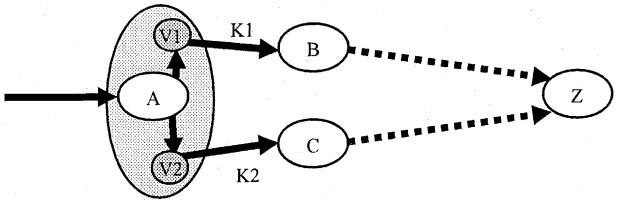

Fig. 11. Binding keys to virtual interfaces allows per-hop IPsec.

known overlay system that integrates both IPsec support and dynamic routing.

This example highlights the issue of lost context, as shown earlier in Fig. 7. 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.

3. Features of X-Bone overlays

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. It allows overlays to be selected on a per-application basis and it supports recursive overlays (overlays on overlays). It also enables dynamic routing inside an overlay and provides fault recovery. Finally, its architecture supports private and secure overlays, both in the invitations and in the configured overlays. These features are discussed in this section.

3.1. Overlay selection

The X-Bone allows applications to be used unmodified inside overlays. 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.xbone.net* as part of the overlay configuration.

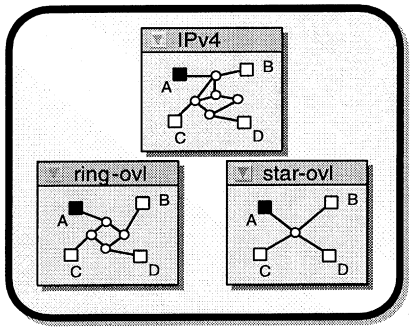

Fig. 12. User views of a network mapping utility; different views in different windows.

Both FreeBSD and Linux support the use of per-process overrides to the resolver default suffix; setting the process environment parameter LOCALDOMAIN allows the name *blue* to resolve to either *blue.abc.com* or *blue.apple.xbone.net*, depending on the whether LOCALDOMAIN is set to *abc.com* or *apple.xbone.net*. The DNS resolver uses LOCALDOMAIN as a per-process override to the default suffix in its system-wide configuration, so different processes on the same host can easily refer to different overlays, even using the same endpoint prefix names (e.g., *blue*). An example of how the overlays and base network from Fig. 1 would appear is shown in Fig. 12.

Base component names (here only hosts are shown named) remain the same; the DNS suffix of LOCALDOMAIN in each window differs. A standard network mapping utility can thus show different network views in different windows. The use of LOCALDOMAIN to set overlay-specific suffixes, and the DNS to subsequently determine IP addresses, allows unmodified applications to select either the base address or an overlay address for a given name prefix.

3.2. Ability to recurse

The X-Bone's version of tunneling supports layering an overlay on an overlay, also known as recursion or stacking. These recursive overlays are useful for managing groups of experiments on a

shared infrastructure. The shared components are gathered as a single overlay, and individual experiments performed on overlays on top of that overlay. This allows the resources of the shared components to be subdivided among experiments.

Recursive overlays are also useful for managing fault tolerance. A single overlay provides a layer of indirection, in which individual components can be replaced or renamed without affecting the superior or inferior overlay layers. A set of overlays at this middle layer can be exchanged to swap sets of resources.

The ability to support recursive overlays relies on a recursive tunneling mechanism; the X-Bone's two-layer tunnels have been tested in several recursive layers. It also relies on the X-Bone's preservation of true IP packet formats; because it relies on an IP substructure and presents IP as the overlay network, X-Bone overlays can be stacked without modification. Finally, recursion depends on isolation of the overlay management components; in the X-Bone, only the IP addresses inside an overlay are visible to processes on the overlay, so OMs and RDs at one overlay layer are not visible to other layers. This latter capability, of X-Bone systems at various overlay levels, is currently under development.

3.3. Dynamic routing

The X-Bone's dual-layer tunnels allow existing dynamic routing (RIP, via *gated* or *mrtd)*, multicast (via *mrouted* 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.

Addresses of an overlay are visible only at the network components (router, host) attached to the overlay, and are unique at those points. The X-Bone can configure gated or mrtd to exchange RIP messages only among sets of interfaces belonging to an overlay. For example, assume that a router has three interfaces, A, B, and C, and that it already has a gated configuration file indicating what routing protocols to use on those interfaces.

When the X-Bone deploys a new overlay, it allocates addresses for that overlay to be used at that router, e.g., X and Y. It updates the gated configuration file to indicate that addresses A,B, and C should ignore routing information relating to X and Y, and vice-versa. Then it indicates that interfaces X and Y run RIP.

The result is that A, B, and C exchange messages as if the overlay were not there, and the overlay exchanges messages as if the base network (A, B, C) were not there. Multiple overlays are handled by somewhat more complicated but corresponding rules. In each case, a set of interfaces is configured to ignore messages relating to all except members of that set.

The benefit of dynamic routing is that routing message exchanges are no longer limited to sets of routing peers in the base network. A telecommuter can utilize redundant network connections with dynamic route selection, e.g., between a cable modem path and DSL to his work network, without requiring complicated peering agreements between multiple ISPs and his work network.

3.4. Robustness

The X-Bone has a variety of fault detection and recovery mechanisms. Each X-Bone action (interface configuration, adding routes, configuring tunnels) has rollback recovery, and all state changes are logged to a state recovery file.

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

As a result, components used in the X-Bone are failsafe. When disconnected, their overlay configuration state is removed, and when reconnected it is restored. The current implementation has only single-component fault tolerance; the entire overlay is removed only when the OM fails or is disconnected. This level of fault tolerance can easily be extended to support hot-backup OMs (becoming 'hot' when failing to hear primary OM heartbeats), or to support piece-wise recovery.

3.5. Privacy and security

The X-Bone achieves secure invitation-based configuration, the use of secure configuration channels, and the ability to deploy IPsec'd overlays.

Invitations keep the configuration and availability of individual network components private. In a bulletin board or registry system, components would advertise their properties and reservation status. Configuration requests would still expose the intention to create an overlay, at least to the components advertised as available. The X-Bone uses multicast invitations, where the invitations are somewhat more public, but only the components available and capable of participating in an invitation respond.

The use of multicast for invitations 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, where the multicast invitation packet originates from the OM. Alternately, loose source routed [22] or an explicit proxy to a remote OM can center the invitation wherever useful, providing proxy resource discovery. 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.

Secure configuration channels are provided by TCP/SSL using X.509 keys, as used for secure Internet Web transactions [16]. The OM sends interface, route, IPsec, and tunnel configuration via these channels, which are opened only to RDs responding to the multicast invitations. The invitations and invitation responses use UDP, and cannot be secured with SSL; instead, S/MIME is used. Multicast invitations are typically only authenticated, whereas unicast responses are encrypted.

IPsec secures the data (IP packets) of the deployed overlay. This prevents packets from non-overlay components from interfering with an overlay, and ensures that overlay components can trust network-level packets, such as routing protocols and ICMP messages.

4. Performance

The performance of the X-Bone has been measured in a lab testbed using 300 MHz Pentium II and 733 MHz Pentimum III PCs running FreeBSD 3.2 with KAME IPsec extensions, FreeBSD 2.2.8 with CAIRN IPsec extensions, and Linux RedHat 6.0 with NIST IPsec extensions. These PCs were connected using a private, switched 100 Mbps Ethernet, and measurements were made between a single source host and destination host.

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. 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 likely results in this substantial bandwidth impact.

Fig. 13 shows the per-hop latency increases, measured using ICMP ping messages for the 300 MHz PCs. 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 the overlay link tunnel layer.

Fig. 14 compares end-to-end throughput of TCP and UDP streams, in similar three- and two-way comparison. Here both 300 MHz (foreground, striped) and 733 MHz (background, solid) PCs are measured. Note that the effects of multiple

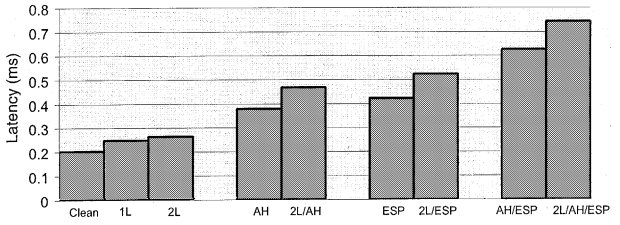

Fig. 13. Per-hop increases in latency using single and two-level tunnels.

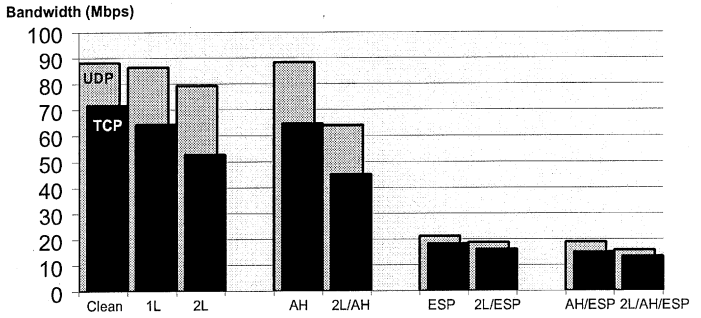

Fig. 14. End-to-end decrease in throughput using single and two-level tunnels.

tunnels are masked where encryption is used, because encryption processing dwarfs the overhead of additional encapsulation and decapsulation processing.

Fig. 14 shows how IPsec processing can result in substantial throughput penalties. Use of hop-by-hop IPsec in the overlay is the only way to ensure that packets from outside the overlay cannot encroach the overlay. It also protects network control inside the overlay, and provides a moderate level of protection to applications that do not use end-to-end IPsec.

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 the IP header and another 20 bytes are the transport layer header (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 data manipulation efficiency. This leaves a slack of 24 bytes of available packet space that can be used by additional encapsulation layers without changing the typical data payload. Such a change could affect applications not fully tested with limited packet payloads.

Even if the application is written to accommodate varying payload sizes, an MTU of 576, −40 for first IP and transport, leaves room for only a few levels of additional two-level tunnels (at 40 bytes each) before performance is severely compromised. These limits can be overcome using path MTU discovery (P-MTU), which both potentially increases the MTU, and stress-tests applications to deal with varying payload sizes. Successful P-MTU depends on contiguous deployment of MTU discovery support, which is not typically the case.

Even discounting the effect on MTUs, the increased headers consume network bandwidth. 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 [4,18]. As a result, our effective MTU is $576 - 40 - 40 = 496$ bytes. The additional two headers increase packet overhead by 7%.

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 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.

The following are further discussions of the particular distinguishing characteristics of the X-Bone and how they relate to ongoing and earlier work. The primary features of the X-Bone are its use of IP tunnels and its support for fully automated deployment and management of overlays. The X-Bone avoids OS modifications and provides a complete, end-to-end solution for overlay deployment. Finally, the X-bone is compared to VPNs, the dominant commercial variant of overlay networks.

5.1. IP tunnels

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 routes along a tunnel path, as noted in Section 4. IP in IP tunneling presents a conventional IP packet to intermediate routes hops, so takes the *fast path*, and does not stress these router's 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.

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 spaces 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.

5.2. Automation

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 MOR-PHnet [1] is an overlay system that supports virtual networks at all layers, from virtual physical, to link, to network, on up to application. MOR-PHnet was designed for use in supercomputer networks, where performance requirements necessitate low- and multilayer solutions. CRATO's Supranet [8] extends this multilayer notion with multilayer optimizations. Columbia's virtual active networks (VANs) are part of the Netscript project [32] and deploy link-layer virtual networks. These systems focus on multilayer 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.

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.

5.3. QoS support and OS requirements

Several overlay systems focus on QoS support for overlay deployment, e.g., Supranet and Darwin [7], the latter of which includes a component 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 avoids OS and application modifications, and supports the use of existing dynamic network services, such as routing, multicast, and nameservers, inside the overlay. Supranet, MOR-PHnet, 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.

5.4. End-to-end overlays

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. There is a trade-off between the X-Bone's end-to-end overlays and backbone overlays as provided by Darwin/VNS and Detour. Complete overlays allow endpoints to participate simultaneously in multiple overlays, and avoid the need for manual coordination to supplement the backbone deployment. Complete overlays also require software deployment at the endpoints, which places additional constraints on the types of participants in an overlay. The X-Bone acknowledges the level to which endpoints are a key component of an overlay, and thus includes them in overlay management. Backbone overlays include routers only implicitly; such implicit participation is much more difficult to monitor, control and coordinate.

The X-Bone's complete overlays are similar to VON's but differ from the partial deployment of Darwin/VNS [7], Detour [24], and VANs [32]. In Darwin/VNS, 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 mod-

ifications (i.e., DARWIN) 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.

5.5. VPNs

The X-Bone differs form 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 five routers emulating a 50-router ring, enabling large-scale experiments using limited resources.

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).

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].

6. Status and future work

The X-Bone is an ongoing effort at USC/ISI, currently in its fourth public release. The following section outlines its current status in detail, and discusses ongoing work to address optimizations, recursive (stacked) overlays, multihoming issues, and fault tolerance.

6.1. Current status

We released our first distribution of the X-Bone, including source code, in February 2000; it's most recent release is v1.3.1 (December 2000). The latest distribution supports FreeBSD 4.2 and Linux RedHat 6.2/NIST (kernel 2.2.5) (some earlier versions are also supported). The NIST patches (and KAME on FreeBSD prior to 4.x [18]) are optional and required only to support IPsec. Without KAME IPsec patches, FreeBSD pre-4.x also 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 [9,17]. Our current release uses non-encrypted multicast invitations and unicast responses, and configures components securely via TCP/SSL. It supports dynamic DNS with pre-overlay namespaces, and currently requires configuration via the GUI from among a fixed set of topologies, including ring, star, and line backbones. The current release also supports static intra-overlay routing, either via conventional static routes or via explicit entries in the *gated* or *mrtd* 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 or mrtd using RIPv2 and support for intra-overlay multicast have been developed and are being tested 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. 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 [4]). 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.

6.2. Optimizations

The X-Bone makes no attempt to optimize the mapping of overlay resources to base network resources. The X-Bone also does not currently use network resource reservation, e.g., RSVP. This is largely due to RSVPs limited (and currently experimental) support for tunneling, and lack of support for multilevel tunnels.

We are investigating other 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.

6.3. Recursive (stacked) overlays

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).

In divide-and-conquer, each overlay will have a master OM, which delegates a portion of the overlay to other slave OMs and then combines the results of their deployment. OMs will discover each other using the same multicast mechanism that OMs use to discover RDs.

6.4. Further multihoming extensions

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 host 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.

6.5. Fault tolerance

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. Fault replacement is intended to provide a dynamic alternative to manual relocation, but not to replace dynamic routing. Dynamic relocation is expected to operate at timescales much larger than that of dynamic routing, i.e., tens of minutes, hours, or even days. Conventional dynamic routing can provide fault tolerance within an overlay; when that overlay loses too many links to be considered viable (e.g., K-connected), it is replaced with a different overlay. Partial (component-wise) and incremental replacement can minimize the effect on overlay traffic.

Consider our first example, of a base network on which various overlays are deployed (Fig. 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 (Fig. 15). 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).

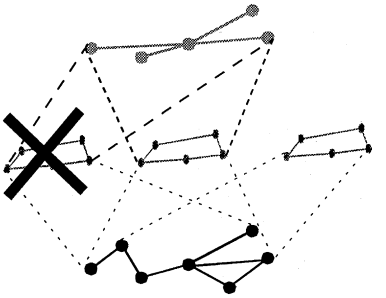

Fig. 15. Multilayered overlays allows dynamic re-mapping, supporting fault tolerance.

Acknowledgements

Current members of the X-Bone project include Gregory G. Finn and graduate students Amy S. Hughes, Lars Eggert, Yu-Shun Wang, Ankur Sheth, and Osama Dosary. 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, the anonymous reviewers of ICNP and Computer Networks for their feedback.

References

[1] R. Aiken et al., Architecture of the multi-modal organizational research and production heterogeneous network (MORPHnet), ANL-97/1, Argonne National Lab, IL, January 1997.
[2] B. Braden, A plan for a scalable ABone – a modest proposal (work in progress), July 1999.
[3] R. Braden (Ed.), Requirements for Internet hosts – application and support, RFC-1123, October 1989.
[4] CAIRN IPsec patches, http://www.cairn.net.
[5] R. Callon, A. Viswanathan, E. Rosen, Multiprotocol label switching architecture (work in progress), RFC-3031, January 2001.
[6] A. Campbell et al., Spawning networks, IEEE Network (July/August 1999) 16–29.
[7] P. Chandra et al., Darwin: resource management for value-added customizable network service, in: Sixth IEEE International Conference on Network Protocols (ICNP'98), Austin, October 1998, pp. 177–188.

[8] L. Delgrossi, D. Ferrari, A virtual network service for integrated-services internetworks, in: Seventh International Workshop on Network and OS Support for Digital Audio and Video, May 1997, pp. 177–188.
[9] Divert sockets man pages, FreeBSD http://www.freebsd.org.
[10] H. Eriksson, MBone: the multicast backbone, Communications of the ACM (August 1994) 54–60.
[11] B. Fox, B. Gleeson, Virtual private networks identifier, RFC-2685, September 1999.
[12] P. Francis, Yallcast: extending the internet multicast architecture (work in progress), September 1999.
[13] B. Gleeson, et al., A framework for IP-based virtual private networks (work in progress), RFC-2764, January 2000.
[14] M. Hattig (Ed.), Zeroconf requirements (work in progress), March 2001.
[15] K. Hamzeh, Ascend tunnel management protocol – ATMP, RFC-2107, February 1997.
[16] Hickman, Kipp, The SSL protocol, Netscape Communications Corporation, February 1995.
[17] Ip-tun man pages http://www.isi.edu/xbone.
[18] KAME IPsec patches, http://www.kame.net.
[19] S. Kent, R. Atkinson, Security architecture for the internet protocol, RFC-2401, November 1998.
[20] W. MacGregor, D. Tappan, The Cronus virtual local network, RFC-824, August 1982.
[21] J. Moy, Multicast extensions to OSPF, RFC-1584, March 1994.
[22] J. Postel, Internet protocol, RFC-791, September 1981.
[23] O. Rodeh, K. Birman, M. Hayden, D. Dolev, Dynamic virtual private networks, TR98-1695, Department of Computer Science, Cornell University, August 1998.
[24] S. Savage, T. Anderson et al., Detour: a case for informed internet routing and transport, IEEE Micro 19 (1) (1999) 50–59.
[25] C. Perkins, IP encapsulation within IP, RFC-2003, October 1996.
[26] C. Perkins, Miminal encapsulation within IP, RFC-2004, October 1996.
[27] C. Scott, P. Wolfe, M. Erwin, Virtual Private Networks, O'Reilly, Sebastapol, CA, 1998.
[28] D. Tennenhouse et al., A survey of active network research, IEEE Commun. Mag. (1997) 80–86.
[29] A. Terzis, J. Krawczyk, J. Wroclawski, L. Zhang, RSVP operation over IP tunnels, RFC-2746, January 2000.
[30] J. Touch, L. Eggert, Use of IPsec transport mode for virtual networks (work in progress), November 2000.
[31] J. Touch, S. Hotz, The X-Bone, in: Proceedings of the Global Internet Mini-Conference at Globecom, November 1998, pp. 59–68.
[32] Y. Yemini, S. da Silva, Towards programmable networks, in: IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, L'Aquila, Italy, October 1996.
[33] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, RSVP: A new resource reservation protocol, IEEE Network (September 1993) 8–18.

Joseph D. Touch received a B.S. (Hons.) degree in biophysics and computer science from the University of Scranton in 1985, an M.S. from Cornell University in 1988 and a Ph.D. from the University of Pennsylvania in 1992, both in computer science. He joined USC/Information Sciences Institute, Marina del Rey, California, in 1992, and is currently Director of the Postel Center for Experimental Networking in the Computer Networks Division. Joe has led projects ranging from gigabit LANs (ATOMIC2), NIC design (PC-ATOMIC), and multicast Web caching (LSAM), to his current project in the automated deployment and management of virtual networks (X-BONE) and optical Internet WANs and LANs. He is also a research assistant professor in the Department of Computer Science, University of Southern California, where he taught Advanced Opeating Systems for several years, and now runs a program for summer students (SGREP) and advises a number of graduate students. Joe also serves on the editorial boards of IEEE Network and Elsevier's Computer Networks, and is a member of several program committees, including IEEE Infocom (since 1994) and Sigcomm, and was co-chair of the IFIP/IEEE Protocols for High Speed Networks Workshop 1999.