Application Deployment in Virtual Networks Using the X-Bone

Yu-Shun Wang and Joe Touch
USC / Information Sciences Institute
yushunwa@isi.edu, touch@isi.edu
January 7, 2002

Abstract[1]

This paper describes a framework for application deployment within virtual networks using X-Bone. The framework supports deployment of arbitrary applications and allows users to configure the runtime environment by executing user-provided scripts instead of hard-coding application commands into the system. It also automates the process of constructing virtual networks and deploying applications by using X-Bone.

Keywords*: application deployment, virtual network, network management, overlay network, VPN, automated configuration*

1. Introduction

Virtual networks, or overlay networks, consist of a set of hosts and routers connected by encapsulated (tunneled) links. They were originally developed for protocol isolation, to test and incrementally deploy new protocols, e.g., Mbone, 6Bone [1][8]. They may also provide security (VPNs) or a simpler view of the network topology (e.g., X-Bone, RONs, Genesis) [2][7][13][18].

Virtual networks are increasingly used as infrastructure for distributed applications. Early applications focused on network services, such as multicast [8]. More recently, they have been used to deploy constellations of web caches, or to support so-called 'peer networks' [3][13][14].

There are several tools for the automated configuration, deployment, and management of virtual networks, such as the X-Bone, GUILN, and others [7][10][12][16][18]. Most of these tools have focused on the configuration of network devices and network-level services such as multicast and QoS. This paper focused on generic extensions to the X-Bone to support application deployment.

Application deployment is the process of installing, configuring and starting an application on a virtual network. The complexity varies with different types of applications. Sometimes privileged access to a system is required because some applications need to access core system components and configurations. Deploying applications over a network adds to the difficulty by requiring remote accesses to each system and sometimes across different administrative boundaries.

The X-Bone is a system for dynamically deploying and managing virtual networks over the Internet [17][18]. The X-Bone automates the process of deploying virtual networks including resource discovery, node selection and configurations, state refreshing (heartbeat) and monitoring, dismantling, and crash recovery. The X-Bone achieves these by using SNMP-like daemons on each system and providing a web GUI.

This paper describes a framework for deploying applications in virtual networks using the X-Bone. The procedure of deploying a virtual network is exactly the same as deploying other applications over a network, because virtual network is a kind of network application. With this framework, a user can request a virtual network with certain topology and also specify the application(s) to deploy over it. The X-Bone system then constructs the overlay and deploys requested applications on the virtual network without additional user intervention.

The next section provides background on application deployment. Section 3 describes the framework and Section 4 discusses the issues therein. Section 5 presents a case study of deploying an ABone. Section 6 lists related works and future directions.

[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.

2. Background

There are a number of steps required to deploy an application on a virtual network. Some of these steps are part of any application configuration, and can be automated easily. Some steps are more difficult to automate because there might be more than one copy of the same application running on the same node. Finally, there are steps that are complicated by running the application on a virtual network, which can challenge some assumptions of the application designer.

2.1. Application Deployment

To deploy network applications, the following tasks need to performed in order:

1. Find and select the systems (discovery),
2. Obtain access privileges (access),
3. Install applications,
4. Configure the systems and the applications,
5. Starting, monitoring, terminating (control),
6. Monitoring

These are discussed in further detail below.

Resource Discovery

In manual application deployment, the choice of systems is usually determined out-of-band. For dynamic and automatic application deployment, finding the 'right' system is difficult. It involves identifying systems that meet the requirements and are capable of running the application requested, and determining whether one has (or is granted) the necessary access. In our framework, the X-Bone resource discovery multicast is used to find the nodes to deploy. The X-Bone's discovery mechanism finds any nodes that are capable and willing, and selects a subset of the responding nodes arbitrarily. This does not address the need to precisely locate applications, which would be supported by augmenting the X-Bone's discovery mechanism.

Remote Access

To deploy applications over the network, users need remote access to the selected systems. This usually involves contacting the system administrators or owners to obtain access with necessary privileges beforehand. In our system, X-Bone daemons already run as root on each node to perform the deployment, based on an earlier arrangement. Basically, installing the X-Bone is as challenging as deploying an application, except that the X-Bone can then deploy subsequent topologies and applications without further reconfiguration.

Building and Installing Applications

During resource discovery, the system needs to verify that the requested applications are available on each node. If an application is not already installed, it can be built locally on each node, or pre-build executables can be deployed over the network. The later is preferred because building applications at deployment time could be time-consuming and may incur errors that need manual intervention. It also allows users to control the specific version or compilation options needed. Our system assumes that applications are already installed on the systems and uses a command provided by the users to verify the availability of each application.

Configuring Applications

Configuration refers to the parameters of the runtime environment for an application. Most of this is contained in configuration files or command-line arguments, but other data may be required (e.g., environment variables, accounts, devices, etc.).

Command-line arguments are easier to script for automatic deployment, and do not affect the local file systems. They are suitable only for applications with simple or few options.

Configuration files are typically used for applications with a large amount of complex runtime options. These files need to be generated and stored in unique locations. Each instance of the application must be configured to use the appropriate configuration file, usually via command line arguments.

Some applications require environment variables, special user login accounts, or new devices created. These can be more difficult to configure, because multiple instances of an application are likely to interfere with each other. It can be difficult or impossible to have different copies of a single application refer to different environment variables, usernames, or devices, unless they can be isolated local to a process.

Application control

Applications can require multiple steps to start or stop. These steps include managing configuration files, log files, or command line parameters. Stopping or signaling a running application can require locating the appropriate process and sending the appropriate shutdown, restart, or cleanup signal. Again, this can be complicated when more than one copy of the same application is running on the same node.

Monitoring

Monitoring is an important requirement for most applications, and a critical requirement for automated deployment and management of applications on virtual

networks. For applications with runtime processes, process IDs are the most common indicators (though an insufficient one) for monitoring purposes. For other applications performing system configurations, it is harder to verify the integrity of the settings. However, as with other areas of application control, it can be difficult to isolate a particular instance of an application when more than one copy of that application is running on a node.

Automatic management is simpler if applications have an integrated status report capability that generates a short, text summary.

2.2. Multi-Instance Application Conflicts

For hosts participating in multiple virtual networks, it is possible to deploy the same application more than once on a system. Although the procedures remain the same, care must be taken to prevent conflicts among instances of the same application regarding files (configuration files, log files, etc.), address binding (wildcard address), account names, process numbers, etc. This issue is discussed in more details in Sections 4.2 and 4.4.

2.3. Virtual Network Complexities

Some network applications make certain assumptions about the 'view' or'scope' of network components within a system. Most applications bind to the so-called wildcard address, which may include all valid IP addresses in a system, or iterate through the list of all network interfaces in a system. These suffice where there is only one instance of the application. On a host participating in multiple virtual networks, the 'view' or 'scope' of network components an application should see must be limited to the particular virtual network on which it is deployed. This is discussed further in Section 4.4.

3. Framework for Application Deployment Using the X-Bone

This section describes the particular framework developed for deploying applications using the X-Bone. It describes the basic operation of the X-Bone, and how it was augmented to support application configuration, deployment, and control. The system is based on a user-supplied script, and the use of the script to generate more specific scripts in multiple phases.

There are a number of assumptions on which this solution is based:

1. Applications are already installed on the hosts, or methods for downloading and installing the application is included in the user-provided script.
2. Virtual network management is entirely handled by the X-Bone. This includes finding and selecting the participating nodes, constructing virtual networks, monitoring, and dismantling virtual networks. The X-Bone currently supports 'arbitrary' selection – of a set of nodes responding to a request, an arbitrary subset is selected. More precise placement of services is supported in the X-Bone architecture, but not in the current implementation.
3. X-Bone works on the Internet, and constructs IP virtual networks. As a result, deployed applications must work on or above the IP protocol layer
4. X-Bone overlays are "closed". Components within the virtual network cannot communicate with those not in the same virtual network.

The internal workings and the virtual network architectures of X-Bone are discussed elsewhere [17][18]. As a simple overview, the X-Bone consists of a web user interface, a per-overlay Overlay Manager (OM) (like an SNMP controller), and a per-node Resource Daemon (RD) (like an SNMP agent). The Overlay Manager receives requests to deploy an overlay via an API, and sends per-node configuration commands to the Resource Daemons. In the current system, resource discovery uses a multicast invitation, and topologies are created from an arbitrary subset of respondents. The general architecture is pictured in Figure 1.

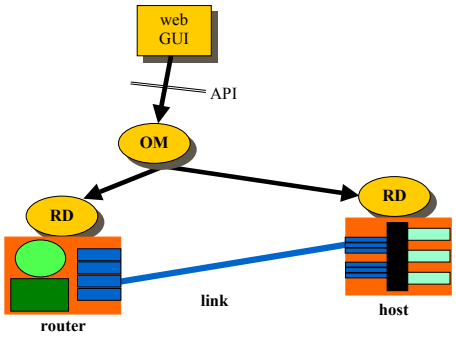

Figure 1. X-Bone architectural components

The application deployment framework uses scripts (usually Perl or shell) to setup and control the applications. This is a generic way to accommodate the different requirements of different applications without the need to hard-code particular commands into the X-Bone. The script needs to be different for different instances of the same application on different nodes, or for the same node when the application is on different virtual network. This will affect network parameters, e.g., IP addresses, routes, host or virtual network names, etc., as well as usernames, log files, or other parameters used in configuring or starting applications. This could require

the user to provide specific scripts for each node on each overlay created.

Instead, our solution uses a script generator generator, a script which is interpreted multiple times. At each interpretation, the script incorporates additional parameters. The final script is a control script for a particular application on a particular virtual network on a partitcular node (Figure 2).

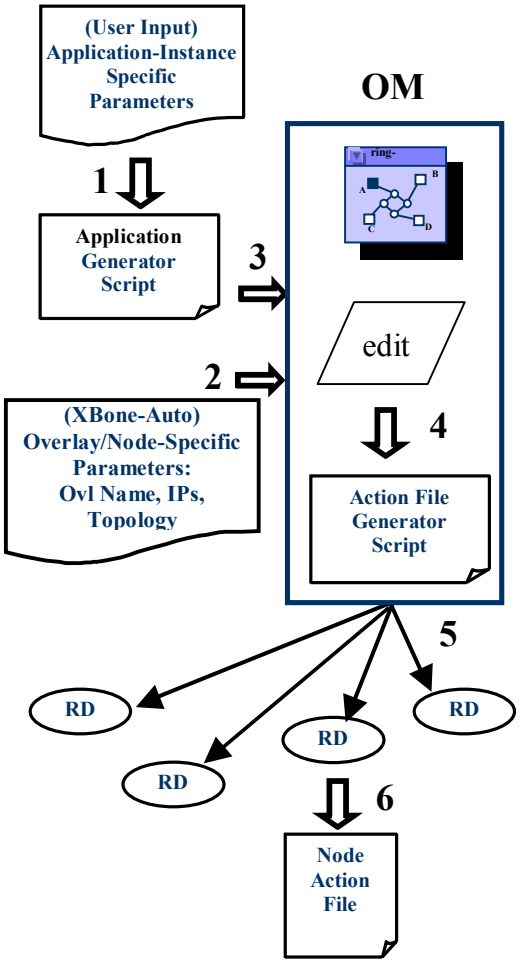

Figure 2. Block diagram of application deployment in virtual networks using X-Bone

The following is a step-by-step walk-through of Figure 2, above:

1. The user provides an Application Generator Script, which incorporates (via user input or modification) some application-instance specific parameters. This is done before creating the overlay and before any applications are deployed. This step may be partially automated itself, i.e., the user may write a script to generate this file, automatically determining software versions, etc. However, this automation happens outside the scope of the system. This Application Generator Script will be fed into the X-Bone system later when the user requests an overlay and applications to deploy.
2. The X-Bone GUI is used to request an overlay with certain constraints (topology, size, platforms, and security, etc.) and …
3. specify the application to deploy by including the URL of the Application Generator Script in the request

From this point on, the X-Bone system takes over the job of creating the overlay and deploying applications.

4. X-Bone Overlay Manager (OM) receives the request from GUI, and starts the resource discovery process. Once enough nodes are found, OM edits the script to integrate the overlay-specific parameters (topology, overlay network address & name, list of nodes, etc. common to all nodes in the overlay). The result is now called the Action File Generator Script.
5. The OM transmits commands to construct the overlay. The Action File Generator Script is also transmitted to each Resource Daemon with node-specific parameters.
6. Each RD modifies the script again to incorporate the node specific parameters (node virtual IP addresses, hostname, list of neighboring nodes, etc.). This final script is called the Node Action File. The Node Action File takes predefined control commands including: *configure*, *start*, *stop*, and *status*. At this point the RD automatically runs the *configure* and *start* commands.

Now the deployment is complete. We have the overlay constructed, and the application is deployed within the overlay as requested. Users will get a feedback on the GUI indicating the result of the overlay and application deployment.

7. X-Bone system has built-in GUI commands for monitoring the status of active overlays. When the monitoring command is issued, RD on each node will check the overlay configurations, and then execute the (pre-defined) Action File target to verify the status of the application.
8. When the users decide to destroy the overlay, they will use the X-Bone GUI to issue the delete-overlay command. RD's on each node will execute the Action File again to terminate the application and cleanup the environment, and then dismantle the overlay.

Script Generators

The Application Generator Script could be in any format. The only requirement is that the final Action File must be an executable text script (e.g., shell scripts, Perl scripts) with the follow format for execution:

```
Action_File [target]
```
where `target={conf|start|stop|status}`

X-Bone modifies the script by substituting the predefined keywords in the Action File with corresponding values.

If there are no differences among instances of the same application for different virtual networks, users can skip the Application Generator Script and provide the Action File Generator Script directly.

Application Installations and Verification

As stated above, we assume the application is already installed on each node. A simple verification assumes users provide a command in the script to verify the availability of the application. The command will be executed on each node during resource discovery process and the result is used as one of the criteria when selecting nodes to deploy. If the application is not available, the script needs to include procedures for installing the application.

Virtual Network Creation

The details on how X-Bone creates the overlays have been skipped, except the additional steps regarding application deployment. For X-Bone details, please refer to [18].

The Action File Generator Script is specified in the form of a URL in the X-Bone GUI request. The X-Bone OM uses HTTP, FTP, or local file access (as indicated by the URL) to obtain the script according to the address format. OM extracts the verification command to use in the resource discovery invitation. It sends the full script to the selected nodes for the overlay.

Application Deployment & Termination

The X-Bone RD on each node runs the Action File with a set of predefined targets (*conf*, *start*, *stop*, *status*) to control the application deployment or termination in the following order:

1. Configure the virtual network,
2. Setup the configuration files and environment with the "conf" target,
3. Start the application with the "start" target,
4. (OPTIONAL) Monitor the application with the "status" target,
5. Terminate the application with the "stop" target. It also handles the cleanup and restoration of the environment if necessary.

Multi-Application Scripts

Users can include multiple applications in one script, but this is not recommended unless these applications are closely related or there are certain constraints or dependency between them. Otherwise the users could include multiple scripts in the create-overlay request to deploy multiple applications. In this case, X-Bone will execute the script in an arbitrary order.

4. Discussion

This framework automates the complex procedure of deploying applications on top of a virtual network. Certain tradeoffs were made to accommodate this task more simply in the prototype. There were also issues regarding security, network reentrancy, and multihoming that have been encountered when designing and implementing this framework.

4.1. Security

Security is probably the most important issue. There are several aspects of security related to this framework: communications between X-Bone components (OM and RD's), integrity of the applications to deploy, access privileges for users who request overlays and deploy applications in them.

X-Bone Communications

All control communications between X-Bone components are SSL-encrypted [11]. Each host and user must present its X.509 certificate signed by designated Certification Authority. The system administrators can grant access to each user according to their IDs on the certificate. Again, please refer to [18] for details regarding X-Bone security.

Integrity of Applications and Action File Scripts

Application integrity has not been addressed in this framework. Instead, we assume the applications are already installed and verified on each host by trusted parties. Users can include the commands to download or obtain the program executables from secure locations in the Action File Script, and install the application files as part of the configuration process.

The integrity of the Action File Scripts is generally protected by the security measures of X-Bone system. It is very difficult to verify the scripts against their original content because X-Bone replaces certain keywords in the

script with overlay- or node-specific values. One solution is to supply those parameters in the form of command-line arguments for the Action File Scripts, this renders the set of arguments very long and complex, and was not selected for this prototype.

Access Privilege Control

The X-Bone RD runs as system administrator (root) on each node, and it executes the user scripts also as root.

This grants the users, though indirectly, the privilege level of system administrator when executing the Action File Scripts. This is a significant security concern particularly because we cannot verify every command listed in the script due to the diversity of applications.

This problem is also compounded by the fact that almost all platforms adopt the 'all-or-none' approach regarding critical system resources. Applications accessing critical system resources must be run at the highest privilege. Examples include the tunnel devices used by virtual networks, privileged ports on virtual addresses, etc. Fine-grain access control over resources would be preferred, so that resources can be partitioned according to virtual networks and grant different users access right to those resources without giving them root access to the whole system. This is addressed in another, ongoing project (NetFS). Such partitioning would also limit the 'scope' or 'view' of each application to its corresponding virtual network only, instead of all the network components available on a system.

4.2. Application Configuration

The framework allows users to perform configuration for each application including specifying command-line arguments, generating configuration files, setting up run-time environment for each application. Although this is achieved via user-provided scripts, options are still limited since the process is non-interactive, and conflicts among different instances of the same applications must be avoided. Users must also separate the working directories for different instances of applications.

4.3. Resource Discovery and Selections

Currently, X-Bone uses authenticated multicast to send out invitations for overlays. This limits the scope of resource discovery because IP multicast is still not widely available. Other forms of resource discovery such as broadcast and bulletin board are possible, but are outside the scope of this work.

The X-Bone uses the following criteria to determine the list of nodes to participate in an overlay: platform capability, host/router, and application availability. The selection process in the X-Bone is not optimized based on locality, hop count, or other quality-of-service parameters. At this time, all nodes deploy all request applications.

4.4. Network Reentrancy

Network Reentrancy refers to multiple instances of an application in a node, which could mutually interfere. Conflict avoidance relies not only on the Action File Script for deployment, but also on how a program is written. If a program hard-codes assumptions (wildcard addresses, login name for runtime account, etc.) without means to override, then reentrancy is not possible regardless of how the scripts are written.

The following are the major areas of conflicts:

- **Account login names**: Some applications must run as certain dedicated user logins in a system. Those applications must not hard-code login names in the programs, but rather should use symbolic references that allow user override. The deployment script might need to create separate user accounts for different instances.
- **File Systems:** Configuration files, log files, and output files must be separated to avoid conflicts. Applications must have command-line arguments to select different files to use, or such conflicts are impossible to avoid.
- **Network Resources:** As described earlier, wildcard addresses will prevent multiple instances of the same applications from running. Routes, DNS entries, interface list are all candidates for such conflicts.

Most applications provide ways to specify those listed above either in command-line options, or in configuration files. Users must study the instruction manuals of each application carefully before writing the scripts and take extra care to prevent and resolve conflicts to achieve network reentrancy.

4.5. Multihoming Implications

Any hosts participating in virtual networks must be multihomed [6]. This may mean that whether they are hosts or routers on the Internet, or whether they act as hosts or routers in virtual networks, they may also need to satisfy the requirements of Internet routers [4][18]. Without this requirement, each host could only participate in one network (virtual or not) since it won't be able to forward packets to and from different virtual and physical networks.

5. Case Study: Deploying ABone Using X-Bone

We have implemented the framework in the X-Bone system and have given several presentations and demos of deploying ABone within overlays using X-Bone [5]. ABone is a DARPA-funded project at ISI and SRI to implement a virtual testbed for active network research programs. ABone consists of nodes running the Active Network Daemon (Anetd) to support multiple Execution Environments (EEs). Active Applications (AAs) runs within different EEs in the ABone architecture.

In our demonstration, the proposed framework deploys the Anetd in virtual networks. The following highlights the procedures of the deployment.

5.1. Configuring Anetd

Anetd on each node requires running 7 instances on 7 different accounts, each with its own set of configuration, log, and output files. The configuration process includes creating the required accounts and all the configuration files, and setting up the commands such that they would use the corresponding accounts and files. To simplify the configuration, the same configuration files were used for all 7 instances. The overlay names were embedded in all account names to differentiate among different virtual networks.

5.2. Starting and Terminating Anetd

When starting Anetd, it's necessary to specify the accounts each instance is associated with, and the IP addresses on the virtual network it should bind to. The script also needs to identify each instance and terminate each process when stopping Anetd.

5.3. Conflicts Resolution

As mentioned in the Section 4, all possible conflicts must be avoided and resolved when multiple ABones are deployed on the same node. Anetd is a good example to demonstrate this capability because it requires dedicated login accounts to run it, it uses several configuration, logging, and output files, and it binds to IP addresses. Separate sets of accounts were created for different ABone instances, so the file system and login account conflicts are resolved. Command-line arguments were used to assign correct IP addresses for each Anetd instances and the account they associated with.

5.4. Caveats

There are some caveats due to the closed nature of X-Bone overlays. Anetd loads codes for EEs and AAs from Trusted Code Servers (TCS) in the form of web servers. To truly implement a complete ABone in an overlay network, a mirror code server was cloned within the virtual network to serve as the TCS for the ABones deployed in the virtual networks.

6. Related Works and Future Directions

This section discusses the related commercial applications of this framework, as well as the future directions of this research.

6.1. Commercial Applications – From VPNs, Peer-to-Peer, to Grid Computing

There are currently several commercial developments on Virtual Private Networks (VPNs) by service providers and network equipment venders. The majority of the works focus on security aspects of virtual networks and management of VPNs. Very few has touched upon the problems of application deployment.

At the other end of the spectrum, peer-to-peer networks and applications (P2P), and content distribution networks (CDNs) among content providers approach virtual networks from a totally opposite point of view: they deploy applications first, and then try to construct the virtual network at the application layer. The problem with doing virtual networks at the application layers is that very often, they will have to re-invent a lot of functionalities which are well-studied and implemented in the Internet like routing, naming, addressing, congestion control, just to name a few. Although different applications might have specific requirements on particular functionality, in general, this represents a duplication of effort and usually the results were not as scalable and well implemented as their Internet counterparts.

Although this framework might not solve all aspects of both ends of the spectrum, it provides a platform in which to combine virtual networks and application deployment.

The third related commercial and academic development is grid computing [9]. In general, the Grid focuses on network application development and resource sharing across the network, in which virtual networking is just one of possibility. Virtual networking provides partitioning, security, and topology abstraction which makes application development and deployment simpler. Our application deployment framework is compatible with the Grid Computing architecture, in that case.

6.2. Extended Definitions of Applications

In addition to just the traditional types of applications, the definition of "application" can be extended to include network management functionalities, distributed services, content distributions, and even protocol deployment and testing.

6.3. Non-Overlay Application Deployment

The current framework of application deployment is implemented as part of the virtual network deployment process. We plan to develop a version of general network application deployment that could either factor out the virtual network part of the framework, or even treat virtual network as just another application to deploy. This new version could potentially be a generic platform-independent application deployment solution.

7. Conclusions

This paper describes a framework for application deployment within virtual networks using X-Bone. The framework supports arbitrary applications and allows users to configure the runtime environment by executing user-provided scripts instead of hard-coding application commands into the system. It also automates the process of constructing virtual networks and deploying applications by using X-Bone.

8. Acknowledgements

This worked was first developed by Oscar Ardaiz [3]. Oscar implemented a prototype to deploy versions of the Squid web proxy cache.

The deployment of ABone on virtual networks was in part supported by the ABone project at ISI, and with the valuable help of Bob Braden, Steve Berson, and Craig M. Rogers at ISI. Steve Dawson at SRI not only made valuable suggestions and corrections on the procedures of deploying Anetd, he also incorporated several changes into Anetd to make the deployment possible.

Current members of the X-Bone project include Gregory G. Finn and graduate students Amy S. Hughes, Lars Eggert, and SunHee Yoon. The authors also acknowledge other prior project members Steve Hotz, Anindo Banerjea, Wei-Chun Chao, Ankur Sheth, Osama Dosary, and Stephen Suryaputra for their earlier contributions.

9. References

[1] 6-Bone web pages - http://www.6bone.net/

[2] Anderson, D., Balakrishnan, H., Kaashoek, M. F., Morris, R., "Resilient Overlay Networks," Proc. 18th ACM SOSP, Banff, Canada, October 2001.

[3] Ardaiz Villanueva, O., Touch, J, "Web Service Deployment and Management Using the X-Bone," Spanish Symposium on Distributed Computing, SEID2000, Sept. 25-27, 2000, Ourense, Spain.

[4] Baker, F., "Requirements for IP Version 4 Routers," RFC 1812, June 1995.

[5] Berson, S., Braden, D., Riciulli, L., "Introduction to the Abone," (internal report), June 15, 2000. http://www.isi.edu/abone/DOCUMENTS/ABoneIntro.pdf

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

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

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

[9] Foster, I., Kesselman, C., (eds.), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, 1999.

[10] GUILN web pages - http://www.cairn.net/guiln.html

[11] Hickman, Kipp, "The SSL Protocol", Netscape Communications Corp., Feb 9, 1995.

[12] Lim, L., Gao, J., Ng, T., Chandra, P., Steenkiste, P., Zhang, H., "Customizable Virtual Private Network Service with QoS," Computer Networks, July 2001, pp. 137-152.

[13] Oram, A. (ed.), *Peer-to-Peer: Harnessing the Benefots of a Distruptive Technology*, O'Reilly, Sebastapol CA, 2001.

[14] Radioactive project web pages – http://www.cs.ucl.ac.uk/research/radioactive/

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

[16] Su, G., Yemeni, Y., "Virtual Active Networks:Towards Multi-Edged Network Computing," Computer Networks, July 2001, pp. 153-168.

[17] Touch, J., Hotz, S., "The X-Bone," Third Global Internet Mini-Conference at Globecom '98 Sydney, Australia Nov. 8-12, 1998 pp. 59-68 (pp. 44-52 of the mini-conference). http://www.isi.edu/touch/pubs/gi98/

[18] Touch, J., "Dynamic Internet Overlay Deployment and Management Using the X-Bone," Computer Networks, July 2001, pp. 117-135. http://www.isi.edu/touch/pubs/comnet2001/ A previous version appeared in Proc. ICNP 2000, Osaka Japan, pp. 59-68.