Environments for Active Networks

Rosen Sharma, S. Keshav, Michael Wu and Linda Wu
Cornell University

Abstract

Traditional networks expose a fixed set of network services, which are hard to modify and to customize. In contrast, active networks seek to create an extensible system by providing an environment where nearly arbitrary applets can be executed, allowing new services to be dynamically created. Applets executing inside the network or on servers allow the network to respond in real time to asynchronous events. Such a framework allows us to rapidly deploy protocols and refine them, dramatically reducing the development cycle. In contrast to declarative frameworks (like RSVP) which needs to encompass the needs a broad range of applications and yet be compact to be useable, a procedural framework (like applets) allows the creation of customized application level libraries, some of which maybe later made part of the declarative framework. An applet-based framework also facilitates the mechanisms for developing the glue between different networks and protocols.

However, applets by themselves are not sufficient to achieve the goal of extensibility, We have shifted the problem from defining a set of services to defining environments in which agents can execute. Creating interesting and useful agents requires us to design a rich execution environments for network services.

Our work focuses on defining these environments, like object models for packet scheduling, packet routing and resource reservation. In this paper we discuss our approach to this problem. We also describe how we defined and exposed the object model for a data repository and present some preliminary results.

Introduction

We define active networks as networks that allow programs or applets to be executed at intermediate nodes. Instances of active network components are already used for mobile communication (for forwarding packets after a handoff), in video conferencing (for prioritized dropping of multicast groups) and firewalls (for allowing customizable filters) among others.

For example, Figure 1~~Figure 1~~ describes the architecture of a firewall which allows scripts to be uploaded from a server. On completion, a script can upload other scripts. Scripts used in this way are very similar to applets. Such a scripting system provides a mechanism to allow new application data streams to be filtered through the firewall. For example, to permit an application to receive data on two UDP ports through the firewall, only a small script needs to be written.

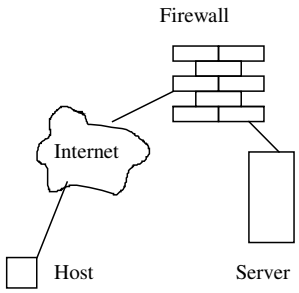

Figure 1~~1~~ Firewall

A sample script is shown in Figure 2~~Figure 2~~. Implicit in the code is the fact that the agent has access to the packets matching some flow specification. The execution environment allows scripts (applets) to access flows (connections) through the firewall in a secure and efficient manner.

```
if (pkt.protocol == udp)
  if  (pkt.src == s1 ) {
    // 1. this is a setup packet
    // 2. we know the application
    //    level protocol
    //  3. read ports that the application
    //     will use from the data
    ......
    //  4. upload script from the server to
    //    allow packets on  these ports
    //    to be forwarded
  }
}
```

Figure 2~~2~~ Sample Script

A framework for agent execution consists of many small building blocks, such as secure languages to write agent code, naming schemes for instances of agents, mechanisms for rendezvous, languages for communication between agents, failure semantics, accounting and storage. Some solutions to these issues exist and others are being investigated as part of the CUFAN (Cornell University Framework for Active Networks) initiative. In particular we plan to use the safe language kernel (SLK); a multiuser Java Virtual Machine that allows applications to share classes as data structures. For example a routing table may be shared between an application and the router, where SLK guarantees that the applet cannot gain access to the private parts of the routing table.

Our work focuses on defining object models, pertaining to networks, which agents can access. In this position paper, we describe the scope of our work in this area using selected examples in areas of protocol stacks, switching elements (routers and PBX's) , signalling and SNMP.

Examples of Active Network Components

Protocol Stacks (Firewalls/Mobility)

As mentioned earlier a firewall allows applets to access the packets belonging to a flow. Firewalls can be implemented at several layers of the protocol stack. For example a firewall can be implemented in user space at the cost of copying data into and out of the kernel. Or it could be implemented in the transport layer where the inside and outside connections share buffers thus avoiding the extra copies. Or as a packet filtering gateway at the network layer. We can migrate firewall functionality from one layer to another by exposing all the layers of the protocol stack to the firewall applet.

A parallel problem arises in the use of applets in mobile TCP schemes. The mobile sends an applet to the base station, to improve the performance of TCP connections. The applet can run at the network layer to give the abstraction of a reliable link or use intelligent caching mechanisms at the transport layer or for forwarding packets after a handoff or in user space by splitting the tcp connection into two connections.

Exposing the object model of a protocol stack allows us to unify the above individual solutions into one general framework, which allows mobile schemes or firewalls to be implemented at any layer of the protocol stack. The framework also allows applets to enhance functionality, for example provide QoS, by using other exposed models like those of scheduling and buffer management.

Routers

We believe that router operating systems are a neglected area of research. Most research testbeds use workstations as compared to real routers. This discrepancy reflects itself in the design of protocols. For example, the shared explicit reservation style in RSVP is simple to implement in a workstation kernel, but almost impossible to implement efficiently in hardware.

In traditional operating systems the hardware abstraction layer, makes the task of structuring and porting easier. Similarly by exposing the object model for the hardware of a router, we provide a substrate over which protocols could be built. The decreased performance is compensated by the richer development environment, which allows us to study several design issues.
PC's can expose models for the hardware

PBX's

Providing qualities of service that are customary in the phone networks on the I~~i~~nternet seems to be a daunting if at all feasible task. The Internet has the best effort delivery model and the telephone network provides guaranteed service. Voice traffic is constant bit rate and is well understood; data traffic on the Internet is not well understood, ever changing in character and bursty on all time scales. We believe that different networks will coexist at ~~l~~least in the near future. There are many interesting applications that need to use services provided by both networks. Consider for example an audio conferencing application which has audio and a whiteboard. For the audio we may want to use the phone network, while we want to use the Internet for the shared whiteboard. This is an example of what we call multi-mode communication. It tries to leverage the strengths of each network and provide interoperability in the control plane. Exposing similar object models for switching elements, both IP routers and telephone switches, provides a framework to develop multi-~~mixed~~ mode applications.

State Establishment and Management

Signa~~l~~ling (RSVP)

Signaling protocols like UNI or RSVP are declarative in nature in that they only specify the quality of service a connection requires and not how to obtain it. Efforts to make these protocols expressive, in order to deal with a broad spectrum of quality requirements, leads to an exponential increase in the number of parameters. Moreover, the class of applications that may use these protocols is unknown. We avoid these inherent problems if signaling is carried out by applets that access a a well understood interface at each router and carry out reservations for the endpoints. Eventually, on gaining sufficient understanding of how to model the environment we intend to come up with a declarative signaling protocol that is likely to be a more compact and usable version of the existing protocols. Our approach here , therefore, is to built on top of the object models developed for routers (described above) and define models for scheduling and routing.~~As the field of networking matures, many problems that arose in other fields are also begining to arise here. For example, SQL, the query language for relational databases was declarative and thus abstracted out data representations and procedural description of query execution allowing for optimizers to find the best way of executing a query. But the downfall of a declarative language is that it needs to be sufficiently expressive to communicate information of widely varying sorts and at the same time be reasonably compact~~. ~~Similarly that signaling protocols like UNI or RSVP are declarative in nature - they just specify what is required and not how to obtain that reservation~~. ~~Efforts to make them expressive, to deal with a broad spectrum of requirements, leads to an exponential increase in the number of options. Also the class of applications that would use these services are unknown. The above points argue favorably for a procedural language where applets are presented with a well understood interface at each router and carry out reservations for the endpoints. Upon gaining sufficient understanding of how to~~

~~model the environment, what is important and what is not we can come up with a declarative language and thus a more compact and usable version of the existing protocols. Our approach here is to built on top of the object models developed for routers and define models for scheduling and routing.~~

SNMP

The designers of SNMP purposely made it ~~is~~ simple because ~~the~~ they did not expect endpoints ~~are not expected~~ to have adequate computing power. T~~t~~oday, however, managed entities ~~endpoints~~ run operating systems (like pSoS) which supports java applets and are powerful enough to do things beyond the scope of SNMP. We plan to enhance SNMP by allowing applets to be sent to the managed entity~~endpoints~~, enabling real time control policies, fine-grained measurement, and ~~and~~ sophisticated trap generation algorithms ~~paradigms~~. This also allows ~~and~~ "semantic routing", where applets move from one endpoint to another based on some information present on the host. For example…

A Prototype Active Network Component

In order to understand some of the issues and potential benefits of an active network component, we exposed the object model of a web server. Even a simple object model, enabled clients to run complex queries involving conjunctions, disjunctions, projections and user defined functions as applets.

The web server exported a set of tables to java applets. Figure 3~~Figure 3~~ shows a sample interaction between a client and server. The client (Netscape) connects to the server which sends a description of it tables and query capabilities in javascript. For a query language it specifies how relational operators can be mapped into its language, allowing the user to query the web server or different data repositories in a consistent manner. The user can also define some small closed functions (applets) to be used in selection predicates. The queries are processed by the perl engine, while the applets are run in the java virtual machine.

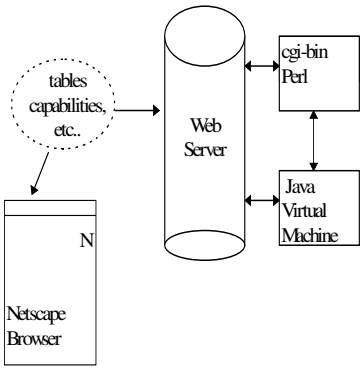

Figure ~~3~~3 Active Network Component

Accessing tupules from a table is similar to accessing rows of a table exported by the web server. Since we cannot access all rows at once we needed to introduce iterators. Expressing the capabilities of an active component (for example, the query capabilities in this case) is the generalization of the problem to provide similar models for different switching elements. In order to establish state at the server, the client sends an applet and did not need to define a signalling protocol. Our experience with this design was extremely positive. Defining the environment was hard and required several iterations. Once defined it allowed rapid development of different clients.

Related Work

A similar vision for what future networks might look like is shared by other reasearchers [7,8,9]. In their network, applications send capsules rather than packets. A capsule is a packet where the header identifies what piece of code should be run for the packet. Their work, unlike ours, put applets in the forwarding path.

[1, 3, 4, 5] study the advantages and disadvantages of using agents. They conclude that agents offer significant advantages in cases of disconnected operation, individualized service, server side execution, high bandwidth - low latency interactions, but they also introduces problems like performance limitations, secure environments, transmission efficiency.

Other work focuses on different aspects of the agent framework. [5, 6] describe the the Arpa Knowledge sharing effort which led to the definition of an agent communication language (ACL), which consists of a vocabulary, Knowledge Interchange Format (KIF) and the Knowledge Manipulation and Query Language (KQML). ACL is a declarative language (similar to SQL), which represents knowledge as tupules and first order logic. [2] classify frameworks based on their capability to support migrate agents and describe how this is done in the visual obliq system. This is similar to the JavaBeans [16] effort for java. [11, 12, 13, 14, 15] describe various aspects of Java and the Java virtual machine, which is the language in which agents are written.

[9, 10] describe operating systems that allow user code to be executed in a safe manner in the kernel. This allows user applications to directly access hardware like MPEG boards or to customize the drivers. The work on Berkeley Packet filters, which allows arbitrary filters to be installed in the kernel is also along the same lines.

Conclusion

We believe that active networks allow us to explore different approaches to solve several existing problems. Our work has applicability beyond Active Networks as it helps us to gain a better understanding of some fundamental issues in networking. Our focus is to define network environments where applets can do useful things. Our experience in exposing a simple object model for a data repository has been very encouraging.

References

[1] Mobile Agents: Are they a good idea ?, Collin Harrison, David M. Chess, Aaron Kershenbaum, IBM Research Divsion, T. J. Watson Research Center, Yorktown Heights, NY 10598, http://www.research.ibm.com/massive/mobag.ps

[2] Migratory Applications , Krishna A. Bharat, Luca Cardelli, http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-138.html

[3] Is it an Agent or just a Program ?: A taxonomy for Autonomous Agents, Stan Franklin and Art Graesser, Institute of Intelligent Systems, University of Memphis http://www.msci.memphis.edu/~franklin/AgentProg.html

[4] What's an agent, anyway ?, Lenny Foner (foner@media.mit.edu), http://fonner.www.media.mit.edu/people/foner/Julia/Julia.html

[5] Intelligent Agents: theory and practice, Wooldridge and Jennings, http://www.doc.mmu.ac.uk/STAFF/mike/ker95.ps

[6] Software Agents, Genesereth and Ketchpel, http://logic.stanford.edu/sharing/papers/agents.ps

[7] Request for Comments: From Internet to Activenet , D. L. Tennenhouse, S. J. Garland, L. Shrira and M. F. Kaashoek, Laboratory for Computer Science, MIT

[8] Towards an Active Network Architecture. David L. Tennenhouse and David J. Wetherall Telemedia, Networks and Systems Group, MIT,

[9] A Case for NOW (Network of Workstations), Thomas E. Anderson, David E. Culler, David A. Patterson, and the NOW team

[10] SPIN http://www.cs.washington.edu/research/projects/spin/www/

[11] Java Language Overview ftp://ftp.javasoft.com/docs/java-overview.ps

[12] The Java Language Environment White Paper, ftp://ftp.javasoft.com/docs/whitepaper.ps.tar.Z

[13] The Java Platform White Paper ftp://ftp.javasoft.com/docs/JavaPlatform.ps

[14] JavaOS(tm): A Standalone Java Environment ftp://ftp.javasoft.com/docs/whitePaper.JavaOS/JavaOS.cover.ps

[16] Java(tm) Beans: A Component Architecture for Java http://splash.javasoft.com/beans/WhitePaper.html

[17] Java Security Faq http://java.sun.com/sfaq

[19] Firewalls and Internet Security, William R. Cheswick and Steven M. Bellovin, Addison Welseley