SNAP Based Resource Control For Active Networks

Walter Eaves (1), Lawrence Cheng (1), Alex Galis (1), Thomas Becker (2), Toshiaki Suzuki (3), Spyros Denazis (3), Chiho Kitahara (3)
1.University College London, Dept. of Electrical Engineering, Torrington Place, London WC1E 7JE, UK; E-mail: {w.eaves, l.cheng,a.galis}@ee.ucl.ac.uk
2. Fraunhofer Institute for Open Communication Systems, Kaiserin-Augusta-Allee31, D-10589 Berlin, Germany; E-mail: becker@fokus.gmd.de
3. Hitachi Europe Ltd., 59 St. Andrews Street, Cambridge, CB2 3BZ, UK; E-mail: {chiho.kitahara, spyros.denazis, toshiaki.suzuki}@hitachi-eu.com

Abstract- **This paper proposes a new efficient mechanism for controlling and managing the resources within and around the active nodes [13]: routers and switches that have a legacy SNMP management system. Existing system such as ABLE [6] has an out-of-band management capability, which is insufficient for managing data flow as the flow progresses through the network. This paper proposes the use of the combined approach of SNAP [2] and SNMP [9] as an efficient network resource management mechanism on the FAIN active nodes [7],[12]. It has an in-band management approach in which each data flow will negotiate its next hop before it goes there; or it can create IPSec tunnels and modify routing table entries for the data flow.**

I. OVERVIEW

The main objective of the EU-IST FAIN (Future Active IP Network) [8] project is to develop the infrastructure for the dynamic and rapid deployment of new services. New services are implemented by the injection of codes on the active nodes. Each VE (Virtual Environment) can be assigned to a different service provider and each VE is guaranteed to have access to the necessary resources of the active nodes in order to support new services. Thus a RCF (Resource Control Framework) in the NodeOS is needed for the management and distribution of node resources to different VEs. The service provider must negotiate with the node operator for the resources requested by the VEs; the RCF will then partition the resources of the node to the VEs according to the agreement, and provide the VEs with guaranteed access to the partitioned resources. As depicted, EEs (Execution Environments) are simply treated as technologies (e.g., JVM) used to implement services that, in turn, may operate entirely in one of the three planes: control, management, or transport. EEs, and consequently services are encapsulated by VEs, which if connected together provide a proper virtual private network (VPN) on top of the network infrastructure. The VE is a necessary abstraction that is used only for the purpose of partitioning the resources of the AN Node. The VE concept is very important for the complete understanding of the delegation approach followed within the FAIN management framework. VEs enclose resources and access rights of principals using the active network. Consequently, all interactions of principals with the active network are checked against the access rights of their particular VE. VEs are built on top of the node operating system that involves the services of a number of extensions in the form of active node facilities that are required to support the instantiation and operation of different VEs. Through the extensions the NodeOS[1] offers facilities through the following facilities: 1) *Security* - The Security component in the active node is in charge of authorising all requests to the node API based on the security policies previously set by an authorised principal; 2) *Resource Access Control* - The Resource Control component within the active node receives request for allocating node-isolated resources (both computational and communication) to different principals; 3) *Demultiplexing* - The demultiplexing component is in charge of forwarding active packets to the corresponding EE and VE, based on the packet header information (e.g., ANEP header [10]) and a forwarding table used by this component; 4) *Active Service Provisioning* - ASP system is in charge of downloading active services into the active nodes or management stations when necessary; 5) *Virtual Environment Manager* (VEMgr) - Component includes activities that assist the policy-based management system to enforce its policies, e.g., monitoring of resources, event notification, VE instantiation, etc. More details on the FAIN AN Node architecture are beyond the scope of this document and can be found in [12]. Allocation of resources in the FAIN RCF is in the interest of the resource consumers i.e. VEs and the active applications, this is achieved by the application of higher-level policies; whereas both allocation and monitoring of resources are in the interest of the network management system [7].

II. INTRODUCTION

This paper describes a new resource control mechanism using SNMP across a network for controlling and managing the resources within and around the active node: routers and switches that have a legacy SNMP management system. In our approach, once a VE is given the authority to access the requested network resources, the resources can be obtained from any SNMP-enabled network devices. Finite state machines are implemented by active packets; these machines can then program a series of SNMP-enabled network devices in a synchronised manner, and provide a means for rollback: should any request for a network resource fail, then the fulfilled requests made earlier are released. Using this active packet mechanism, it will be possible to implement complex network reconfigurations; for instance, it can create IPSec tunnels and modify routing table entries to use it. The system uses the SNAP programming language to implement the finite state machines. It offers the facilities to issue the following SNMP commands that can be applied to network devices:

[1] NodeOS is an OS for active nodes and includes facilities for setting up and management of communications channels for inter–EEs, manages the router resources, provides APIs, and isolates EEs from each other.

0-7803-7632-3/02/$17.00 ©2002 IEEE

SET - set and changes their current operational configuration; GET - get the current device status; SET TRAP - set traps to report changes in device status. The system will also be possible to issue an instruction to any active extensions available in (or around) the active node; this will be used to demonstrate the loading of mobile software agents into a JVM (Java Virtual Machine) near the active node. These mobile software agents will be used for monitoring network conditions and reporting directly to any VE or twoot her management systems[2]. Security will be provided for by the standard mechanism used in SNMP: username, password and community. SNAP packets will be transmitted in cleartext, but the authority to activate the SNMP commands will be an active extension provided by the VE within the active node. Mobile agents will be loaded in a similar manner, the VE will be given the authority to load them. The mechanism in this latter case will be that available within the Grasshopper agency [4].

III.SY STEM DESIGN GOALS

A. Interceptor Paradigm

Active network management is the application area for this system. Active networking is an interceptor paradigm. It is difficult to develop applications that rely upon intercepting data packets because the interceptor must decode each data packet and its intention must be understood.

B. ABLE: Active Networking Out-of-Band [6]

The ABLE platform for network management is shown in figure 2. The ABLE platform used a router's packet filtering capabilities to supply ANEP UDP packets that contained a Java class to the system component "The Activator". The Activator reconstructed the Java class from the packet stream and forked itself. Its child then performed an `exec()` to launch a JVM (Java Virtual Machine) that could run the Java class intercepted it.

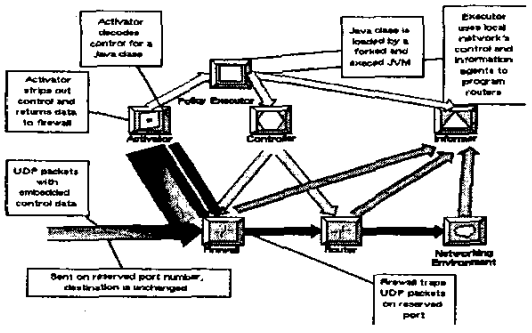

Figure 1 ABLE: An Example of the Interceptor Paradigm

This paper suggests that ABLE is deficient as an active networking system. It is useful for loading network monitors and managers into routers (or nearby management stations) wherever a customer data flow appeared in the network. It is not suitable for managing the data flow as it progresses through the network. It is, in effect, an out-of-band management facility. It did provide a mean to locate flow managers more effectively, but it did not provide a means to locate the flow itself. This is clearly a problem: a data flow is most unstable when it is first established. The network has to adapt to the load it presents and consequently the nodes through which the flow passes are most likely to change when the flow first presents itself to the network.

In contrast to traditional network management, what is needed for effective network management is an in-band management capability. Each flow will negotiate its next hop before it goes there. It will be seen that SNAP and SNMP can come close to achieving this: the SNAP packet will precede the data and go to the next hop, it will then establish a route for the data that will follow it. The information used by the SNAP packet to choose the route will state the intention of the data flow. For example:

- The data flow may be an HTTP request for a large resource to be delivered to the requesting machine.
- The data flow may be the start of a large system backup: sending large amounts of data to the accepting machine.

In both cases, the data flow will be asymmetric; in the former case, it will require a larger capacity in the reverse direction; in the latter, in the forward direction. The information that states the requesting machine's intent is only available at the edge of the network where the request is made - only the local network administration knows the capability and priority of its machines for a limited resource. The statement of intent is contained in an active packet that attempts to match its source with the sink of the data flow. The active packet can revise and choose how the source and sink impedances are matched.

C. Active Packets & Active Extension Technology: SNAP & SNMP

SNAP (Safe Network with Active Packets) [2] is a programming language that provides active packets at high level of safety. Essentially, SNAP packets are UDP packets that are embedded with assembly codes. As a SNAP packet traverses through the network, simple computations[3] such as to add and remove data to a stack within the packet can be performed. This is a genuinely active mode of operation. It will be seen that the application of SNAP within active network management is as a finite state machine that

[2] It is also planed that the extension of Grasshopper by IKV for the FAIN project can be exploited by the SNAP system. The Grasshopper extension allows agents to be transported using ANEP packets [4].

[3] SNAP programming language is an assembly language and it cannot perform any computations that are comparable in complexity to that of a C or Java program; nor can it support the wide range of data types that are available in these languages.

follows the progression of a reconfiguration of a network. Finite state machines do not need a complex runtime environment and SNAP will prove to be sufficient. SNMP (Simple Network Management Protocol) has been chosen as the active extension technology to work with SNAP for a number of reasons:

- It is the de-facto language of network management.
- SNMP v.3 provides cryptographically strong role-based access control.
- An extensible MIB and programmable SNMP v.3 agent have become available for conventional operating systems.
- Machines that run conventional OS are now capable to act as network routers as well.

The extensible MIB allows complex operations to be simplified to one macro instruction. In SNMP, the GET and SET commands can be thought of as operation codes for a programming language: LOAD and STORE. One could think of the object identifiers in the extensible MIB as memory locations. Simple programs can be written in SNAP to test the operational state and branch to different operation sequences.

IV. SYSTEM DESIGN

A. Injectors

Injectors inject programs into the network to reconfigure it. An injector will decide to inject code, because it has intercepted a request for a data flow from its own network. An injector intercepts and interprets some part of an application protocol. For example, the injector may intercept NFS (Network File System) requests, obtain the user identification contained within the NFS request and use that to priorities the use of bandwidth to deliver the data. It can also make use of the MAC address, the IP address, and the current network topology in its own administrative domain. In effect, it monitors the state of its own network and its connection with external networks.

When a new network condition develops, an injector will attach control information to the data flows it hopes to control:

- *Appearing flows* - A new network condition is engendered by a new data flow and the control information will be attached to the new flow.
- *Disappearing flows* - An injector may know that a flow, or a set of flows has finished: a machine or user or another network may have disappeared from the network.

An injector was demonstrated in the FAIN project: the ABLE platform used a packet filter to trigger the injection of code that constructed an IPSec tunnel [7], [11]. For the SNMP SNAP approach proposed here, a more sophisticated packet filter will be used. This will be a PromethOS system [3]. PromethOS is preferred means of providing node operating system plug-ins for active nodes. It is an extension of the Netfilter [10], a standard part of the Linux kernel. PromethOS packet filters will have a degree of feedback; they will be programmed by SNAP packets to wait for particular network events. The SNMP SNAP packets will inject code. These will be sent to the same host as the data that triggered the network event. All active SNAP-enabled routers will intercept these packets as they traverse the network. The SNAP packets should precede the data packets in the network, so that the data packets will not be able to traverse the network until the SNAP packets have a created a route for them. If this was the case, it would be desirable to implement another PromethOS module that performs packet spooling. The difficulty with operating injectors is to decide what code to inject.

B. Interceptors

Intercepting SNAP packets is more complicated than injecting them. These are the constraints: 1) The code has to be executed as quickly as possible, so that the packet can be quickly forwarded and minimise latency during the establishment of the data flow; 2) The functionality required will need to make use of active extensions on the node; 3) Active extensions require blocked I/O; 4) Blocked I/O cannot be performed in the same thread as the execution of the SNAP packet, because it would add too much latency. Because of this a new invocation model is proposed.

C. Active Extensions

SNAP provides a facility to access services within the SNAPS: CALLS (call service). A service is a C function. This will be used to dispatch the SNMP commands embedded in the SNAP program (figure 3). SNAP also provides a facility to read variables maintained by the SNAPD: SVCV (service variable collect). This will be used to return the state of SNMP variables. In this way, an SNMP command can be issued on one thread and the result can be returned, stored within the SNAPD and dispatched as the result in a subsequent SNAP packet. To illustrate, figure 3 shows an simplified example of the assembly codes that can be embedded in a SNAP packet to perform a SNMP SET:

```
here                ; push current node address
push 6              ; push the 6th stack value
push ((address1), (address2))
                    ; push the addresses onto the
                     stack
push ((\"set\"), (\"OID\"))
                    ; push a SNMP command onto the
                     stack
mktup 4             ; create a tuple
push bar            ; push result returned by the
                     service "bar"
bar:
istup               ; is it a tuple
bne 1               ; branch to "bar1" if yes
bar1:
calls \"testSvc\"
                    ; calls the service "testSvc"
svcv \"testVal\"
                    ; pushes the variable "testVal"
push 4455           ; pushes a port number
demux               ; sends a string (on 2nd TSV) to
                     the port (on TSV)
```

Figure 3 Assembly codes for SNMP SNAP SET

The first four instructions push the current node address; the 6th stack value e.g. authority in our example; the source and destination addresses; and the SNMP SET command onto the stack respectively. The fifth instruction creates a tuple, which holds the top four popped stack values and returns a value, which is an offset into the heap. bar checks whether it is a tuple; if true, then branches to bar1. bar1 is a service in the service table that formats the tuple as a string and puts it into a registered variable testVal – this variable now holds the "stringnified" SNMP command. The desire port number is then pusho nto the stack, and the SNMP command is then delivered to the desire port.

D. Invocation Model

A kernel-based SNAP packet processor is currently under development at the University of Pennsylvania [2]. This will be a node OS plug-in. This will be unable to invoke any active extensions outside of the kernel. Their proposed invocation model is for two SNAP-enabled nodes: A and B.

1. A: Execute SNAP instructions that do not invoke active extensions.
2. A: On reaching an instruction that does invoke an active extension:
   - A: Stop executing in the kernel and forward the packet to the next hop arriving at B.
   - A: Continue executing the packet program in user space. Invoke the active extension, wait for the result and, when it arrives, send it onto the next hop as a SNAP packet that only contains the result.
3. B: the SNAP packet sent by A is now executed. Two conditions may arise:
   - The result of the invocation of the active extension at the previous active node is required to progress the computation.
   - The result at the precious active node is not needed.
4. B: If the latter is the case, the packet can continue to execute.
5. *B: If the former is the case, then apply 2.*

In this way, SNAP packets can proceed very quickly through the network. A SNAP program will be in place at each active node waiting for the I/O to unblock at preceding nodes in the network. Diagrammatically, the situation is as given in the figure 4. At time interval, 1, packet p arrives, denoted p_1. Its blocking commands are invoked asynchronously and the packet is passed on. At time period p_r the result is ready.

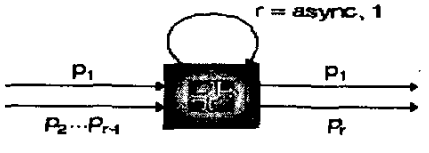

Figure 4 The invocation model

Other packets arrive, p_2 though to p_{r-1}. They may be forwarded or spooled. SNAP packets will almost certainly be dispatched, but it would be desirable to spool data packets. Eventually time period r arrives and the result of the SNAP operation invoked at time period 1 is available and it is dispatched immediately. If the data packet flow is being spooled, a release indication would be sent by the next hop, presumably after it has received and processed the result of the operation of $p_1$arr iving after period r. A more sophisticated analysis than this would show that the synchronisation of the operation invocation and the arrival of the result form a self-organising protocol - similar to Djikstra's leader election protocol for communications bus synchronisation.

In effect, the interaction between the kernel and user space SNAP interpreters requires two new primitives within SNAP: FORK and JOIN [5]. These will be implicit in the calls to the active extensions: CALLS and SVCV. The design of the FORK and JOIN primitives is common to many OS. An identifier will be needed to specify the thread to join. The usual problem of finding a unique identifier in an open distributed system will be faced. Also SNAP will require two stacks: a supervisor stack used for synchronisation and a user stack used for the SNAP program. The operation of the kernel SNAP interpreter will be an atomic copy, increment the program counter and forward.

E. Implementation

The kernel space SNAP interpreter is not currently *available, but the proposed invocation model (FORK and* JOIN) can be proved using the current user space overlay network architecture of SNAP. Interceptors will be SNAPD running on active routers. They will listen on several SNAP control ports. At the time of writing, the SNAP interpreters are not part of a system that has packet spooling, which is still an experimental of the Linux kernel [11].

V.TH E SMNP-SNAP APPLICATION SCENARIO

The below collaboration diagram shows how a SNAP packet implementing a finite state machine could be used to create an ad-hoc network. There are four routers in this system: r, s, t, u. Each of which must move to its respective operational state: s1, s2, s3, s4. The network to be constructed is a sub-network that passes all of its traffic through an ATM switch. The traffic must be conditioned so that the bit-rate limited virtual channel carrying the traffic does not arbitrarily drop cells and corrupt the IP packets. To simplify the management of the traffic conditioning, the traffic is carried in an IP in IP tunnel and it is conditioned. It is then unencapsulated and given to the ATM switch's IP interface. Typically, this network might be used to support ADSL *access for a neighbourhood.*

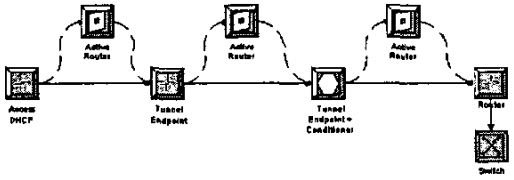

Figure 6 The network diagram for the SNMP SNAP Application Scenario

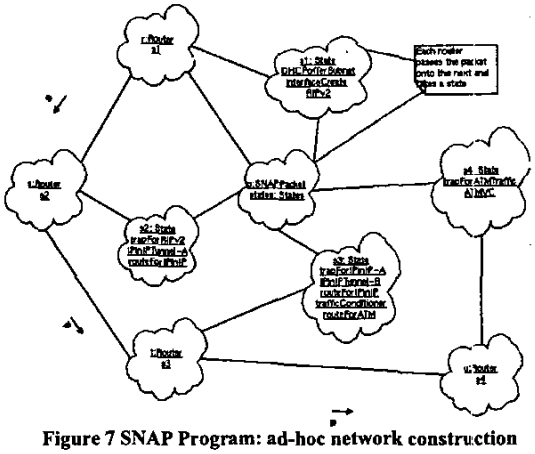

Figure 7 SNAP Program: ad-hoc network construction

The progression of the states of construction is this:

- S1 - The router r, supporting DHCP, RIP version 2 and, say, two 100BaseTx interfaces offers a sub-network to client machines. It creates an interface for that sub-network on one of its 100BaseTx interfaces and announces a route to the sub-network with RIP version 2 on the other interface. It injects the SNAP packet.
- S2 - An upstream router, s, receives the SNAP packet and is told to wait for a RIP version 2 event- the announcement of the new sub-network. In response it will create an IP in IP tunnel endpoint for it. It passes the SNAP packet on.
- S3 - The next upstream router, t, receives the SNAP packet and is told to construct the other IP in IP tunnel endpoint and to apply a traffic conditioner to the tunnel and to route the traffic to an ATM switch.
- S4 - The router with the ATM interface creates a route for the unencapsulated traffic of the sub-network.

This is the sort of network construction task that many system administrators must perform. SNAP is used to carry the instructions and to record the changes of state of the network. The instructions can be at conceptually a high level, the extensible SNMP agent allows many simple instructions to be grouped together. The states correspond exactly to the construction of the system. Clearly, this task could be automated, the only variables are: IP sub-network to be supported; IP tunnel endpoints addresses; traffic conditioning parameters; ATM interface IP address. A set of each of these could easily be embedded into a number of SNAP programs. The DHCP routers would be given at least one each to inject into the network when a client machine starts to use the network they manage the addresses for.

VI. CONCLUSION

This paper describes a new mechanism using SNAP language for controlling and managing the resources within and around the active nodes: routers and switches that have a legacy SNMP management system. The SNAP system propagates SNMP command execution through a network lends itself to mass production of SNAP programs to construct large numbers of network. It exploits active networking by having control information move with the data it must support. This is an efficient mechanism for invocation of any active extensions outside of the kernel. It provides synchronised changes in state, thus disruption caused by transient operational states will be minimised. The effect of the latter could be entirely eliminated with the use of packet spooling. This would be synchronised to an acknowledgement message that the network has attained its new state.

ACKNOWLEDGEMENT

This paper describes work undertaken and in progress in the context of the FAIN – IST 10561, a 3 years project during 2000-2003. The IST programme is partially funded by the Commission of the European Union.

REFERENCES

[1] R. Keller, S. Choi "An Active Router Architecture for Multicast Video Distribution", http://www.tik.ee.ethz.ch/~keller/infocom2000.pdf
[2] J. Moore "Safe and Efficient Active Packets", technical report MS-CIS-99-24, USA, Oct 1999. http://www.cis.upenn.edu/~jonm/
[3] L. Ruf, "Design of PromethOS", FAIN technical report, WP3-ETH-002-PromethOS, ETH Zurich, Dec 2001. http://www.promethos.org/
[4] J. Dittrich, C. Weckerle "ANEP Extension for Grasshopper", FAIN technical report, Germany, Oct 2001. http://www.grasshopper.de/
[5] D. Lea, "A Java Fork/Join Framework", New York, USA. http://g..oswego.edu/dl/papers/fj.pdf
[6] D. Raz, "An Active Network Approach for Efficient Network Management" IWAN99, July 99, http://www.cs.bell-labs.com/who/ABLE/
[7] FAIN Deliverable 2 "Initial Active Network and Active Node Architecture", May 2001 http://www.ist-fain.org
[8] FAIN Public WWW Server http://www.ist-fain.org
[9] NET-SNMP community, http://net-snmp.sourceforge.net
[10] Netfilter Core Team, http://www.netfilter.org
[11] M. Blaze, J. Ioannidis "Trust Management for IPSec", NDSS 2001, San Diego, Feb 2001 http://www.crypto.com/papers/
[12] A. Galis, B. Plattner, J. M. Smith, S. Denazis, H. Guo, C. Klein, J. Serrat, J. Laarhuis, G.T. Karetsos, C. Todd "A Flexible IP Active Networks Architecture" in the Proceedings Second International Working Conference, IWAN'2000 – Japan, October 2000, ISBN 3-540-41179-8, Springer Verlag
[13] D. Tennenhouse, D. Wetherall – "Towards an active network architecture" Computer Communications Review, 26, 2 (1996), pp 5-18