Active Networks: Towards Service Enabling Unified Networks

Jamal Hadi Salim (Computing Technology Labs),
Muhammad Jaseemuddin (Computing Technology Labs)

Keywords: Active Networks, Service Enabled Networks, Unified Networks, Java, IP

Abstract

Mounting competition among service providers is making it imperative for the providers to introduce new services and extend features of their existing ones to attract new customers. On the other hand, competition among network equipment vendors, such as Nortel Networks, is driving Network Elements to become a commodity item. In this scenario, the major revenue generator for Nortel will be to provide features in the Network Elements that help its customers to introduce new services. In this paper we present a case for Service Enabled Networks (SEN), which we consider as vital for Nortel Networks' Unified Networks initiative to address the growing customer demands. We also present Active Networks as a key technology for SEN. Active Networks provide execution engines and a mechanism to safely deploy new code in the Network Elements to extend their functionality for service provisioning.

Introduction

The next step in the evolution of communication networks is the support for services. The idea of developing Service Enabled Networks (SEN) is not new. This evolutionary path was pursued in the circuit-switched telephony world using Advanced Intelligent Networks (AIN) as the enabling technology. Unfortunately, the technology has not achieved its promise because it took too long for vendors to come to an agreement on AIN models. Moreover, given that AIN is limited to database queries on user profiles, it is not attractive for the data networks.

Today's IP core network is *dumb*. The funtionality of the Network Element is fixed. For example, Routers forward packets between networks based on OSI layer 3 (IP) headers, leaving any processing or special treatment of packets to occur at the edges or in the end systems. Unfortunately, user demands and requirements are constantly changing, resulting in continuous adhoc solutions to address the *problem du-jour*. For example, assurance of Quality of Service (QoS) through the IETF DiffServ initiative, requires additional processing that is not traditionally incorporated into the router.

Over time, transport and application layer (OSI layer 7) functionalities are also beginning to appear in routers. As an example: voice codecs in Cisco routers, email virus scanners (Blazenet routers[15]), firewalls which require peeking into headers above those of layer 3 and web-caching which is starting to appear in some newer edge routers. These type of processing ideally do not belong in a router.

Nortel Networks' Unified Networks initiative is meant to address growing user demands as exemplified above. We believe that more flexible Network Elements whose functionalities will continue to be extended easily to accommodate our customers' requirements is a key value-add to future Unified Networks.

We define Service Enabled Networks (SEN) as networks which are ready for service (re)programmability. In essence, SENs provide a vehicle towards Unified Networks.

Active Networks (AN) is a key enabling technology for SEN. It allows the injection of customized programs into the Network Elements. The Network Elements provide programmable engines to allow dynamic and fast deployment of programs/protocols. In essence, the network provides a computing engine. *The network is the computer*. It facilitates the introduction of new services based on deployed programs, as driven by innovation and as dictated by the customers. Services are created, configured, deployed, and reconfigured rapidly and dependably. AN has the potential of reducing the standardization process immensely. Guarantees that a service will be acceptable are not delayed by political motivations of other vendors and operators (as is in the standards bodies today). As an analogy, in the PC market today, the bus interface (e.g., PCI or ISA) is standardized and innovation or customer requirements define which PCI card providing a certain service will become popular. A good example is the Sound blaster (for sound cards) or the Hayes AT command set (for modems). One could say that AN attempts to *PC-fy the network*.

Business Value

It is apparent that once the Network Element becomes a commodity item, the major revenue generator for Nortel will be to provide features in the Network Elements that help its customers to easily introduce new service. The richness of the services provided by an xSP will be the differentiating factor from their competitors. For Nortel, the distinguishing factor will be the service enabling features on the Network Elements that

we make available to all our customers.

The drift towards Service Enabled Networks (SEN) is inevitable. Programmable network nodes as suggested by AN provide a good context for SEN. This implies that some services which are currently running on an end-to-end basis will either move into the network fully or will be aided by certain services within the network. Moore's Law and the ever increasing amount of cheap bandwidth will be a contributing factor towards the acceptance of network processing. A further implication of this means that the differences in what are known today as *end systems* and *Network Elements* will become less broad from an application point of view.

Companies like Microsoft (through their Millenium project[3]) and Sun (through their Jini project[4]) are today interested in maintaining the Network Element being dumb, so that new services can be offered using server farms in the end systems which are their primary market. However, we can provide greater value to our customers (so they can generate more revenue) if we can pull services into the network. It also opens opportunity (for Nortel) to get into the services creation business for us. It is, therefore, in our best interests to promote the use of Network Elements for service programing.

Some of our competitors are already moving towards service enabling their Network Elements. Cisco recently announced[5] Jini-enabling their Cable-modem routing devices. They have announced some rudimentary service where the cable company offers the cable user "storage" space on their servers. 3COM, HITACHI, Intel and a few other vendors are also investigating the potential of Active Networks.

The availability of a processing engine for services programming within a Network Element provides a rich environment for fast adaptability. It implies flexible customization of the Network Element. It means opportunities for innovation and fast deployment of the newly created services. We feel this is a definite requirement in a fast adapting environment as defined by Unified Networks.

Position

There are several approaches to AN. A radical approach is to inject intelligent packets carrying code (referred to as capsules)[1]. The capsule's code is executed at the Network Elements enhancing the capability of those elements by making use of certain services already deployed at that network node. In the capsule concept, anything above the physical layer is reprogrammable and reconfigurable by the capsules for a period of time. This makes IP just another option that can be layered.

A more realistic approach (in our view) is the Switchware[2] proposal where the network is generic enough such that it can be reprogrammed to suit a desired use. Mostly, the programming is for delivering certain services via downloading of the code to deliver that service to the Network Element as enforced by a network administrator. Selected packets are then conditioned using the downloaded code by the programmed node.

The CTL has been involved in AN research since February 1997. In 1998, we studied some of the academic research implementations in existence to come up with an educated opinion. For more details visit our AN website[14].

Our investment in the evaluation of some proof-of-concept AN implementations has helped shape our thoughts for an AN design. As a result, we have come up with a set of requirements which we feel are necessary to make AN a commercial viability. The experience has also enlightened us on the kind of AN services we feel can be deployed with the design.

Requirements To Making AN Acceptable

Separating dream research ideas from industry realities, we have come up with the following requirements to make Active Networks acceptable as a business case:

- The node **MUST** run on an RTOS (not a general purpose OS such as Linux).
- The node **MUST** run the Java Virtual Machine (JVM) as the Active Engine. It is the only VM with the right properties that is widely accepted by the industry at the moment. The potential of addressing weaknesses it currently has is more viable as opposed to more workable but academic solutions such as OcaML or Erlang. The availability of the JVM in hardware is an added bonus.
- When deployed on a Network Element, the Active Engine **MUST NOT** affect the packet forwarding path. For example, if deployed on a router, all packets not requiring use of any services other than forwarding **MUST NOT** be affected by the processing of packets in the Active Engine.
- The design **MUST** accommodate legacy equipment such as current routers and switches. The active node should be able to be a stand-alone box sitting beside a router providing it the execution environment.
- A global framework tying together services **SHOULD** be based on some existing standard (RSVP, CORBA etc.).
- The ability to deploy code or new services on the network **MUST** initially be limited to an administrator or some trusted third party. This helps to reduce the problem to be equivalent to a normal hot-swap Network Element upgrade that is done by an administrator today. As a consequence, code that delivers a service is deployed to the Network Elements; thereafter incoming packets are conditioned by these deployed services either to aid an

application or deliver the application via the cumulative effect of multiple aggregated active nodes' conditioning.

- Both the local node and global framework designs **MUST** take into consideration security. Although security is built in from the beginning (as opposed to the after-thought in the case of IP), there is still a great deal of concern given mobile code is used to customize elements within the network. Authentication and authorization *are of a lesser concern* because the technologies are well known and are easily built in. The major security concern is hostile or buggy code. Some form of formalism (lacking in Java) would be a good value add.

  The design **MUST** provide levels of privileged authorization based on what code can be accessed and/or downloaded by a particular node user. Namespace security is difficult to do in Java and therefore careful deliberation is a major requirement. Global data is shared between threads.

  The design for authentication **SHOULD** include various mechanisms ranging from simple hash signatures to total encryption with trusted consultation to allow differentiated privileges.

  The design **SHOULD** also include various authorization techniques.

  The design **MUST** have a built-in way to fall back and terminate threads of execution which are misbehaving. Java provides sandboxing which half solves this problem. Fall back (built into Erlang for example) is more challenging to do in Java.

  The problem of denial of service type of attacks on node resources could be relieved by validating the effects of a piece of code. Currently this is considered a hard problem (NP complete). Proof Carrying Code which allows formal method checks at the active node is the best known solution for program verification. This is similar to hardware verification before the chip is manufactured. Techniques such as those built into the OcaML environment where extreme static checking is enforced at compile time help eliminate most of the unintentional buggy code but do not alleviate the intentional hostile coding. Type checking before loading at the node also helps to further reduce the problem (but increases the loading time). Generally, if static checking is enforced at compile time, this would be done only on less privileged code.

  In essence, there **SHOULD** be a *verifier* of some sort built into the node design and static type checking **SHOULD** be enforced at byte code compilation time.

- Resource management **MUST** include CPU time and memory space, in addition to bandwidth. *Quality of Service means all those three parameters.* This will enable Service providers to offer the service concept of RAM and CPU for lease (as is currently suggested by Jini). The Service Level Agreement(SLA) must include at least those three resources.

- Consideration for scalability is a high priority. The design **MUST** be able to handle a multitude of services loaded into the execution environment.

- Policies on how long code is active before being flushed out **MUST** be in place (probably as part of the SLA). SLA contracts should also include what to do with the de-activated code after the SLA time has expired. For example should the code be discarded? Should it still be stored in RAM for later use for a cost or, even cheaper, should it be stored on off-line secondary storage? Consider a situation where a customer wants to lease RAM and CPU space for three hours for a proprietary (probably more efficient) video conferencing protocol. The customer wishes to store some proprietary codec code to handle heterogeneous network links of the video conference attendees. Obviously not many people can use this service other than the single customer and the service provider would not be interested in keeping the code around after the three hours.

AN Applications/Services Which Can Be Deployed Today

Most of our choices are driven by the desire to have incrementally acceptable changes as well as general acceptability of the selected applications. It became obvious to us, during our analysis , that there are some services/applications that can benefit from information that is only available inside the network. Conversely, the network can support the application better by having more knowledge about it. A simple case is the decision of which of the packets belonging to a service are more important (and thus should have preferential treatment in situations of congestion).
The following have been identified as good applications which can benefit from an available Active Network.

Control Plane applications

These are applications that run traditionally on the slow path of the Network Element.

- Network Management:

Consider the DiffServ DEN type of architecture where requirements are not yet decided or continuously being redefined. One could, for example, not only deploy the schema but also the code that services it as well and have the liberty of continuously changing that code. *The action code becomes part of the policy*.

The flexibility to change the protocol used to communicate to the policy server (e.g., from COPS to COPS-2 or to proprietary) is inherent.

Back in the 80s SNMP took off because programmable systems were feared to be too complex [16]; however, current improvements in code mobility and understanding of programmable systems provides new avenues.

Current scalability problems of SNMP could be easily addressed (polling; most of the time there is nothing to report back). Consider a situation where you send a single capsule to collect statistics from devices which it encounters on its path up to a certain depth. Fission of the packet happens at junction points onto several directions. Later fusion of collected data at the fission points happens; unimportant data is filtered and the information is fused into a new packet which eventually reaches the network management station. This a mere simple example of how the availability of an Active engine could significantly benefit the network management arena.

- Routing Protocols:

In the long run one could move path discovery (OSPF, RIP etc.) into the programmable part of the node. This will let the fast path worry only about fast forwarding (as is done in large routers today) and add the value of quickly being able to deploy newer extensions.

Application specific Processing

Today end-to-end code mobility is already in place: postscript, applets, CORBA, Jini, Millenium, etc. Tremendous value and flexibility has been shown at that level. The driving factor to put application/service awareness into the network is driven by the fact that only applications are best aware of how their packets should be treated. These applications require help from the network to do certain levels of processing for them to enhance their end-to-end performance. For example, the Cisco voice codecs for voice over IP fall under this category.

- Mobile services/servers:

This category covers services that can be enhanced by help from the network.

A simple example (one provided by MIT ANTS) is that of a stock quote service where different users receive different (time) granularity of stock updates depending on their subscription fees. The intelligence of who receives what level and which stocks, is delegated to the execution engine on a router closest to the subscriber.

Commercial network infrastructures such as Sun's Jini or UMTS' VHE[7] would be a good target to collaborate with. AN enabled Network Elements could provide them an underlying assistance to alleviate end to end traffic as well as allow them to be more distributed. Currently all these offerings deliver services on an end to end (client/server) basis.

- Protocol boosting:

Programmable Engines on a router provide a very good opportunity for dynamic conditioning/boosting of protocol behavior.

A quick look at Transmission Control Protocol (TCP) reveals that it was really designed for networks with low Bit Error Rates (maximum of about 1%) [8,9]. Any other losses are considered to be caused by congestion. The assumption is invalid in wireless or even satellite networks where the BERs are higher. This forces many *fudges* such as SNOOP-TCP etc. to help alleviate these oversights.

A boundary router capable of offering services for boosting TCP depending on the media of the next hop would be a good fit. It would be a wiser investment to have a programmable engine which can be used to condition TCP for either a wired to radio wireless network or wired to satellite network. Both have problems with TCP but require different boosting.

- Access Device Interfaces:

This is the idea of reprogrammable layer 2 Medium Access Control as was originally initiated by Angelo at CTL in 1997 and currently being evangelized by DSP manufacturers[10].

Consider a scenario where a PDA is hooked up with a modem and dials into an access device containing a modem pool. The first step could be to download the appropriate signalling protocol (for example 56Kbps signalling). Then you load to the PDA a small PPP protocol engine which proceeds to establish the IP connection as well as load only the necessary protocols on an as needed basis (e.g., HTTP or FTP etc.).

- Multimedia:

A big boost could be provided by AN for these type of applications. A simple example is to have Active nodes at strategic positions in the network fuse different types of applications and synchronize them as instructed. An offered service here could be delivery of differentiated video scenes with synchronized audio and text based on a user's taste or billing profile. Consider two users watching the same movie (same plot) but different scenes.

Information fission and data translations/formatting (e.g., compression by switching an audio codec when going to a slower network or formatting images for a 3COM palm pilot) also fall under these categories.

A case in point is the selective discard of MPEG's B frames over I frames at time of congestion.

- Reliable Multicasting:

Reliable Multicasting tends to be application specific. Cisco is working with TIBCO to deploy PGM. What if our customers could deploy any vendor's solution? This is what the programmable solution provides the customer.

Software deployment/upgrades

AN features such as security can also be used to provide a new safer way of mass delivery of software regardless of the type of targeted systems. One could, for example, deploy a single active packet which went around querying the scheduler type in a node. The response is used to deploy a newer version to all nodes below a certain version.

Observations and Comparisons

The design goal of MIT ANTS [1] is to facilitate deploying new protocols and services inside the network. ANTS is a toolkit to design and implement an active network. It is implemented in Java. It consists of some core classes and a runtime system. The runtime system is installed on end systems (computers, etc.) as well as on network nodes (routers, switches, etc.). Thus, ANTS provides an end-to-end support to build the active network. A protocol in ANTS is implemented using capsules. For a protocol, a capsule is a unit of transfer through the network carrying data as well as code that is executed at each active node. The capsule code is serialized and transferred through the network packet. The capsule code is downloaded from the previous active node. The inband code distribution technique in ANTS is implemented using the same capsule delivery mechanism that is used for protocol capsules.

Netscript [11] is an implementation of AN at Columbia University. It consists of two parts: a textual language, and a toolkit consisting of a set of Java classes to which the language compiles. A NetScript network consists of one or more NetScript engines running on one or more network nodes. Each engine is composed of a number of packet processing elements called boxes. These boxes can be dispatched to remote engines and connected with other resident boxes to form bigger boxes and more complicated packet processing devices. These boxes typically perform some protocol function, such as packet header analysis or packet demultiplexing.

PLAN (Packet Language for Active Networks) [12] is a simple functional programming language. It is based on a subset of ML with some added primitives to express remote evaluation. The PLAN language provides no mechanism for non-finite looping or recursion.

A network of PLAN nodes(PLANet), consists of several PLAN Active Routers running on one or more networked machines. Each of these active routers has a built in PLAN language evaluator than can execute PLAN programs. PLAN programs form the body of the Active packets, which are injected into a PLAN network by host applications and evaluated on remote routers. The PLAN system also includes a number of dynamically installable services which can be used by PLAN programs executing on the node. Users of the system may also write their own services and dynamically install them in routers.

Our analysis of these systems has helped gain insight and shaped our design philosophy. We are influenced by a desire to maintain IP while making the node programmable. We feel that the ANTS idea is too revolutionary to be deployed. We find the Switchware approach of out-of-band code distribution more acceptable because it makes the distribution more secure. However, the sample services ANTS deploys are very exciting. While PLAN uses the Switchware approach, and its concept of partitioning services' code is appealing, we find that it has poor security features and its services are not well defined. Its use of the OcaML language, which is acceptable to the research community, does not make much business sense to us. Netscript's use of a dataflow model was appealing to us. Dataflow provides a simple model for concurrency and synchronization. Data may flow only through the formal interfaces of the Netscript box and there is no global or shared concurrent state between boxes. This avoids the need for primitives for synchronizing shared state between boxes and multiple boxes can run concurrently. NetScript also cleanly separates the namespaces of loaded boxes such that each box only has access to itself and classes loaded using Java's main class loader. There was absolutely no protection against denial of service attacks in the preliminary Netscript version we looked at. The analysis reports are available on our web site.

Figure 1 shows our opinion of a high level description of what one would see in deployed in routers. Note that the Active Engine would reside in a background engine so as to not affect the fast packet path.

The classifier is programmed to identify packets requiring a certain service. Incoming packets matching the criteria are redirected to the Active Engine (shown by the arrow labelled 1) in the control plane for further processing. Packets not interested in the service are forwarded as usual to the proper egress interface. After the packets are serviced by the Active Engine, they are forwarded back to the classifier (indicated by the arrow labelled 3). Since the fast path is normally done in hardware, we are expecting minimal impact on packets not interested in extra services.

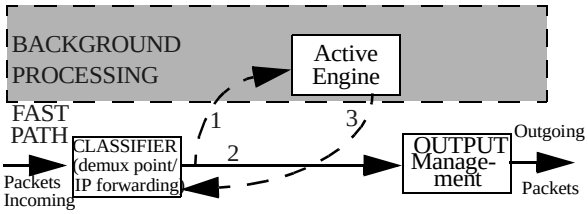

Figure 1. High-level logical dissection of an Active Router

The background processing might become an issue since most Network Elements tend to have limited processing capabilities which are mostly burdened with distributed path discovery protocols (such as routing). We are therefore proposing an *Active Card* which is inserted in a slot on an existing Network Element such as a router or a routing switch. The Active Engine will reside on the Active Card.

We have experimented with the above model since september 1998 using a software based VxWorks ethernet switch. The Active Engine Java code is activated from a Voyager server[13]. Services code is stored in an LDAP server and is downloaded, validated and authenticated before being activated.

Conclusion

In this paper we have presented a mechanism to lay a infrastructure to Service Enable Unified Networks based on the concept of Active Networks. Our protoype work has shown that we can build the foundation which will provide dynamic, highly-reconfigurable, pay-per-use extensions to features in a Network Element whether they are specific to voice or data, regardless of whether that they are proprietary or open standards.

Network Element product groups within Nortel Networks collaborating with us can benefit from the insight we have gained over the years that we have been involved in this area.

We feel that Nortel can provide better customer value by pulling services into our boxes instead of letting end system vendors like Microsoft or Sun have control over them. We are not ruling out working with any of these vendors to come up with application enhancements within the network; however, we are concerned that once the Network Element becomes a commodity item, we loose control. It is our opinion that Nortel Networks should be moving aggressively into this arena.

References

[1] MIT ANTS, an Active Node Transfer System, http://www.sds.lcs.mit.edu/activeware/ants/

[2] Univ. of Penn, The SwitchWare Project, http://www.cis.upenn.edu/~switchware/

[3] Micorsoft, The Millenium project, http://research.microsoft.com/sn/Millennium/

[4] Sun Microsystems. JINI Project. http://www.javasoft.com/products/jini

[5] Announcement at: http://www.javasoft.com/features/1999/01/convergence.html

[6] DARPA. Active Networks Website. http://www.sds.lcs.mit.edu/darpa-activenet/

[7] Virtual Home Environment. http://47.96.10.142/mobile/umts/umtsframe.html

[8] W Richard Stevens. RFC 2001. IETF. Nov 1997

[9] V Jacobson. Congestion Avoidance and Control. V Sigcomm, 1988.

[10]Texas Instruments. TI and Java: Bringing the Revolution in Network Intelligence to Wireless Systems http://www.ti.com/sc/docs/wireless/java/techbrf.htm

[11] Columbia University. NetScript: A Language and Environment for Programmable Networks. http://www.cs.columbia.edu/dcc/netscript/

[12] Univ. of Penn. A Packet Language for Active Networks. http://www.cis.upenn.edu/~switchware/PLAN/

[13] Voyager. http://www.objectspace.com

[14]CTL AN website at: http://ctl.ca.nortel.com/ctl/activities/an/

[15] http://www.blaze-net.com/

[16] Craig Patridge. IEEE Network Special issue on Active Networks. 1998

Active Networks: Towards Service Enabling Unified Networks

Jamal Hadi Salim (Computing Technology Labs),
Muhammad Jaseemuddin (Computing Technology Labs)

Keywords: Active Networks, Service Enabled Networks, Unified Networks, Java, IP

Abstract

Mounting competition among service providers is making it imperative for the providers to introduce new services and extend features of their existing ones to attract new customers. On the other hand, competition among network equipment vendors, such as Nortel Networks, is driving Network Elements to become a commodity item. In this scenario, the major revenue generator for Nortel will be to provide features in the Network Elements that help its customers to introduce new services. In this paper we present a case for Service Enabled Networks (SEN), which we consider as vital for Nortel Networks' Unified Networks initiative to address the growing customer demands. We also present Active Networks as a key technology for SEN. Active Networks provide execution engines and a mechanism to safely deploy new code in the Network Elements to extend their functionality for service provisioning.

Introduction

The next step in the evolution of communication networks is the support for services. The idea of developing Service Enabled Networks (SEN) is not new. This evolutionary path was pursued in the circuit-switched telephony world using Advanced Intelligent Networks (AIN) as the enabling technology. Unfortunately, the technology has not achieved its promise because it took too long for vendors to come to an agreement on AIN models. Moreover, given that AIN is limited to database queries on user profiles, it is not attractive for the data networks.

Today's IP core network is *dumb*. The funtionality of the Network Element is fixed. For example, Routers forward packets between networks based on OSI layer 3 (IP) headers, leaving any processing or special treatment of packets to occur at the edges or in the end systems. Unfortunately, user demands and requirements are constantly changing, resulting in continuous adhoc solutions to address the *problem du-jour*. For example, assurance of Quality of Service (QoS) through the IETF DiffServ initiative, requires additional processing that is not traditionally incorporated into the router.

Over time, transport and application layer (OSI layer 7) functionalities are also beginning to appear in routers. As an example: voice codecs in Cisco routers, email virus scanners (Blazenet routers[15]), firewalls which require peeking into headers above those of layer 3 and web-caching which is starting to appear in some newer edge routers. These type of processing ideally do not belong in a router.

Nortel Networks' Unified Networks initiative is meant to address growing user demands as exemplified above. We believe that more flexible Network Elements whose functionalities will continue to be extended easily to accommodate our customers' requirements is a key value-add to future Unified Networks.

We define Service Enabled Networks (SEN) as networks which are ready for service (re)programmability. In essence, SENs provide a vehicle towards Unified Networks.

Active Networks (AN) is a key enabling technology for SEN. It allows the injection of customized programs into the Network Elements. The Network Elements provide programmable engines to allow dynamic and fast deployment of programs/protocols. In essence, the network provides a computing engine. *The network is the computer*. It facilitates the introduction of new services based on deployed programs, as driven by innovation and as dictated by the customers. Services are created, configured, deployed, and reconfigured rapidly and dependably. AN has the potential of reducing the standardization process immensely. Guarantees that a service will be acceptable are not delayed by political motivations of other vendors and operators (as is in the standards bodies today). As an analogy, in the PC market today, the bus interface (e.g., PCI or ISA) is standardized and innovation or customer requirements define which PCI card providing a certain service will become popular. A good example is the Sound blaster (for sound cards) or the Hayes AT command set (for modems). One could say that AN attempts to *PC-fy the network*.

Business Value

It is apparent that once the Network Element becomes a commodity item, the major revenue generator for Nortel will be to provide features in the Network Elements that help its customers to easily introduce new service. The richness of the services provided by an xSP will be the differentiating factor from their competitors. For Nortel, the distinguishing factor will be the service enabling features on the Network Elements that

we make available to all our customers.

The drift towards Service Enabled Networks (SEN) is inevitable. Programmable network nodes as suggested by AN provide a good context for SEN. This implies that some services which are currently running on an end-to-end basis will either move into the network fully or will be aided by certain services within the network. Moore's Law and the ever increasing amount of cheap bandwidth will be a contributing factor towards the acceptance of network processing. A further implication of this means that the differences in what are known today as *end systems* and *Network Elements* will become less broad from an application point of view.

Companies like Microsoft (through their Millenium project[3]) and Sun (through their Jini project[4]) are today interested in maintaining the Network Element being dumb, so that new services can be offered using server farms in the end systems which are their primary market. However, we can provide greater value to our customers (so they can generate more revenue) if we can pull services into the network. It also opens opportunity (for Nortel) to get into the services creation business for us. It is, therefore, in our best interests to promote the use of Network Elements for service programing.

Some of our competitors are already moving towards service enabling their Network Elements. Cisco recently announced[5] Jini-enabling their Cable-modem routing devices. They have announced some rudimentary service where the cable company offers the cable user "storage" space on their servers. 3COM, HITACHI, Intel and a few other vendors are also investigating the potential of Active Networks.

The availability of a processing engine for services programming within a Network Element provides a rich environment for fast adaptability. It implies flexible customization of the Network Element. It means opportunities for innovation and fast deployment of the newly created services. We feel this is a definite requirement in a fast adapting environment as defined by Unified Networks.

Position

There are several approaches to AN. A radical approach is to inject intelligent packets carrying code (referred to as capsules)[1]. The capsule's code is executed at the Network Elements enhancing the capability of those elements by making use of certain services already deployed at that network node. In the capsule concept, anything above the physical layer is reprogrammable and reconfigurable by the capsules for a period of time. This makes IP just another option that can be layered.

A more realistic approach (in our view) is the Switchware[2] proposal where the network is generic enough such that it can be reprogrammed to suit a desired use. Mostly, the programming is for delivering certain services via downloading of the code to deliver that service to the Network Element as enforced by a network administrator. Selected packets are then conditioned using the downloaded code by the programmed node.

The CTL has been involved in AN research since February 1997. In 1998, we studied some of the academic research implementations in existence to come up with an educated opinion. For more details visit our AN website[14].

Our investment in the evaluation of some proof-of-concept AN implementations has helped shape our thoughts for an AN design. As a result, we have come up with a set of requirements which we feel are necessary to make AN a commercial viability. The experience has also enlightened us on the kind of AN services we feel can be deployed with the design.

Requirements To Making AN Acceptable

Separating dream research ideas from industry realities, we have come up with the following requirements to make Active Networks acceptable as a business case:

- The node **MUST** run on an RTOS (not a general purpose OS such as Linux).
- The node **MUST** run the Java Virtual Machine (JVM) as the Active Engine. It is the only VM with the right properties that is widely accepted by the industry at the moment. The potential of addressing weaknesses it currently has is more viable as opposed to more workable but academic solutions such as OcaML or Erlang. The availability of the JVM in hardware is an added bonus.
- When deployed on a Network Element, the Active Engine **MUST NOT** affect the packet forwarding path. For example, if deployed on a router, all packets not requiring use of any services other than forwarding **MUST NOT** be affected by the processing of packets in the Active Engine.
- The design **MUST** accommodate legacy equipment such as current routers and switches. The active node should be able to be a stand-alone box sitting beside a router providing it the execution environment.
- A global framework tying together services **SHOULD** be based on some existing standard (RSVP, CORBA etc.).
- The ability to deploy code or new services on the network **MUST** initially be limited to an administrator or some trusted third party. This helps to reduce the problem to be equivalent to a normal hot-swap Network Element upgrade that is done by an administrator today. As a consequence, code that delivers a service is deployed to the Network Elements; thereafter incoming packets are conditioned by these deployed services either to aid an

application or deliver the application via the cumulative effect of multiple aggregated active nodes' conditioning.

- Both the local node and global framework designs **MUST** take into consideration security. Although security is built in from the beginning (as opposed to the after-thought in the case of IP), there is still a great deal of concern given mobile code is used to customize elements within the network. Authentication and authorization *are of a lesser concern* because the technologies are well known and are easily built in. The major security concern is hostile or buggy code. Some form of formalism (lacking in Java) would be a good value add.

  The design **MUST** provide levels of privileged authorization based on what code can be accessed and/or downloaded by a particular node user. Namespace security is difficult to do in Java and therefore careful deliberation is a major requirement. Global data is shared between threads.

  The design for authentication **SHOULD** include various mechanisms ranging from simple hash signatures to total encryption with trusted consultation to allow differentiated privileges.

  The design **SHOULD** also include various authorization techniques.

  The design **MUST** have a built-in way to fall back and terminate threads of execution which are misbehaving. Java provides sandboxing which half solves this problem. Fall back (built into Erlang for example) is more challenging to do in Java.

  The problem of denial of service type of attacks on node resources could be relieved by validating the effects of a piece of code. Currently this is considered a hard problem (NP complete). Proof Carrying Code which allows formal method checks at the active node is the best known solution for program verification. This is similar to hardware verification before the chip is manufactured. Techniques such as those built into the OcaML environment where extreme static checking is enforced at compile time help eliminate most of the unintentional buggy code but do not alleviate the intentional hostile coding. Type checking before loading at the node also helps to further reduce the problem (but increases the loading time). Generally, if static checking is enforced at compile time, this would be done only on less privileged code.

  In essence, there **SHOULD** be a *verifier* of some sort built into the node design and static type checking **SHOULD** be enforced at byte code compilation time.

- Resource management **MUST** include CPU time and memory space, in addition to bandwidth. *Quality of Service means all those three parameters.* This will enable Service providers to offer the service concept of RAM and CPU for lease (as is currently suggested by Jini). The Service Level Agreement(SLA) must include at least those three resources.

- Consideration for scalability is a high priority. The design **MUST** be able to handle a multitude of services loaded into the execution environment.

- Policies on how long code is active before being flushed out **MUST** be in place (probably as part of the SLA). SLA contracts should also include what to do with the de-activated code after the SLA time has expired. For example should the code be discarded? Should it still be stored in RAM for later use for a cost or, even cheaper, should it be stored on off-line secondary storage? Consider a situation where a customer wants to lease RAM and CPU space for three hours for a proprietary (probably more efficient) video conferencing protocol. The customer wishes to store some proprietary codec code to handle heterogeneous network links of the video conference attendees. Obviously not many people can use this service other than the single customer and the service provider would not be interested in keeping the code around after the three hours.

AN Applications/Services Which Can Be Deployed Today

Most of our choices are driven by the desire to have incrementally acceptable changes as well as general acceptability of the selected applications. It became obvious to us, during our analysis , that there are some services/applications that can benefit from information that is only available inside the network. Conversely, the network can support the application better by having more knowledge about it. A simple case is the decision of which of the packets belonging to a service are more important (and thus should have preferential treatment in situations of congestion).
The following have been identified as good applications which can benefit from an available Active Network.

Control Plane applications

These are applications that run traditionally on the slow path of the Network Element.

- Network Management:

Consider the DiffServ DEN type of architecture where requirements are not yet decided or continuously being redefined. One could, for example, not only deploy the schema but also the code that services it as well and have the liberty of continuously changing that code. *The action code becomes part of the policy*.

The flexibility to change the protocol used to communicate to the policy server (e.g., from COPS to COPS-2 or to proprietary) is inherent.

Back in the 80s SNMP took off because programmable systems were feared to be too complex [16]; however, current improvements in code mobility and understanding of programmable systems provides new avenues.

Current scalability problems of SNMP could be easily addressed (polling; most of the time there is nothing to report back). Consider a situation where you send a single capsule to collect statistics from devices which it encounters on its path up to a certain depth. Fission of the packet happens at junction points onto several directions. Later fusion of collected data at the fission points happens; unimportant data is filtered and the information is fused into a new packet which eventually reaches the network management station. This a mere simple example of how the availability of an Active engine could significantly benefit the network management arena.

- Routing Protocols:

In the long run one could move path discovery (OSPF, RIP etc.) into the programmable part of the node. This will let the fast path worry only about fast forwarding (as is done in large routers today) and add the value of quickly being able to deploy newer extensions.

Application specific Processing

Today end-to-end code mobility is already in place: postscript, applets, CORBA, Jini, Millenium, etc. Tremendous value and flexibility has been shown at that level. The driving factor to put application/service awareness into the network is driven by the fact that only applications are best aware of how their packets should be treated. These applications require help from the network to do certain levels of processing for them to enhance their end-to-end performance. For example, the Cisco voice codecs for voice over IP fall under this category.

- Mobile services/servers:

This category covers services that can be enhanced by help from the network.

A simple example (one provided by MIT ANTS) is that of a stock quote service where different users receive different (time) granularity of stock updates depending on their subscription fees. The intelligence of who receives what level and which stocks, is delegated to the execution engine on a router closest to the subscriber.

Commercial network infrastructures such as Sun's Jini or UMTS' VHE[7] would be a good target to collaborate with. AN enabled Network Elements could provide them an underlying assistance to alleviate end to end traffic as well as allow them to be more distributed. Currently all these offerings deliver services on an end to end (client/server) basis.

- Protocol boosting:

Programmable Engines on a router provide a very good opportunity for dynamic conditioning/boosting of protocol behavior.

A quick look at Transmission Control Protocol (TCP) reveals that it was really designed for networks with low Bit Error Rates (maximum of about 1%) [8,9]. Any other losses are considered to be caused by congestion. The assumption is invalid in wireless or even satellite networks where the BERs are higher. This forces many *fudges* such as SNOOP-TCP etc. to help alleviate these oversights.

A boundary router capable of offering services for boosting TCP depending on the media of the next hop would be a good fit. It would be a wiser investment to have a programmable engine which can be used to condition TCP for either a wired to radio wireless network or wired to satellite network. Both have problems with TCP but require different boosting.

- Access Device Interfaces:

This is the idea of reprogrammable layer 2 Medium Access Control as was originally initiated by Angelo at CTL in 1997 and currently being evangelized by DSP manufacturers[10].

Consider a scenario where a PDA is hooked up with a modem and dials into an access device containing a modem pool. The first step could be to download the appropriate signalling protocol (for example 56Kbps signalling). Then you load to the PDA a small PPP protocol engine which proceeds to establish the IP connection as well as load only the necessary protocols on an as needed basis (e.g., HTTP or FTP etc.).

- Multimedia:

A big boost could be provided by AN for these type of applications. A simple example is to have Active nodes at strategic positions in the network fuse different types of applications and synchronize them as instructed. An offered service here could be delivery of differentiated video scenes with synchronized audio and text based on a user's taste or billing profile. Consider two users watching the same movie (same plot) but different scenes.

Information fission and data translations/formatting (e.g., compression by switching an audio codec when going to a slower network or formatting images for a 3COM palm pilot) also fall under these categories.

A case in point is the selective discard of MPEG's B frames over I frames at time of congestion.

- Reliable Multicasting:

Reliable Multicasting tends to be application specific. Cisco is working with TIBCO to deploy PGM. What if our customers could deploy any vendor's solution? This is what the programmable solution provides the customer.

Software deployment/upgrades

AN features such as security can also be used to provide a new safer way of mass delivery of software regardless of the type of targeted systems. One could, for example, deploy a single active packet which went around querying the scheduler type in a node. The response is used to deploy a newer version to all nodes below a certain version.

Observations and Comparisons

The design goal of MIT ANTS [1] is to facilitate deploying new protocols and services inside the network. ANTS is a toolkit to design and implement an active network. It is implemented in Java. It consists of some core classes and a runtime system. The runtime system is installed on end systems (computers, etc.) as well as on network nodes (routers, switches, etc.). Thus, ANTS provides an end-to-end support to build the active network. A protocol in ANTS is implemented using capsules. For a protocol, a capsule is a unit of transfer through the network carrying data as well as code that is executed at each active node. The capsule code is serialized and transferred through the network packet. The capsule code is downloaded from the previous active node. The inband code distribution technique in ANTS is implemented using the same capsule delivery mechanism that is used for protocol capsules.

Netscript [11] is an implementation of AN at Columbia University. It consists of two parts: a textual language, and a toolkit consisting of a set of Java classes to which the language compiles. A NetScript network consists of one or more NetScript engines running on one or more network nodes. Each engine is composed of a number of packet processing elements called boxes. These boxes can be dispatched to remote engines and connected with other resident boxes to form bigger boxes and more complicated packet processing devices. These boxes typically perform some protocol function, such as packet header analysis or packet demultiplexing.

PLAN (Packet Language for Active Networks) [12] is a simple functional programming language. It is based on a subset of ML with some added primitives to express remote evaluation. The PLAN language provides no mechanism for non-finite looping or recursion.

A network of PLAN nodes(PLANet), consists of several PLAN Active Routers running on one or more networked machines. Each of these active routers has a built in PLAN language evaluator than can execute PLAN programs. PLAN programs form the body of the Active packets, which are injected into a PLAN network by host applications and evaluated on remote routers. The PLAN system also includes a number of dynamically installable services which can be used by PLAN programs executing on the node. Users of the system may also write their own services and dynamically install them in routers.

Our analysis of these systems has helped gain insight and shaped our design philosophy. We are influenced by a desire to maintain IP while making the node programmable. We feel that the ANTS idea is too revolutionary to be deployed. We find the Switchware approach of out-of-band code distribution more acceptable because it makes the distribution more secure. However, the sample services ANTS deploys are very exciting. While PLAN uses the Switchware approach, and its concept of partitioning services' code is appealing, we find that it has poor security features and its services are not well defined. Its use of the OcaML language, which is acceptable to the research community, does not make much business sense to us. Netscript's use of a dataflow model was appealing to us. Dataflow provides a simple model for concurrency and synchronization. Data may flow only through the formal interfaces of the Netscript box and there is no global or shared concurrent state between boxes. This avoids the need for primitives for synchronizing shared state between boxes and multiple boxes can run concurrently. NetScript also cleanly separates the namespaces of loaded boxes such that each box only has access to itself and classes loaded using Java's main class loader. There was absolutely no protection against denial of service attacks in the preliminary Netscript version we looked at. The analysis reports are available on our web site.

Figure 1 shows our opinion of a high level description of what one would see in deployed in routers. Note that the Active Engine would reside in a background engine so as to not affect the fast packet path.

The classifier is programmed to identify packets requiring a certain service. Incoming packets matching the criteria are redirected to the Active Engine (shown by the arrow labelled 1) in the control plane for further processing. Packets not interested in the service are forwarded as usual to the proper egress interface. After the packets are serviced by the Active Engine, they are forwarded back to the classifier (indicated by the arrow labelled 3). Since the fast path is normally done in hardware, we are expecting minimal impact on packets not interested in extra services.

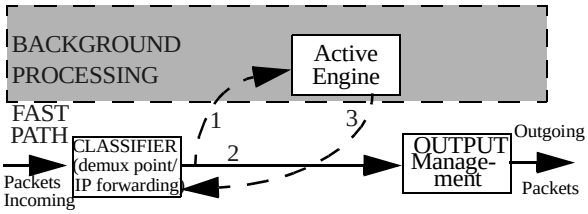

Figure 1. High-level logical dissection of an Active Router

The background processing might become an issue since most Network Elements tend to have limited processing capabilities which are mostly burdened with distributed path discovery protocols (such as routing). We are therefore proposing an *Active Card* which is inserted in a slot on an existing Network Element such as a router or a routing switch. The Active Engine will reside on the Active Card.

We have experimented with the above model since september 1998 using a software based VxWorks ethernet switch. The Active Engine Java code is activated from a Voyager server[13]. Services code is stored in an LDAP server and is downloaded, validated and authenticated before being activated.

Conclusion

In this paper we have presented a mechanism to lay a infrastructure to Service Enable Unified Networks based on the concept of Active Networks. Our protoype work has shown that we can build the foundation which will provide dynamic, highly-reconfigurable, pay-per-use extensions to features in a Network Element whether they are specific to voice or data, regardless of whether that they are proprietary or open standards.

Network Element product groups within Nortel Networks collaborating with us can benefit from the insight we have gained over the years that we have been involved in this area.

We feel that Nortel can provide better customer value by pulling services into our boxes instead of letting end system vendors like Microsoft or Sun have control over them. We are not ruling out working with any of these vendors to come up with application enhancements within the network; however, we are concerned that once the Network Element becomes a commodity item, we loose control. It is our opinion that Nortel Networks should be moving aggressively into this arena.

References

[1] MIT ANTS, an Active Node Transfer System, http://www.sds.lcs.mit.edu/activeware/ants/

[2] Univ. of Penn, The SwitchWare Project, http://www.cis.upenn.edu/~switchware/

[3] Micorsoft, The Millenium project, http://research.microsoft.com/sn/Millennium/

[4] Sun Microsystems. JINI Project. http://www.javasoft.com/products/jini

[5] Announcement at: http://www.javasoft.com/features/1999/01/convergence.html

[6] DARPA. Active Networks Website. http://www.sds.lcs.mit.edu/darpa-activenet/

[7] Virtual Home Environment. http://47.96.10.142/mobile/umts/umtsframe.html

[8] W Richard Stevens. RFC 2001. IETF. Nov 1997

[9] V Jacobson. Congestion Avoidance and Control. V Sigcomm, 1988.

[10]Texas Instruments. TI and Java: Bringing the Revolution in Network Intelligence to Wireless Systems http://www.ti.com/sc/docs/wireless/java/techbrf.htm

[11] Columbia University. NetScript: A Language and Environment for Programmable Networks. http://www.cs.columbia.edu/dcc/netscript/

[12] Univ. of Penn. A Packet Language for Active Networks. http://www.cis.upenn.edu/~switchware/PLAN/

[13] Voyager. http://www.objectspace.com

[14]CTL AN website at: http://ctl.ca.nortel.com/ctl/activities/an/

[15] http://www.blaze-net.com/

[16] Craig Patridge. IEEE Network Special issue on Active Networks. 1998