Survey of Active Network Research

Dragos Niculescu
dnicules@cs.rutgers.edu

July 14, 1999

Abstract

Active networks address the problem of slow network evolution, when compared with the evolution of applications. The active approach builds programmability into the network, so that new protocols and functionalities may be added later. While classical, "passive" routers all implement the same functionality, the active routers all implement the same computational model. The active technology is motivated twofold by user need: the increasingly popular applications that require custom processing in the network, and the backlog of Internet services that are yet to become ubiquitous (IPv6, multicast, mobility). The main issues in active networking are mobility, efficiency, safety and backwards compatibility.

In this paper we explore the status of research in the area active networks, the proposed architectures, the issues they address, and the currently under way standardization efforts to provide a common framework for the active nodes.

1 Introduction

Active networking [WT96] is a step beyond the traditional networking technology in which the routers have the role to merely forward datagrams and update routing tables. Its primary goal is to modify the switching infrastructure of the network on the fly, and more generally, to allow customized computation to be run in the network. Although some argue that putting functionality in the network would contradict the "end to end argument", a number of widely used applications may suggest that this is not necessarily a bad thing. Here we refer to ad hoc solutions that in time became de facto applications, such as web proxies, firewalls, multicast, media gateways and overlay networks. What also comes as an argument for the active technology is the slow evolution of networks: it takes years to deploy a new protocol on the Internet, and this only after it passed through the slow process of standardization. Active networks allow deployment of new protocols dynamically, on a per domain basis or even per application or per user basis.

Problems posed to the the active networking technologies, as identified in [TGSK96] are mobility, efficiency and safety. Due to the heterogeneous nature of the Internet today, mobility is a must, as the edge of the network has no knowledge about the particular architecture of a router. Efficiency is a concern because running customized code could potentially slow down the non-active traffic that requires from the internal nodes only the basic service of forwarding. It is therefore desirable that the active networking could be installable as a service amongst other differentiated services, or, a different point of view may argue that differentiated services may be implemented under the active architecture. Safety is an issue if entities from the network have access to the shared resources of a node, which now involve not only bandwidth, but also processing time, and memory local to the node. Another concern regarding the deployment of the active networks is the inherent latency that would be introduced by the extra processing in nodes. However, the bandwidth and the computational power grew respecting Moore's law for the last 25 years, but the propagation delay is in many cases close to the physical bound. For this reason, in network computation would be a small fraction of the propagation delay in WANs. Let us consider the following example: if a typical delay between two nodes 10000km apart would be of 50ms, increasing this delay to 55ms would be acceptable if services better than best effort would be provided. Tomorrow's processors will execute at over 1GHz clock speed which would make possible the execution of five million cycles on the routers along the route. Even if this amount of computation is not enough for certain compression or encryption tasks, it could certainly improve tasks based on routing, caching or congestion control. The challenge is to design an active node architecture that is scalable for both forwarding and processing.

There are currently a number of architectures proposed by the active networking groups, and efforts are under way to standardize both the operating system support and an encapsulation protocol to allow active protocols to run over "passive" IP networks, such as Internet. [BCZ98a] classifies the architectures proposed on the granularity of control, statefullness, and language expressive power. ANTS [WGT98] offers a Turing-complete machine model at the active router, therefore the possibility for each user to execute any new code. At the other end of the spectrum, DAN[DP98] only allows the user to call functions already installed at the node. In terms of granularity, packets may modify the entire node behavior, like in ANTS, or only the flow to which they belong. To these criteria, we may add one that is essential for deployment: accessibility. ANTS allows the injection of the code from anywhere on the network, while Switchware [Sco98] only allows adding of new functionality by the local authorities at each node, the user being only able to make use of the services, and compose those services in a controlled manner.

1.1 Roadmap

The rest of the survey is organized as follows: Section 2 presents a number of applications that would benefit of active solutions, thus motivating the need for research in the area of active networks. Section 3 explores a number of technologies that make active networking possible, or that could improve future active architectures. Sections 4 presents the main architectures for active nodes proposed today and their features. Section 5 provides a comparison between the presented architectures and the way services may be implemented, deployed and composed, and section 6 summarizes with the conclusions of the survey.

2 Motivation (Applications and Active Approaches)

Mobile code technologies, such as Java, came to an widespread use largely due to network related applications, agents and applets that are used in a compiled form across platforms. But applets and agents are designed to deploy new functionality only to the edges of the network, while certain problems, such as congestion control, or transcoding need meta-information that is timely available only in-network. Application specific congestion control is an often used example: applications that run customized functions in the network can handle dependencies among application data units, and can differentiate between the importance of units. Sending an MPEG stream over a passive datagram network gives no control over what packets are dropped in case of congestion, while an MPEG specific transcoding method would drop less important frames, or even frames that are of no use at all once some previous frame was lost.

There are applications for which performance is evaluated in specific metrics instead of usual network metrics, such as delay or bandwidth. The auction site application [WLG98] defines the performance as the number of successful bids processed per second as opposed to the total number of bids issued to the server. Stock caching application [LWG98], allows a finer tuning for both the promptness in delivery and the recency of a stock quote, while defining the performance measure in quotes delivered per second, instead of the usual measure in megabits per second.

The following table summarizes applications mentioned in the active networking literature and the solution adopted in the current Internet.

Application	**Non-active solution**
firewall	flow specific filters are installed on edge routers
web proxy	application level: Squid, Harvest, manually deployed
scalable auction server	requires AS code to be installed in all the routers in a radius around the server
routing around congestion	differentiated services, reservations, heuristics in the routers
multicast, reliable multicast	MBONE
transcoding	media gateways
congestion control	end to end policies (TCP)
protocol boosters	manually install boosters/deboosters
network management	configurability only in the limits allowed by the installed system
WAN cooperative caches	application level: Squid, Harvest

sensor data merging/dispatching	application specific, can only be done at edges, or at specialized servers
conjunctive/disjunctive data streams	processed in servers, or at the edge hosts
custom stock tickers	end to end, using agents at both ends
overlay networks	XBone, specific bone
Quality of service	RSVP, differentiated services
mobile routing	mobile IP, snoop TCP
mSMTP, mHTTP, CTCP	not possible end to end
Ad hoc Multicast	use several unicast links

All these applications, or problems, cannot be solved efficiently in an end to end manner. The need to customize in-network nodes to accommodate the newest applications materialized in time in various modifications to the router software, such as firewalls, or, when access to the node is not possible, in routing performed at application level , such as web proxies, transcoding, or overlay networks. Some of the problems cannot be even solved in an end to end manner - the most popular examples here include multicast, mobility and QoS guarantees.

Tomorrow's internet will see not only an increase in the total volume of traffic, but also the dominance of the multimedia traffic. Long distance companies use packet switching to lower their costs, music is being sold in binary form on the internet, conferencing over the internet is becoming increasingly popular - all these longer lived flows are more likely to benefit from the processing capabilities of the routers, than the shorter lived flows from email and web browsing we see today.

2.1 Routing, Bridging

The deployment of multicast and reliable multicast has been long awaited by the Internet community as multiuser applications and games are becoming more popular. Active Reliable Multicast [wHLGT98] is a protocol implemented on top of ANTS that provides reliable delivery to all members subscribed to a multicast group. The issues in reliable multicast are caching of data, NACK implosion and answering individual repair requests. ARM handles them in a way that is close to optimal due to the fact that caches can be placed at the most appropriate locations in the multicast tree, and repair requests will be handled as close as possible to the lossy links. ARM uses soft local state in the routers to keep track of the repair data that is requested often, and to fuse NACK messages. Advantages of ARM are that it introduces a small increase in wide-area end to end latency and that its deployment issues are solved by the active architecture. Caching at routers may pose a scalability problem, but that is inherent to the reliable multicast, not to the active solution.

Active Bridging [ASNS97] is an application based on Switchware architecture [Sco98], from University of Pennsylvania, that proves the utility of the active technology in a local network. A short description of Switchware is presented in section 4.2 in this survey. The active bridge may be programmed on the fly to become from a simple buffered repeater, a bridge supporting multiple spanning tree algorithms. This experiment makes a case for the incremental deployment of network protocols showing that versions of the spanning tree algorithm may be updated without shutting down the network. The approach is also able to validate the newly inserted protocols, and in case of failure, to fall back to the old version of the protocol.

2.2 Caching

Active nodes allow packets to leave soft state behind, and sometimes to allocate larger amounts of memory, which makes the active architecture a good environment to implement applications that require in network caching. An example is presented in [LG98], where a web caching infrastructure is deployed over active nodes to reduce latency for warm documents. The solution addresses the problems present in hierarchical caches: multiple transitions between application level and network level (because routing is performed in the application layer), repeated miss processing and large number of hops traversed. The cause for high latency in the hierarchical approach is that the protocol used to query caches (ICP) is connectionless and requires require one extra round trip, and TCP SYN used by HTTP to open the connection doesn't contain the URL. The idea for active caching is that participating active routers keep redirecting pointers towards the nearby caches, and as a request goes on the shortest path to the server, it may be redirected to a cache that has the URL. Pointers kept in the active routers are the message digests of the actual URLs. Each router reports URL requests to the cache which then installs a redirection pointer in the router. This solution also enables a form of load balancing: when busy, a server enables path-tracing for all its requests for a monitoring period. All the incoming routes are then collected in a tree and appropriate documents are sent to the caches placed at the root of loaded trees.

Self organizing caches [BCZ98b] are another approach to active caching that uses many small caches instead of a few large ones. Any cache will have a specific set of objects that can be cached in it, and sets do not overlap with one another. The path between the client and the server is partitioned in chunks of a given radius, so that an item is cached only once inside a radius. To obtain an even distribution of objects on the path, a modulus function function is used. A cache will keep pointers to items stored in nearby caches, and these pointers are used to reroute the requests. The scheme is referred to as modulo caching with lookaround. Simulations have shown that for a fixed topology, there is a unique value for the ratio between the memory used for items and the memory used for pointers, that would give the least latency. Too large a memory for the lookaround cache makes the routes longer, and leaves too little space for the actual objects in the nodes. Not enough pointer memory won't give enough strength to the lookaround.

[Joh98] presents a new transport protocol CTCP(Client side TCP), based on NACKs from the clients. Active nodes along the route between the client and the server perform caching for all datagrams. When NACKs from the clients are received by a node that has a cached copy, the request can be solved without involving the server. However, this solution has the disadvantage that due to the NACK transport style, the server should maintain the entire copy of the document. "Good" cache sizes needed for these active routers are up to 200MB - this value is obtained by multiplying the bandwidth of the router with the time to detect a loss and retransmit a packet.

2.3 Congestion control

Congestion is a problem for which an active solution would be appropriate because some packets are queued, therefore delayed or even dropped anyway, and therefore, active processing on these packets won't incur an extra processing delay noticeable at the edges of the network. As the product bandwidth x propagation delay increases, it becomes harder to deal with congestion in an end to end manner. This makes a point for the active capabilities in routers that would allow internal nodes to deal with congestion locally. It is also better for the application to control losses, than to leave this to lower levels of the protocol stack.

[BCZ96a] describes how the congestion is handled by application specific code injected in the network in order to obtain better throughput and better network utilization for an MPEG video transmission. The scheme uses a fixed priority between MPEG frames B, P and I (in this order of importance), and then applies three possible policies at each node:

- Discard-Other: packets with lower priority are discarded to fit an arriving packet in a full output queue. Dependencies of the dropped packet are also dropped
- Discard-Own: all packets in the same frame with a discarded packet are discarded
- Discard-On-Arrival: when a packet is dropped, all subsequent dependent packets are dropped

The basic datagram service competes with this model in terms of performance only if the utility of the data at the receiver is more fine grained: that is, if the receiver is able to use any parts of an incompletely received frame. Another possible advantage for the application control schemes is that they are able to implement some labeling of the packets to indicate permitted manipulation at the active nodes, which is somehow similar to differentiated services schemes.

In [BCZ96b], an active node model designed specifically for congestion control is presented. Functions may be executed on any packet in the protocol stack: on application units, transport units, IP datagrams or datagram fragments(frames). The functions available at each node are fixed: buffering and rate control, unit-level dropping, media transformation, multi-stream interaction. Each capsule carries a list of function addresses, and a list of labels, specifying data to be used by the functions. Soft state is accessed by using tags that are aged and eventually discarded when not used. Because functions are fixed, security is not an issue. The active networking service here is defined to be best-effort, that is, in the worst case it should not be worse than the service offered by the passive network.

2.4 Virtual Networks, Network Management

Virtual, or overlay networks are a way to build a logical network on top of another network for purposes like: emulation of physical resources, development of new protocols, reserved services and customized topologies. X-Bone[TH98] is a system for automated deployment and management of overlay networks. Examples of overlay include MBone for multicast IP, 6Bone for IPv6, and

ABone for active networks. The basic idea of an overlay network is to use encapsulation to provide a virtual infrastructure on top of existing internet. This requires routing software to be installed at selected sites - this is a common point with the active networks. X-Bone can bootstrap and manage active networks, or X-Bone can work on top of services offered by active networks. However, active routers, once deployed, would allow a higher degree of customization, beyond that of deployment of virtual networks. The main problems X-Bone addresses are manual configuration of the overlay network, and for distributed applications, application level routing and resource discovery. The X-Bone architecture comprises of three components:

- overlay managers run at user locations, at the edges of the network and control the deployment and configuration of the overlay; they are distributed GUI applications
- resource daemons run at resource locations (routers) and provide external access to resources; routers require configuration of addresses for virtual interfaces - when global addresses are used, the daemons manage the address space
- multicast control protocol used for locating resources

X-Bone operates at IP layer, so resources should have IP addresses. Addresses may be either local to the overlay, which allows richer overlays, but requires additional routing, or global to the Internet which is easier to configure, but harder to allocate globally. In order to support X-Bone, hosts should support multiple IP addresses, links should support tunnels and routers should support partitioned route tables and multiple forwarding engines. These requirements are very similar to those for the NodeOS that has to support several execution environments for active nodes.

[Yec96] Netscript is a project developed at Columbia University and aims at managing a virtual active network on top of existing internet. Netscript views the network as a collection of VNEs (Virtual Network Engines) linked by VLs(Virtual Links). An agent is a program that can be dynamically dispatched and executed at at remote system. NetScript language is an object oriented, data-flow language used to glue together pieces written in a low level language that optimize the common paths. In this respect the approach is very similar to the one adopted in Scout/Joust, with the difference that in Netscript, programs/devices may be added/removed on the fly. Code loading is automatic, based on signatures that describe links between modules. Applications of NetScript include network monitoring, SNMP agents and custom ATM signaling protocols.

[Van97] describes another possible application in network management that would benefit from an active architecture: a method for defense against address spoofing and certain DoS attacks. One of the effective DoS attacks is SYN-flooding. Its popularity is due to its simplicity, exploiting the way TCP servers handle pending connections, and to the fact that effective protection would require either modification of TCP, or deployment of firewalls -which is a somehow active solution. TCP connection are established with a three-way handshake: the client sends a SYN request to the server, which answers with a SYN-ACK as soon as it can satisfy the request. The client acknowledges again and the connection is established at this point. After sending back the SYN-ACK, the server queues the requests in a queue for pending connections. The attack consists in sending a large number of SYN requests and never acknowledging any of them. The server only discards pending requests after a period of timeout to allow slower clients to respond. The attacker may send SYN packets with a spoofed return address, so that SYN-ACK never reaches him. The active solution is to deploy an active agent that will push itself with one hop closer to the attacker with each new SYN received. The agent is deployed only when the number of pending connections is high.

Smart Packets [SZJ$^+$98] is another project that use active networking for network management. Their approach is however more general, defining a virtual machine for the CISC-like assembly language Spanner, and a C-like high level language - Sprocket. Programs are described in Sprocket and then compiled to Spanner before being sent in the network. The implementation is oriented towards network management by having built-in instructions to access resources on routers, non-persistent state across packets and imposing the restriction that a smart program must not be fragmented, so it should fit in 1K (one Ethernet frame). Programs are sent and executed at a remote host and results are encapsulated with ANEP and send back. Smart packets are authenticated with cryptographic methods before entering the execution environment of the active node.

3 Enabling Technologies

From the technologies that enable implementation of active networking, code mobility is the most important one. Due to popularity and uniformity across implementations Java is used in many implementations. The next table summarizes the performance estimated for several bytecode technologies, measured as slowdown from C code compiled to the native code.

Bytecode	Slowdown
Java	1500%
Caml	1000%
Omniware	1.8%-23%
ANDF	3%
Dis (Inferno)	?%

ANDF (Architecture Neutral Distribution Format) is not actually a bytecode, but a way to stop the compilation before generating the architecture specific code and use this intermediate code as a method to share binary distributions across platforms. Optimizations not specific to the platform are performed on the intermediate code, which is then linked to produce a portable executable. An installer is then used to produce an architecture specific binary from the distribution binary, applying machine specific optimizations. In the context of active networks, ANDF can be used mainly to install dynamic libraries and architecture independent code repositories. Installation time, which is actually the last phase of a normal compilation may be prohibitive for installation of the code on the fly, as in the case of bytecode.

3.1 Java, JIT, WAT

The most important feature of Java for active networking purposes is bytecode mobility. Java source code is compiled to a bytecode that is interpreted on the target machines. Java virtual machine is stack based, which imposes a high performance penalty, as shown in the above table. Various techniques have been developed to overcome this problem: JIT (just in time compiling) translates the bytecode to native code when a class is loaded. Disadvantages of this method are that the speed of resulted code is lower than that of a code compiled directly to native code, and the extra time used for translation. WAT (way ahead of time compilation) is a method that performs off line compilation of code that never changes, thus having the chance to apply more efficient optimizations. In an active node where users are allowed to inject code on the fly, precompilation such as JIT or WAT may incur high overheads, but there are architectures, such as Joust, which make use of these methods in a different manner.

3.2 Scout/Joust

Scout [MP96] is a communication oriented operating system which has the path as an explicit abstraction. If we consider protocols involved in some application representing nodes in a graph, then paths are flows of data between two nodes. A certain graph configuration is compiled into the Scout kernel, but paths are instantiated dynamically by applications. This approach allows installation of optimized code for frequently used paths, and such optimizations may expose and exploit non-local context. For example, the system places data in buffers that are accessible to all modules along the path, with one initial copy. Paths may decide to discard unnecessary work early - e.g. a video frame whose deadline passed. The more invariants the system knows, the more optimized the path can be, and the more layers a path works across, the more opportunities are there to optimize.

Joust [HPB$^+$97] is a re-implementation of Java virtual machine on Scout to add support for low level fine grained control over resources. JVM is implemented as a module that may be part of a Scout path. JVM was extended with path specific routines - Java applications are able to move data between modules, and to create paths. Paths that span the JVM module are scheduled by a priority scheduler so that JVM does not interfere with real-time paths. Joust supports WAT(way ahead of time) compilation of classes that are not changed (AWT) and JIT for injected code. This solution shows that Java is good to assemble services in an active node, instead of implementing an entire service or processing capsules.

3.3 Omniware

Virtual machines are not all of them slow, and for now, there are not many other ways to provide code mobility. Omniware [Col96] is a RISC based virtual machine with some CISC features that provides performance close to the native code. Mobility of code is achieved by compilation to the virtual machine OmniVM bytecode, which is then loaded by OmniRun - the dynamic translator. OmniRun has the role to translate the high level features of OmniVM in sequences optimized for particular architectures. OmniVM supports a segmented virtual address space, and uses software fault isolation - the host has complete control over the memory usage behavior of a module. The slowdown compared with the native code running on the same RISC machine ranges between 1.8% and 23%. The high performance of OmniVM, as compared with other bytecodes is due to the design of the virtual architecture, which is basically an intersection between current popular RISC architectures, thus allowing fast translation, with some CISC instructions, to allow the dynamic translator to use machine specific optimizations.

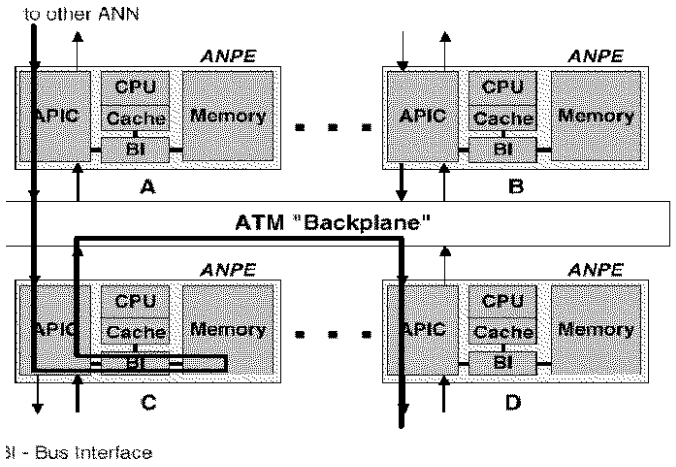
3.4 Proof-Carrying Code

It is clear that in an active environment, where users across the network share not only routers' bandwidth, but also CPU and memory, a method to control global resource consumption is necessary. Also, if users are allowed to install code on the fly in the active node, such code must abide by some restrictions imposed by the execution environment at the node. Proof Carrying Code(PCC)[Nec97] is a validation method where the untrusted producer of the code produces a *safety proof* that attests the adherence to a previously established policy. The consumer can then easily and quickly validate the proof, once, and possibly off-line. The consumer specifies *safety rules* and *interface.* Safety rules describe all authorized operations, their preconditions and the interface describes calling conventions and invariants. Type invariants are established for each loop, or for each destination of a backward jump. A function is given with the type signature (one precondition and one postcondition) and with the invariants annotated. The verification computes a predicate for each instruction, in one pass. PCC basically extends ML's compile time type checking to assembly language and defines safety in terms of types. Readable memory and writable memory are also defined as types. The method makes some hard guarantees: any program with a valid verification condition will reference only memory locations that are defined by the typing rules.

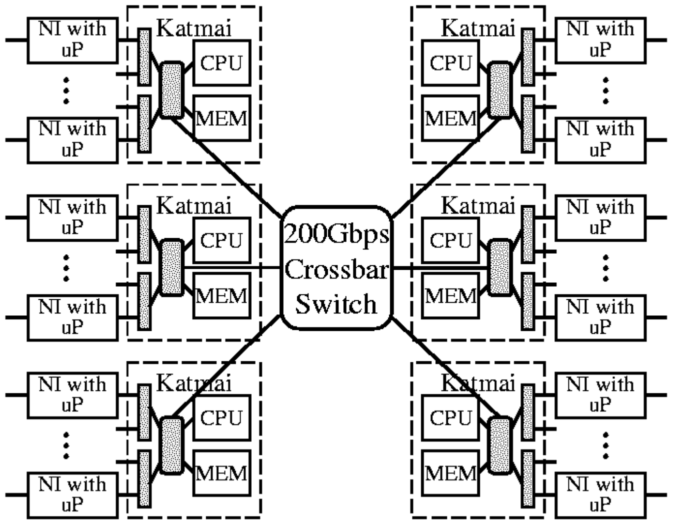
Some drawbacks are that the proof is obtained with a theorem prover, based on depth first search, which is not scalable for large programs. Also, there is no way to do bounds checking without runtime checks, but PCC may work for restricted cases. It may be possible to make use of this techniques when verifying dynamically injected code, because the verification can be done in linear time, and a restricted model may work well enough to cover many applications.

3.5 Hardware Architectures

In order to prove the viability of the active networking concept, it is necessary to achieve active routing at speeds comparable to passive routing. The problem here is to have scalability not only for the bandwidth offered by a router, but also for the processing and operating memory. Towards this goal, some projects attempted to use hardware architectures that achieve scalability using the scalability of an ATM backplane. [DP98] uses an ATM switch fabric that supports rates of 2.4Gbps on each port, and capsules are executed in ANPE (Active Networking Processing Element), which are basically line cards integrating CPU, cache, memory and ATM network access(APIC).



A capsule received on port A is forwarded to C, and then to D, and its execution will take place in the least loaded ANPE. APICs support zero copying semantics, therefore no copying is performed from the network to the execution environment.



Another project, [PKL98] at Princeton University aims at building scalable routers as clusters of tightly coupled PC systems around a 200Gbps crossbar switch. Tightly coupled means that the crossbar is connected to the main bus of each PC. A singe PC can support up to 40 ports, and each interface card is expected to have local memory and local CPU to support simple processing.

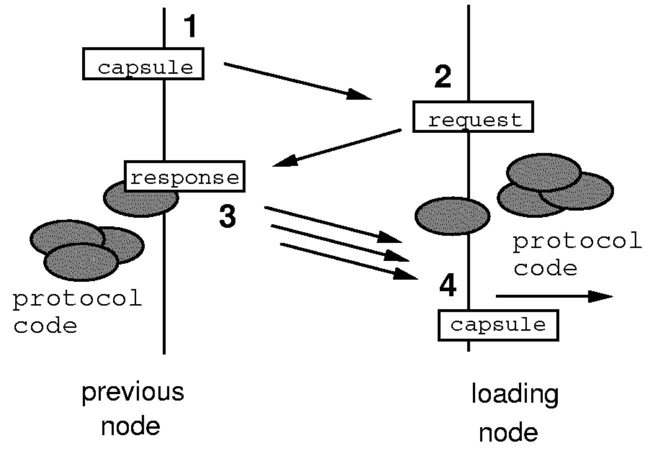
The nodes are based on off-the-shelf CPUs, thus providing a low cost for scalability. The main processors are to be used for more complex services, while basic, simpler actions can be performed faster on the line cards, which are also equipped with local CPU and memory.

4 Active Architectures

4.1 ANTS (Active Network Transport System)

ANTS [WGT98] is the MIT project that implements a model that is probably the most representative for active networking. An active router is enriched with a Java interpreter that may execute any code installed on the fly from anywhere in the network. All the packets are replaced by capsules, which are grouped into code groups, and protocols. Besides header and payload, each capsule has a field called protocol identifier, which is the message digest of the protocol code. This allows protocols to be allocated in a decentralized manner, and minimizes the possibility of interferences among protocols.

When a capsule arrives at a node, its code is immediately executed in a protected environment. A generalized TTL scheme is used to control the global resources used by the capsule. Code carried by the capsules makes calls to the API provided by the active node. The API includes access to the node environment, capsule manipulation, and access to soft state. If code pertaining to a certain protocol is not present at the node, it is loaded dynamically, using a lightweight distribution protocol:

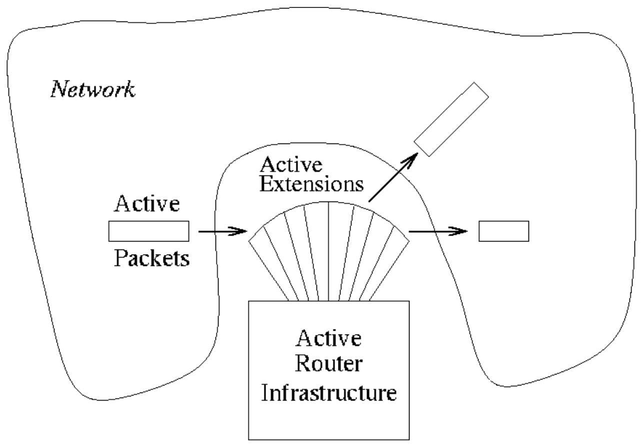


The on-demand scheme increases the startup latency of the protocol, other possibilities being to preload the code in advance or to carry the code in each capsule. On the other hand, it has the advantage of being adaptive to network failures and appropriate for short lived protocols, when the scope is not known in advance.

4.2 Switchware

The second major approach to active networking is the programmable switch, described in [Sco98]. The injection of new functionality into the active node is controlled by local node authorities. The extensions are called switchlets and are basically libraries. Users from the network are able to evaluate PLAN programs in the environment provided by the node. PLAN (Programming Language for Active Networks) is a stripped ML with the following features:

- guaranteed termination, strong typing
- no high order functions
- assignment
- no recursivity or looping constructs
- no type definitions and forward function reference
- dynamic calls to libraries installed by switchlets



Active router infrastructure is a secure environment that supports the execution of the upper layers - switchlets, and PLAN packets. Basic services such as routing, address resolution, forwarding, and even PLAN interpretation are implemented as switchlets. The entire

Switchware architecture is based on Caml, both as a development language for the extensions, and as a base for PLAN. Its main advantage is strong static typing, and compilation to bytecode.

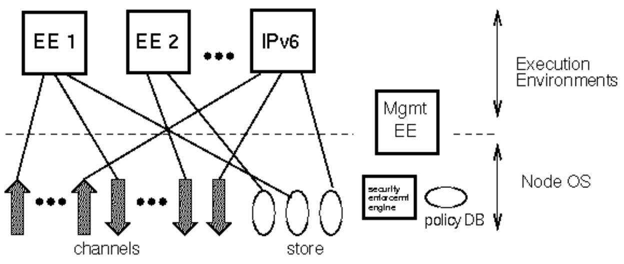
4.3 DAN (Distributed Code Caching for Active Networks)

Acknowledging problems with efficiency and security present in other approaches, [DP98] argues that active networking at gigabit speeds will not be possible in the near future based on virtual machines, interpreters and runtime verifications. Here, instead of carrying code, packets carry pointers to active modules loaded on-the-fly from trusted servers. "Code" carried by a packet is a list of functions to be called, and parameters to be passed to each function. If a node does not have the code for a function, it temporary suspends the packet and contacts a code server to get the function. Being binary specific to router architecture, the code is fast. The approach incurs some delay when deploying the protocol, but so does ANTS, only here, authentication servers are involved, which provides some security guarantees. Code is cached in a local, persistent database on each node.

This scheme will be used in conjunction with the hardware architecture afore mentioned in 3.5, but performance numbers are not published yet.

4.4 Standardization: NodeOS, ANEP, ABONE

Currently, efforts are under way to standardize the way an active router shares its resources among several active architectures and the way active packets are encapsulated for transport over the internet. Each active node runs an operating system (NodeOS)[Cal98] and one or more execution environments (EE). The functionality of the node is divided between the NodeOS and EE: NodeOS provides the basic services, sharing of the node resources and support for several concurrent EEs, while the execution environments implement an actual active architecture, such as ANTS, or Switchware. Communication is done through channels, similar to Scout paths in that they are built from protocol modules. Some channels are anchored in an execution environment, others are cut-through, used just to forward packets through the node. Packet classification is performed by ANEP (Active Network Encapsulation Protocol), which is also used for error handling instructions, security vouchers and fragmentation/reassembly. The primary abstraction for accounting is the principal. Once principals are authenticated and admitted to the node, they may allocate threads, memory or channels.



The ABONE [ABO] is an experimental network consisting of 24 nodes from all over the world used to prototype and test new ideas related to Active Networking. It allows the deployment, configuration and control of networking software (including current active networking execution environments prototypes) into the network. It demultiplexes active network packets encapsulated using ANEP to multiple EEs located on the same network node and sharing the same input port.

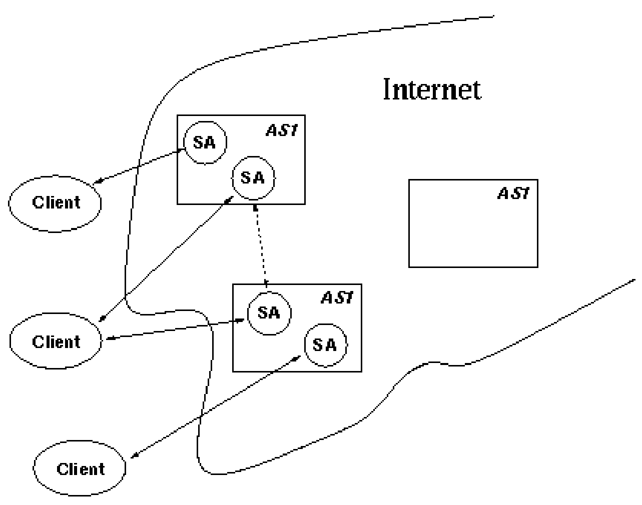
4.5 Performance Issues

Most active architectures we see today are not focused on the efficiency, but on extending the functionality. The primary function of the active network is communication, not computation, therefore one should not sacrifice functionality for efficiency. However, a number of applications, such as protocol boosters [MCS97], transcoding [AMK98], encryption, and compression will always require a high amount of CPU resources. Therefore in any context that involves mobility of code across heterogeneous environments, and a high degree of control of the CPU, efficiency will be an issue. In [DP98], Dan Decasper makes the argument that in order to route at a rate of 10Gbps, a computer running at 300Mhz has only 234 cycles to receive, process and forward a packet of 1KB, which completely excludes the possibility of using virtual machines and interpreters for high-speed active networking in the near future.

The active bridge implemented over Switchware achieves a throughput of 16Mbps, compared with the speed of 76Mbps unbridged, which is low when compared with a passive bridge. On the positive side, tests performed on a modified ANTS, working with binary linux code proved that the performance penalty is not inherent to the active architecture but is in fact due to the bytecode. It is also known that efficiency and security are many times opposing goals in the same problem. Executing each capsule in an isolated environment may be impractical performance-wise, but techniques such as PCC may provide a solution for restricted cases. The measure of performance is often an application dependent one, and this is the place where active networking would find better applicability.

5 Active Services, Composition of Services

At the opposite end of the spectrum from ANTS, and full configurability of the node, there are less radical solutions. Active services [AMK98] have the same goals as active networks, but routing and forwarding are kept passive, and the "activeness" is at the application layer. This idea pursues more realistically the practical requirements of incremental deployment of such services in today's Internet in that existing routers do not have to change, and application defined agents (called servents) run on clusters of hosts at the edge of the network instead of running on core routers. The use of clusters also addresses the problem of scalability, which is a difficult one for routers. This solution is particularly suitable for intranets and campus sized networks.



Clients may instantiate servents on clusters of hosts placed at the edge of the network. This respects the internet philosophy of keeping the complexity at the edges of the network, but gives the users the possibility to install customized servents. One example of such servents are media gateways, which perform video transcoding for clients with low bandwidth links.

The active networking technology does not simplify implementation of the new protocols, the designer still faces the same problems: packet loss, changing routes, state loss, concurrency, multiple sessions of the same protocol. One of the main goals is to simplify development and deployment of new protocols, most of them being assembled from basic blocks. Therefore a method of composition is needed inside the active architecture to provide users the possibility to remotely combine existing functionalities. NodeOS should offer standardized composable services to EE implementors, much in the way any unix offers the same API to applications [Zeg98]. Composite network services may range from a choice from a set of options (IPv4 and IPv6) to a Turing complete programming language(ANTS). While choosing from a set of options does not allow much flexibility, it can be applied to existing IP. Intermediate solutions are used by Switchware, where PLAN allows composition of services offered by switchlets, and by LIANE, an event based framework, that eases the correctness proof of the global guarantees of the network behavior. [BCZ97] describes the slot programming model, in which the active node has a fixed underlying program, and the injected code may only replace certain slots. The underlying program consists of two other programs: the common packet processing, and the default slots; injected code may replace some of the slots. The underlying program interfaces with injected code using well-known variables. A *receptive* underlying program satisfies some safety conditions with regard to the variables that are accessed. The goal is to make statements about the behavior of the network based on the fixed part of each node by constraining the injected code.

Although each protocol implemented with the active paradigm is application specific, a lot of functionality is common among protocols. [KM99] identifies classes of of active protocols - protocols belonging to the same class have similar resource needs and make use of similar active node functionality. Protocols are classified based on deployment, common interfaces, node and primitives required. The following classes have been identified:

Class	What is used for	Deployment	Primitives/Resources needed
Filtering	packet dropping, bandwidth reduction, compression, layered MPEG	near the rate mismatch	query bandwidth for all interfaces; small state management
Combining	combine packets from different streams/same stream, ATM voice cells, sensor data, caching, multicast	near the sources	storage, small state management packet duplication
Routing	virtual networks, geographical routing, information based routing.	everywhere	access to interfaces and queue sizes
Transcoding	encryption, compression, image conversion	endpoints	CPU and memory
Security Management	smart packets execute at various levels of security	everywhere	manage security associations and privileges
Network Management	network diagnose, custom snapshots	everywhere	create/access/modify the state of the active node; set up frequency and formats of reports
Supplementary Services	does not alter the packets, but may use their contents for decisions: real-time, content based buffering	everywhere	small state management

6 Summary

The paradigm of active networks combines research issues from operating systems, programming languages, and of course networking. It is part of a larger trend of software intensive approaches such as active disks and active operating systems(Spin), which aim at allowing users to customize functionality and service. The active networking approach is technologically sustained by advances in code mobility and code verification, and conceptually by the fact that network speed is not increasing as fast as the processor speed.

Introduction of new functionality into the network most of the time involves changing the routing software, which is a manual and costly process. The main goal of active networking is to make routers programmable, thus enabling the decentralized construction and use of new protocols. Allowing random users across the network to share resources in a router poses problems with efficiency, security and scalability. While it is clear that networks need to be more “active” than they are today, it is not yet clear what the best balance is between the degree of control a node should offer, and the scalability and efficiency that made internet so popular.

References

[ABO] `http://www.csl.sri.com/ancors/abone/`.

[AMK98] Elan Amir, Steven McCanne, and Randy Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. 1998. University of California at Berkeley.

[ASNS97] D. Scott Alexander, Marianne Shaw, Scott M. Nettles, and Jonathan M. Smith. Active Bridging. 1997.

[BCZ96a] Samrat Bhattacharjee, Kenneth Calvert, and Ellen W. Zegura. Network Support for Multicast Video Distribution. 1996. College ofComputing, Georgia Tech.

[BCZ96b] Samrat Bhattacharjee, Kenneth Calvert, and Ellen W. Zegura. On Active Networking and Congestion. 1996. College ofComputing, Georgia Tech.

[BCZ97] Samrat Bhattacharjee, Kenneth Calvert, and Ellen W. Zegura. Reasoning About Active Network Protocols. 1997. College ofComputing, Georgia Tech.

[BCZ98a] Samrat Bhattacharjee, Kenneth Calvert, and Ellen W. Zegura. Directions in Active Networks. 1998. College ofComputing, Georgia Tech.

[BCZ98b] Samrat Bhattacharjee, Kenneth Calvert, and Ellen W. Zegura. Self Organizing Wide-Area Caches. 1998. Georgia Rech, College of Computing.

[Cal98] Kenneth Calvert. Architectural Framework for Active Networks. August 1998. Active Networks Working Group.

[Col96] Omniware Technical Overview, 1996. Colusa Software White Paper.

[DP98] Dan Decasper and Bernhardt Plattner. DAN: Distributed Code Caching for Active Networks. In *INFOCOM*, April 1998.

[HPB$^+$97] John H. Hartman, Larry L. Peterson, Andy Bavier, Peter A. Bigot, Patrick Bridges, Brandy Montz, Rob Piltz, Todd A. Proebsting, and Oliver Spatscheck. Joust: A Platform for Communication Oriented Liquid Software. Technical report, University of Arizona, December 1997. TR 97-16.

[Joh98] Edwin N. Johnson. A Protocol for Network Level Caching, May 1998.

[KM99] Amit B. Kulkarni and Gary B. Minden. Active Networking Services for Wired/Wireless Networks. In *INFOCOM*, San Francisco, CA, April 1999.

[LG98] Ulana Legedza and John Guttag. Using Network-level Support to Improve Cache Routing. In *International WWW Caching Workshop*, Manchester, England, June 1998.

[LWG98] Ulana Legedza, David Wetherall, and John Guttag. Improving the Performance of Distributed Applications Using Active Networks. In *INFOCOM*, San Francisco, CA, April 1998.

[MCS97] A. Mallet, J. D. Chung, and J. M. Smith. Operating System Support for Protocol Boosters. Technical report, Distributed Systems Laboratory, University of Pennsylvania, 1997.

[MP96] David Mosberger and Larry L. Peterson. Making Paths Explicit in the Scout Operating System. 1996. University of Arizona.

[Nec97] George C. Necula. Proof-Carrying Code. In *POPL*, January 1997.

[PKL98] Larry Peterson, Scott Karlin, and Kai Li. OS Support for General Purpose Routers. Technical report, Princeton University, 09 1998.

[Sco98] Scott D. Alexander and William A. Arbaugh and Michael W. Hicks and Pankaj Kakkar and Angelos D. Keromytis and Jonathan T. Moore and Carl A. Gunther and Scott M. Nettles and Jonathan M. Smith. The Switchware Active Network Architecture. Technical report, University of Pennsylvania, July 1998.

[SZJ$^+$98] Beverly Schwartz, Wenyi Zhou, Alden Jackson, W. Timothy Strayer, Dennis Rockwell, and Craig Partridge. Smart Packets for Active Networks. 1998. BBN Technologies.

[TGSK96] D.L. Tennenhouse, S.J. Garland, L. Shrira, and M.F. Kaasoek. From Internet to ActiveNet. RFC, January 1996. Available at `http://www.tns.lcs.mit.edu`.

[TH98] Joe Touch and Steve Hotz. The X-BONE. 1998. USC/Information Sciences Institute.

[Van97] Van C. Van. A Defense Against Address Spoofing Using Active Networks, May 1997. MIT Master Thesis.

[WGT98] David J. Wetherall, John V. Guttag, and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE OPENARCH*, San Francisco, CA, April 1998.

[wHLGT98] Li wei H. Lehman, Stephen J. Garland, and David L. Tennenhouse. Active Reliable Multicast. In *Proceedings of the INFOCOM*, pages 77–87. IEEE, 1998.

[WLG98] David Wetherall, Ulana Legedza, and John Guttag. Introducing New Internet Services: Why and How. May 1998. IEEE Network Magazine.

[WT96] David J. Wetherall and David L. Tennenhouse. Towards an ACtive Network Architecture. In *Conference on Multimedia Computing and Networking*, San Jose, CA, January 1996.

[Yec96] Yechiam Yemini and Sushil da Silva. Towards Programmable Networks. Technical report, Columbia University, April 1996.

[Zeg98] Ellen Zegura. Composable Services for Active Networks. May 1998. AN Composable Services Working Group.