Enabling Active Flow Manipulation in Silicon-based Network Forwarding Engines[1]

Tal Lavian[2], Phil Wang, Franco Travostino, Siva Subramanian[3] and Ramesh Duraraj
{tlavian, pywang, travos, ssiva, radurai}@nortelnetworks.com
Advanced Technology, Nortel Networks Corp.

Doan Hoang[4]
dhoang@it.uts.edu.au
Department of Computer Systems, University of Technology, Sydney, Australia

Vijak Sethaput[5]
vijak@eecs.harvard.edu
Division of Engineering and Applied Sciences, Harvard University

David Culler
culler@eecs.berkeley.edu
Computer Science Division, University of California at Berkeley

Abstract

A significant challenge arising from today's increasing Internet traffic is the ability to flexibly incorporate intelligent control in high performance commercial network devices. This paper tackles this challenge by introducing the Active Flow Manipulation (AFM) mechanism to enhance traffic control intelligence of network devices through programmability. With AFM, customer network services can exercise active network control by identifying distinctive flows and applying specified actions to alter network behavior in real-time. These services are dynamically loaded through Openet by the CPU-based control unit of a network node and are closely coupled with its silicon-based forwarding engines, without negatively impacting forwarding performance. AFM is exposed as a key enabling technology of the programmable networking platform Openet. The effectiveness of our approach is demonstrated by four active network services on commercial network nodes.

Keywords

Programmable networking, intelligent network control, Active Flow Manipulation, Openet, active networks, traffic flow

I. INTRODUCTION

The Internet infrastructure has been tremendously evolving to transport increasing network traffic arising from fast introduction of various commercial applications. The Internet is ubiquitous, however,

[1] This article is revised from an earlier version published in Journal of Communications and Networks, March 2001, under its copyright and reprint permission granted.
[2] Tal Lavian is a Ph.D. candidate at the Computer Science Division, University of California, Berkeley.
[3] Siva Subramanian is a Ph.D. candidate in the Electrical and Computer Engineering Dept. at North Carolina State University.
[4] Doan Hoang was a visiting research professor at Advanced Technology, Nortel Networks Corp in 2001.
[5] Vijak Sethaput was an interim employee in Nortel Networks Corp. in 2001.

fragmented in structure into large heterogeneous network domains controlled by Internet Service Providers (ISPs). The providers have to rely on a complex collection of operational management methodologies and techniques in order to operate their networks. In this increasingly competitive environment, it is important for service providers to easily control their networks. It is also important for them to allow customization of network services to differentiate their offerings by rapidly introducing intelligent network services on demand such as QoS (Quality of Service) to their clients. In brief, they are in need of a comprehensive programmable networking framework through which they manage their networks intelligently to satisfy the clients' needs.

The fundamental element of the Internet infrastructure is the network node, e.g., a router or switch. Typically, the distinction of the data (or forwarding) and control planes is drawn at each node with the hardware realizing the forwarding operations and the software realizing the control operations. Nowadays, the trend in commercial grade routers and switches is to accelerate performance critical forwarding functionality using hardware technologies such as ASIC (Application-Specific Integrated Circuit) technology. As a result they provide little programmability and thus are limited in ability to deliver intelligent control. However, dynamically enabling and deploying new intelligent services on network nodes implies that they must possess not only high forwarding performance, but also high degree of programmability.

It is a challenging task to come up with an enabling technology that allows network service providers freedom to deploy intelligent services on current commercial network devices. One of the requirements for such an enabling technology is that it must have little or no adverse impact on the processing performance in the data path. Another important requirement of such technology is that it should be distributed rather than centralized in the network.

To tackle the challenging issues discussed above, in this paper the emphasis is placed on the Active Flow Manipulation (AFM) mechanism, which aims to affect the data traffic in real networks. AFM is a key enabling technology of the open programmable networking architecture Openet. The AFM proposition is that the characteristics of a basic data flow can be identified and its behaviors can be altered in real-time by customizing control-plane services. Openet is a platform-neutral, service-based internetworking infrastructure developed by Nortel Networks Corp., aiming to deliver dynamic network programmability on heterogeneous network devices. Commercial network devices such as the Nortel Networks multi-gigabit routing switch Passport [4], possess the ability to alter traffic flow behaviors in the silicon-based forwarding plane. The control plane in such a device, however, lacks user-programmability required to introduce intelligent services. Openet provides such programmability to enable users with control of the forwarding hardware. This programmability is in this paper manifested as the ability to alter network behavior of flows in real-time, i.e., AFM, in order to enhance the functionality of network devices.

This paper introduces the concept of Active Flow Manipulation for identifying and affecting traffic flows of interest in silicon-based high-speed network nodes. It also introduces the Openet open programmable platform and its mechanisms that can dynamically enable programmed services in the control plane. Finally it demonstrates the use of AFM mechanism with the Openet infrastructure through several experimental applications.

The remainder of the paper is organized as follows. Section 2 introduces the mechanism of Active Flow Manipulation to enable actual network control in real-time. Section 3 introduces the Openet architecture, and its mechanisms for dynamic service deployment. Section 4 presents four AFM applications that are developed through the Openet programmability. Section 5 gives a brief review of related work. Finally, the paper concludes with future work.

II. ACTIVE FLOW MANIPULATION

In hardware, an Internet router or switch typically consists of a control plane and a forwarding plane. The forwarding plane has a set of networking forwarding engines and is responsible for per-packet activities such as classifying, queuing and forwarding. The control plane is usually a CPU-based system unit and responsible for control functions such as routing, signaling, admission control and other mechanisms altering the behavior or data of selective traffic on the forwarding plane. Control functions can be realized in three ways: 1) executing wholly in the control plane (e.g., connection management), 2) inserting additional software in the data path, 3) allowing control entities to act both in the control plane and in the forwarding plane without adding software in the data path. The first control incurs significant forwarding performance penalty when data processing is involved, the second suffers in functionality because of the little ability to add software in the data path. Thus, this paper focuses on the third control, especially the simple type of dynamic control that affects a vast amount of data transporting through a network node in real-time.

Consider the situation where a massive amount of data traffic has to be switched through a network node, and it is desirable and essential to exercise selective controls over selective traffic. To avoid a switching performance penalty, the controls have to be done in real-time. The Active Flow Manipulation (AFM) mechanism is introduced to solve this by having the control functions differentiate traffic flows, not individual packets.

Current commercial-grade nodes are provided with high-performance forwarding engines, for example, a typical routing switch such as Passport [4] can forward packets at a total throughput of 256 Gbps. With such a high rate, there is no simple way to inspect packet by packet in the control plane and then figure out appropriate actions. Fortunately, the hardware in the forwarding plane can perform several tasks extremely well. For example, it can differentiate packets based on packet headers and selective payload portions, and perform some simple actions such as filtering, diverting, dropping and forwarding selected packets.

The AFM mechanism involves two abstraction levels in the control plane. One is the level at which a node can aggregate its data into traffic flows, and the other is the level at which it can perform actions on the traffic flows. It is futile in the control plane to think of packets individually; instead, it is more appropriate and productive to think and act in terms of primitive flows whose characteristics can be identified and whose behaviors can be altered by primitive actions in real-time. For example, one wants to exercise some controls over "all TCP traffic to an HTTP service at a web server", "all RTP/UDP datagram generated from several identifiable video stations to a particular display machine", or "all traffic passing through a physical port of a router". It is at this level of abstraction that active control of data on the forwarding plane can be performed without violating its real-time constraints.

To formalize this framework, let the *primitive flow* set be a set of atomic elements that can be matched and identified by hardware in real-time, see Table 1. The set is based on the general hardware capability (e.g., the Passport), but can be expanded with more sophisticated next-generation hardware.

Table 1: The *primitive flow* set of identifiable elements

Destination Address (DA)
Source Address (SA)
Exact TCP protocol match (TCP)
Exact UDP protocol match (UDP)
Exact ICMP protocol match (ICMP)
Source Port number, for TCP and UDP (SP)
Destination Port number, for TCP and UDP (DP)
TCP connection request (TCPReq)
ICMP request (ICMPReq)
DS field of IP datagram (DS)
IP Frame fragment (FrameFrag)

The primitive flow is the first abstraction level of AFM necessary to deal with the various attributes of data traffic. Essential properties of a flow are identifiable atomic elements and can be acted upon in real-time. A simple TCP flow can be identified by 5-tuple (TCP, SA, SP, DA, DP), for which all packets in this flow have to match. If any of the 5 elements is relaxed, the flow becomes a less restrictive flow. For example, a flow of all TCP traffic destined to a particular service on a particular machine is identified by 3 elements, the protocol (TCP), the destination address and the destination port (i.e., TCP, *, *, DA, DP).

In the most general sense, a set of operators (and, or, not and range) can be defined on this primitive set of elements to obtain new composite elements. One is interested in these composite elements that define composite flows. The scope of flows that can be generated from the base elements of the *primitive flow* set is much more general than just simple TCP flows. In principle, one can "operate" on elements of the *primitive flow* set to construct a particular composite flow of interest, for example, a "all premium-grade traffic to a particular destination machine" flow using the DS field and the Destination Address elements of the packet header.

Ideally, a proper set of *primitive flow elements* and a proper set of operators form an algebra, in which any operation on a flow results in another valid composite flow. However, hardware is not completely designed for this algebra. Realistically, some of the operations on the *primitive flow* set, or some composite flows, may not be practical and hence can be eliminated. It should be noted that all elements of the primitive flow set are associated with some control actions that can be realized in real-time. It is possible to enumerate all possible combinations of the *primitive flow* set and to identify all realizable flows. However, due to the space limitation of the paper, only a subset of realizable flows is shown in Table 2. The whole set of realizable flows presents a network controller with a powerful set of targets to develop applications.

Table 2: A subset of realizable flows to one destination

Sources	Destination Address (DA)
Any	All packets to the destination
Source Address (SA)	All packets from the SA machine to the destination
Range of SAs	All packets from many source machines to the destination
TCP	All TCP packets to the destination
UDP	All UDP datagrams to the destination
ICMP	All ICMP messages to the destination
ICMP Request	All ICMP request messages to the destination
TCP ACK	All TCP acknowledgement packets to the destination
TCP RST	All TCP packets with the RST bit set to the destination
DP (TCP/UDP)	All TCP packets or UDP datagrams to a particular service in the destination
SA-SP (TCP/UDP)	All TCP packets or UDP datagrams sourcing from a particular client of a source machine to the destination
IP Fragments	All IP fragments to the destination
DS Field	All packets of a particular DSCP value to the destination
VLAN	All packets from a particular VLAN number to the destination
Switch-Port	All packets through a particular interface port to the destination

The second abstraction level is the *primitive actions*, which constitute a base set of actions that can be associated with a real-time identifiable flow to realize an active flow manipulation. One can also operate on this base set to obtain composite actions. A subset of actions of interest is shown in Table 3. Combining the set of composite and realizable flows with the set of composite actions generates a set of desirable active flow manipulations. Examples are “increase the forwarding priority of all TCP traffic to an HTTP service at a web server” and “drop all traffic through a physical port of a router (to a broken link)”.

Table 3: The *primitive action* set of permissible actions

Action
Drop
Forward
Divert
Mirror
Stop on Match
Out-of-Profile behaviour
Change DSCP bits
Prevent TCP Connection Request
Change IEEE 802.1p bit

More importantly, on a commercial node like the Passport routing switch, a composite flow is readily realized by a list of hardware filters applied to a particular port of this node. A composite action to be performed also in hardware on the composite flow is the combination of all the actions of individual filters applied.

By definition, AFM requires the ability to change the flow-action combinations in real-time. A programmable network platform is all that is needed for AFM to operate. Openet is such a platform that can dynamically inject smart control services and house them in the control plane. It allows the control mechanisms to couple intimately with the hardware to perform actions in real-time. Such a service generally requires simple computation in the control plane to set various policies supported by the switch. Typical services belonging to this category include filtering firewall, dynamic flow identification, classifying and marking, remarking flows, altering priority of a flow, intercepting special control messages for further processing. However, it should be again emphasized that our focus is on the type of control that preserves the forwarding plane by avoiding introduction of software on the data path.

III. OPENET AND SERVICE DEPLOYMENT

Openet originated from the open programmable architecture for Java-enabled network devices [1] has been evolving with later work [2,5,6]. It is platform-neutral, and works closely with commercial nodes such as the Passport routing switch [4] and Alteon web switch [24] to provide the flexible networking programmability.

A. Openet

Figure 1 depicts the Openet architecture in a distributed network that comprises routers and switches, end hosts, repository servers and control consoles. The routers and switches download service codes and policies from the repository servers and run network services locally, as demanded by end applications and control consoles. Repository servers run downloading services (e.g., HTTP or FTP) and store network-related resources such as service codes, network configurations and policies. Control consoles perform manually or programmatically the management tasks such as service initiations on routers and switches and storage maintenances on the repository servers.

Openet provides network service providers as well as end users the programmable networking ability through four major components: the runtime environment (*ORE*), hierarchical network services (*Oplets*), the Oplet development kit (*ODK*) for service creation, and the management part (Openet *managers* and *agents*).

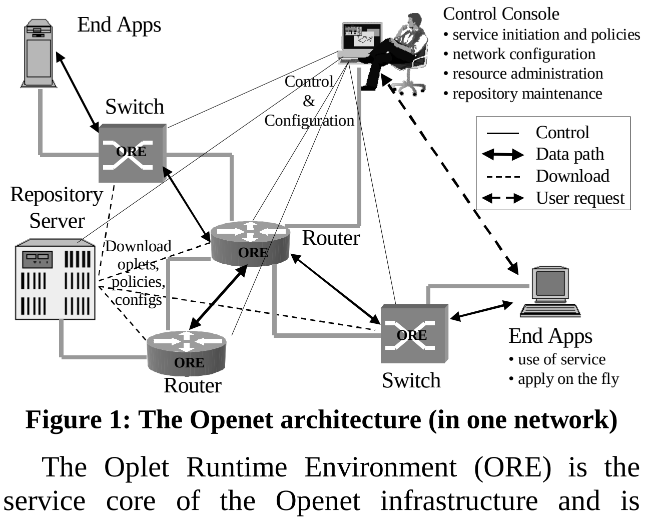


Figure 1: The Openet architecture (in one network)

The Oplet Runtime Environment (ORE) is the service core of the Openet infrastructure and is distributed on network nodes. It is an open object-oriented networking environment for customer service creation and deployment. At runtime, it is dynamically installed on network nodes and supports injecting customized software of network services to the nodes through secure downloading, installation, and safe execution of Java-based service code inside a JVM.

The Openet management part, consisting of the Openet managers running on control consoles and the Openet agents on routers, switches and repository servers, conducts service management, resource administration, repository maintenance, and network configuration.

B. Passport Routing Switch

The Passport[6] achieves a significantly higher level of performance by employing two separated working planes *control* and *forwarding*, as depicted in Figure 2. The forwarding plane has multiple ASIC-based forwarding engines that can forward packets at the wire speed and reach a total throughput of 256 Gbps (gigabits per seconds) without consuming any CPU resource.

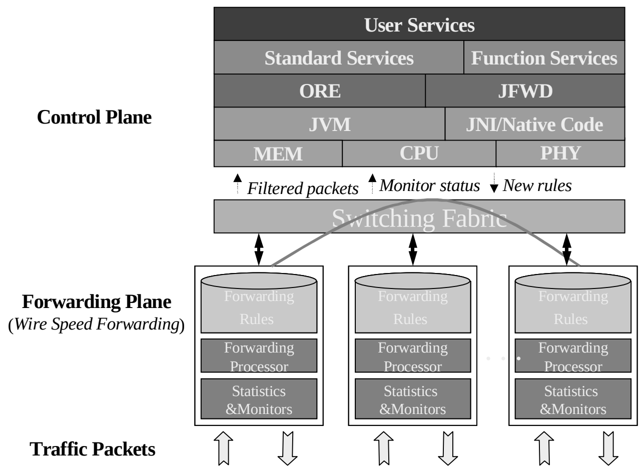


Figure 2: Passport with Openet (a node's view)

The control plane, however, is based on a CPU blade and contains the embedded Java VM. It executes ORE and that in turn enables execution of diversified Oplet services. Thus, Openet introduces programmability into the otherwise rigid routing

[6] Passport 1100B, 8600 and other models are available in commercial market without Openet included. Openet is open source and free available for research purpose.

switch Passport and makes it capable of supporting AFM and other mechanisms that customized network services provide.

C. Service Deployment

Network services are composed of normal Java objects, and encapsulated as Oplets. The *Oplet* is a self-contained downloadable unit that embodies a non-empty set of services in order to secure service downloading and management. Along with the service code, an Oplet relates service attributes, authentication, and resource requirements. Furthermore, it publishes the service and its public APIs to application services.

On a network node like the Passport, ORE and network services are initiated at the control plane, but can operate with either or both of the two planes: *control* and *forwarding (or data)*. Control-plane services change network configurations (e.g., routes) and affect the data forwarding behaviors by altering the hardware instrumentation, while data-plane services cut through the data path and seize and process particular packets prior to forwarding.

To ease service creation and gain platform independency, Openet employs a service hierarchy that places network services into four categories: *System*, *Standard*, *Function* and *User*. First, "System services" are low-level network services that have direct access to the hardware features, e.g., JFWD that provides neutral Java APIs used by AFM and other mechanism services to alter the hardware routing and forwarding behaviors. They require particular hardware knowledge and are implemented using native programming interfaces or the hardware instrumentation. Thus, in fact, they by their neutral APIs determine how much of the programmability Openet brings to hardware. Second, "Standard Services" provide the ORE fundamental features for customer service creation and deployment, e.g., "OpletService" is a base class of service creation. They make up the ODK that is used at service development. Third, "Function Services" provide common functionality or utility used to rapidly create user-level services, and are usually intermediate services coming with the ORE release or contributed by the third party. Finally, "User Services" are the customers' application services for particular purposes.

The final step of service deployment in real networks is to inject network services, which requires downloading and activating the service code within the ORE on commercial network nodes. There are at least three ways to do dynamic service injection, using the ORE shell service, the ORE startup service or a user service initiation service. During runtime one can instruct ORE to download and then activate these services, which are thus deployed on network nodes and run locally.

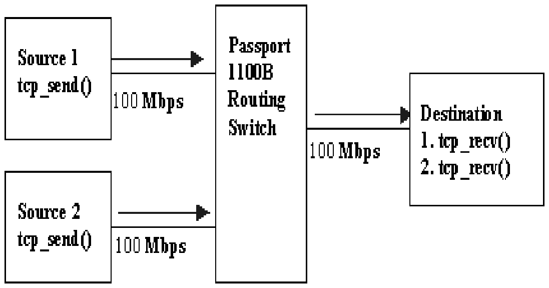
IV. AFM-ENABLED APPLICATIONS

With Openet, AFM-enabled applications are actually those network services developed using the AFM mechanisms for particular application purposes. In this section, four AFM services in both control plane and forwarding plane are described and their experimental results are measured with the Passport routing switch and other network systems.

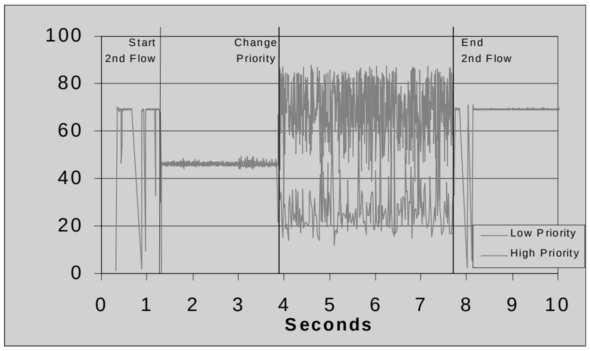
A. Active Flow Priority Change in Real-time

The active flow priority change is a control-plane network service that applies AFM to alter the packet forwarding priorities of particular flows in real-time. It is a simple and effective application showing how a customized service controls the hardware behaviors. The experiment network depicted in Figure 3a is established with the Passport 1100B routing switch, and three hosts that are Linux-based PC systems.

The experiment procedure is as follows (see Figure 3b). At the beginning, the first TCP flow at a constant rate of 100Mbps is set up from Source 1 to the Destination through the Passport. The link bandwidth between the Passport and the Destination is 100Mbps at maximum. At time 1.3 seconds, the second TCP flow at the same rate from Source 2 is set up through the same link to the Destination. When they become stable, each claims nearly half of the link bandwidth (47Mbps). Then, the ORE on the Passport is instructed to activate the "active priority" service, which employs AFM to detect particular flows and increases the packet priority of the second flow at time 3.8 seconds. As expected, the receiving rate of the second flow (now with a high priority) increases and stabilizes at the desired bandwidth (70Mpbs) and the low-priority first one at a lower rate (24Mbps).



(a) Network layout



(b) Flow throughputs at the Destination

Figure 3: Two TCP flows competing for the link bandwidth with active priority change

The bandwidth jumping of the second flow at time 3.8 seconds shows that the Passport forwarding plane carries out the AFM identification and action of packet flows at the wire-speed, without obvious performance reduction. The reason is that because the control service does not require packet processing in the data plane. The forwarding engine of the network node processes and forwards packets while the CPU executes the Java control code implementing the AFM-based service.

Even though the experiment and its result are not groundbreaking, this application service indicates an immediate benefit of active detection of flows and dynamic adjustment of packet priorities on commercial-grade nodes. It can be used widely in traffic control such as end-to-end video and audio traffic, and QoS mechanisms such as Intserv and Diffserv.

B. Active IP accounting

In traditional IP accounting, network nodes (e.g., routers, switches and firewall gateways) collect data regarding the network traffic that flows through them, and then upload the data periodically into centralized accounting servers. The servers synthesize the unwashed accounting data off-line, and make the outcome available to accounting applications such as billing and load auditing. As the Internet becomes ubiquitous, traditional IP accounting is facing a number of new challenges such as "pay for what you use" custom pricing schema, accounting data volumes linearly growing with the bandwidth, and real-time QoS monitoring.

The Active IP Accounting Co-processor Environment (AIACE) [3] revises traditional IP accounting at the very foundation, and is a control-plane service infrastructure. Based on the AFM mechanism, the AIACE infrastructure argues that the number of accounting tasks performed at both network nodes and accounting servers must be fluid and not necessarily known a priori. That is, network nodes cease being accounting-illiterate to the contrary, effectively pre-process flows' accounting data at an extent that the recipient accounting servers can control.

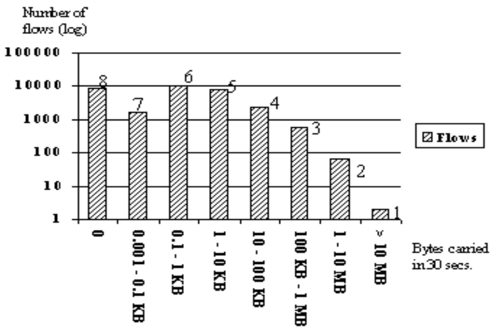
In this model, accounting plug-ins are the elemental processing units, and stacked into the AIACE co-processor and perform specific accounting tasks at the network nodes on behalf of accounting servers. Thus, the new accounting-savvy network nodes can eagerly do a number of active tasks such as aggregating the accounting data of flows meeting pre-set affinity criteria, reflecting settlements among providers, enabling real-time accounting data mining, and signaling accounting servers to meet accounting applications' needs.

The merits of AIACE are shown in the following sample scenario for high-confidence flow monitoring. An application periodically sweeps a network topology and reports flow vitals to the operator (e.g., the cumulative traffic figures concerning flows with the most remunerative SLAs). The high-confidence attribute implies that such an application is dependable in reporting traffic figures in real-time, in spite of overloads (e.g., CPU, or accounting data overloads) possibly induced by partial failures in the network. In other words, this type of traffic monitoring application must be especially well behaved when things in the network start to go wrong.

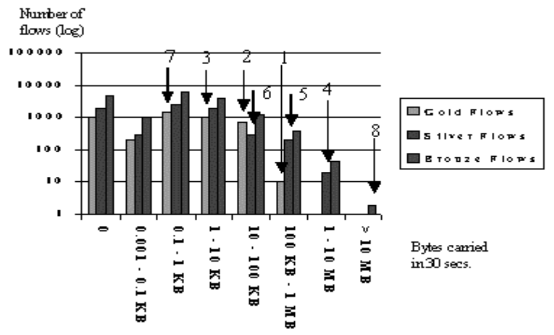
In this scenario, it is crucial to manage the finite monitoring capacity and to make the most effective accounting data mining out of it. A whole sweep of the network topology represents a cycle; cycles are typically configured to complete in a few seconds. At the end of each cycle, the breakdown of the monitoring capacity is revisited to adapt to conditions occurred in the previous sweep, or to accommodate an operator’s explicit request to zoom-in on “hot” sectors of the sweep. In principle, at each cycle the monitoring application communicates to AIACE network nodes with the below steps.

1) How much accounting data the application wants to handle from a given network node;
2) How the network node should weight its accounting data for flows, decide what to mine out, and package it within the aforementioned limit;
3) Which accuracy is expected from the network node while performing this accounting data mining.

The opportunities in Step 2 become evident in the examples showed in Figure 4. In example a), a network-node organizes about a million PDU traces into 30,000 IP flows. It classifies the resulting flows based on the bytes transferred on each flow. It then ranks flows (from 1 to 8). The higher the rank number, the higher the chance that the flow will not be transferred to the accounting server in case of data overload.



a) Flow ranks by bytes transferred on each flow



b) Flow ranks with a QoS-specific weighting algorithm

Figure 4: Results of a flow monitoring scenario

In example b), the network under analysis is QoS-enabled and three QoS classes—gold, silver, and bronze—are defined. The node now structures the same accounting data into QoS-flavored flows (same X and Y axes as in a). After applying a QoS-specific weighting algorithm to the flows. The node ranks flows with different results than a). The weighting algorithm can be arbitrarily complex and take into account other considerations besides bytes transferred (e.g., hosts, number of packets, duration).

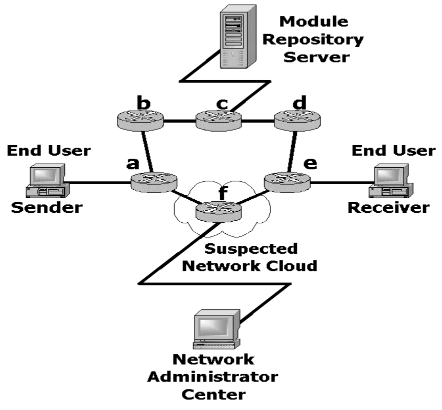
By specifying a weighting algorithm for accounting data in the various QoS classes, the application passes tidbits of its business model to the network node—i.e., it says what the most significant accounting data are and how much this matters. This node thus weights the accounting data that best reflect this business model. Should the accounting data exceed the size pre-set by the application (i.e., overload), this node will throttle itself by pruning the least significant accounting data from its reports.

AIACE’s accounting plug-ins realize Steps 1 through 3 by operating at both network nodes and accounting server. Some plug-ins that define the weighting algorithms are loaded and executed only at network nodes. Other plug-ins that implement the accounting wire protocol and its capability to drive the nodes’ accounting data output are reciprocated at network nodes and accounting servers.

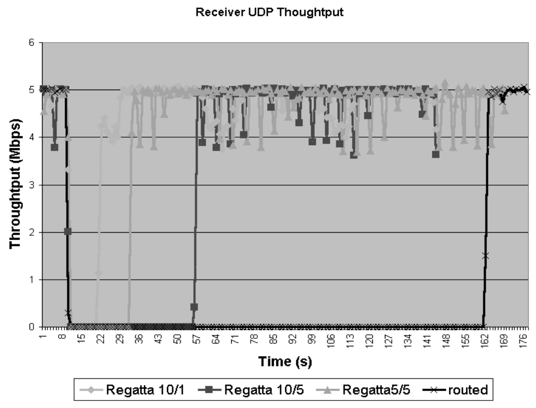
C. Dynamic bypassing flows for automated supervision

Regatta is another control-plane service that employs the Openet infrastructure and AFM for automated supervision [23]. Regatta stops, in a dynamic fashion, flows through routers when a node operation fails and leaves them to the Regatta (routing) supervision procedure. The Regatta supervision procedure handles the bypass with minimal service interruption to the user. Consider the example network (Figure 5a) of 6 nodes constituting two disjoint network paths between the end systems. Node f, for instance a beta-level prototype, is known not to work reliably. It has a failure semantics that can be described as "the link layer is always up, but the IP layer sometimes suddenly fails to route PDUs".

The network operator can thus aim Regatta at node *f*, with two goals that a) node *f* should be bypassed as soon as Regatta detects that it has a failure, without any user interruption; and b) Regatta notifies the operator who then starts post-mortem analysis of node *f* before it gets rebooted. Here we present a quantitative measure of a), and contrast it with the self-healing properties that have been already built into the network in terms of standard routing protocols.



(a) Experimental setup



(b) Varying degree of disruption at the end-user during the failure to node "f"

Figure 5: Dynamic flow bypass using Regatta

Traffic flows between the two end users go through nodes *a*, *f*, and *e* which is the shortest path. After the network operator installed Regatta at node *f*, Regatta begins to unfold itself outside of node *f*. In particular, it installs itself in the two adjacent nodes: *a* and *e*. Node *f* does not play any active role of control except in this bootstrap operation. It becomes the subject of the attentions by other reliable neighboring nodes. The Regatta-activated nodes *a* and *e* exchange periodic heartbeats between them to supervise the well being of suspected node *f*. The type and rate of the heartbeats are defined by the operator.

Upon a defined number of heartbeat missed, the Regatta diagnosis module in node *a* determines that there is a failure of suspected node *f*. At that time, the repair module on node *a* gets control and, "clamps" the flow path(s) that goes through node *f*. Through AFM and other Openet routing service, this repair module establishes a new route that reaches the end receiver via node *b* in place of node *f*. Then, node *a* reactivates those flows destined to node *e* through this route. On node *e*, this "clamping" procedure similarly gets reciprocated upon the heartbeat missed, without any support for synchronicity between nodes *a* and *e*.

Table 4 shows a comparison of reactivity times applying such a network bypass in our experiment, of which all these nodes are Linux PCs installed with the Linux Openet/ORE system. The first two rows "static route" and "routed" are two flow re-activities without Regatta while others are with Regatta. The last entry "Regatta M/HB" means a heartbeat interval of HB seconds and a tolerance of M consecutive heartbeat missed before calling for a failure.

Table 4: Comparison of reactivity times

Flow Path	Reactivity Time (s)
Static route	Infinite
Routed	152
Regatta 10/1	10
Regatta 10/5	47
Regatta 5/5	24
Regatta M/HB	≈M*HB

Figure 5b plots the throughputs at the end receiver with varying levels of disruption. With the bypass being applied in real-time upon detection of a failure, there is minimal effect to end-user traffic. The network operator can balance the trade-off between the reactivity time and the heartbeat overhead using different parameters (i.e., M and HB).

D. Active Networks Service in Real Networks

It is obviously of interest that Openet enables commercial network devices such as the Passport rather than host-based systems such as Linux routers to host Active Networks services or execution environments (EEs) and thus to embody the Active Networks approach in real networks. Of Active Networks EEs, MIT ANTS [10] is a typical mechanism for dynamically composing and installing new transport protocols in networks, and actually it is a data-plane service that processes active packets or capsules directly. Through Openet, the ORE ANTS service [6] is developed wrapping the MIT ANTS. It also uses AFM to collect specified capsules for CPU processing in the control plane.

The ORE ANTS is completely injected to two Passport routing switches: 1100B and 8600 [5]. Of the Passport routing switch family, 1100B is an economic product that is equipped with a PowerPC 403@67MHz CPU and 8600 is a superior one that is equipped with a much stronger PowerPC 740@266MHz CPU.

This service is tested with the ANTS applications such as the ANTS Ping (APing) and compared with regular Linux Ping. Within the corporate intranet, we construct an experimental active net that includes 3 active nodes and 3 non-active ones, shown in Figure 6. The Passport 1100B or 8600 routing switch, and 3 PII/400MHz PC boxes are located in an experiment network (*net 10*), which is routed to the intranet where working machines such as Sun workstations are.

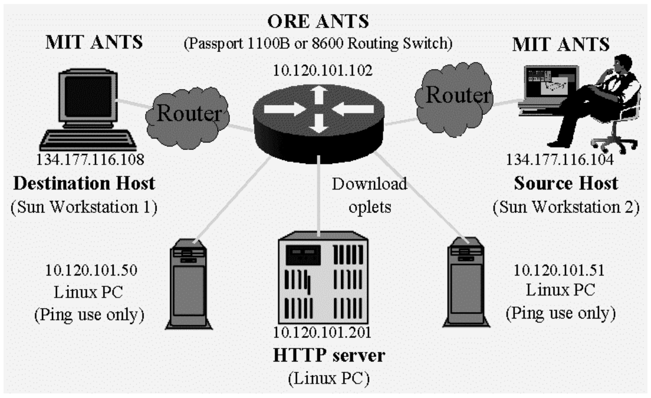


Figure 6: The experimental active net running ANTS EEs

Table 5 lists the packet average delays and throughput rates of the APing and Ping tests, both of which send at given intervals 83-byte capsules or 64-byte packets from the Source to the Destination through the Passport. The first two Aping tests with Passport 1100B have capsules lost and do not have their average delays and throughput rates. The maximal throughput of the APing tests is found to be 32.3 cps (capsule per second) for Passport 1100B and 90.9 cps for Passport 8600. However the two Ping tests have the same throughput of 10,000 pps (packet per second) and the same average delay of 0.1ms as well. This comparison reveals that capsule-by-capsule CPU processing in the ANTS service cannot match the performance level of regular Ping packet processing which is actually accelerated by hardware.

To understand what contributes the capsule delay, capsule-processing time consumed at each node is measured by comparing the time of receiving and re-transmitting one capsule. Figure 7 depicts the delay distributions in the two extreme Aping tests that have minimal average delays: 31ms for Passport 1100B and 11ms for Passport 8600.

Table 5: Delays and Throughputs of Ping and Aping Tests (time in milliseconds)

Interval	Ping		
	First packet	Average	Throughput (pps)
0	1.2	0.1	10000
1000	0.8	0.1	10000
Interval	APing (1100B)		
	First capsule	Average	Throughput (cps)
0	3209	-	(capsules lost)
10	551	-	(capsules lost)
100	139	32	31.5
1000	131	31	32.3
Interval	APing (8600)		
	First capsule	Average	Throughput (cps)
0	47	391	2.55
10	12	11	90.9
100	12	11	90.9
1000	13	11	90.9

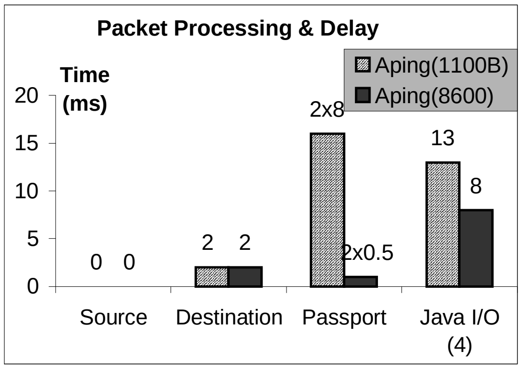


Figure 7: Delay distributions (Transmitting a capsule involves one time processing at Source and Destination, two times processing at Passport and four times Java I/O on all three nodes)

It is found that the round-trip network communication of a capsule takes 13ms for Passport 1100B and 8ms for Passport 8600. At the link speed, it takes very little time (less than 0.1 ms) for transferring a capsule among three nodes. Hence, a large portion of the round trip time must have been taken up by 4 pairs of Java network I/O operations (reading and writing a capsule) on the three nodes (1 on the Source, 1 on the Destination and 2 on the Passport). This is not really expected! However, additional network tests based on a Linux PC and a Sun workstation confirm that the Java overhead for a pair of simple UDP socket I/O operations (i.e., a DatagramSocket server writes a 32-byte message and a DatagramSocket client reads it) needs 2~3 ms while the same socket operations using C/C++ takes nearly 0 ms.

The lesson to be learned here is that JVM implementation, particularly in Java network I/O operations, is the real bottleneck of active networking performance besides the capsule processing ability on network nodes. Of course, adding faster CPUs to the control plane of commercial hardware is highly preferable in order to reach the high processing performance that the forwarding plane does already. However, it might be a reasonable consideration to re-engineer ANTS and other active network services with improved AFM mechanisms so that they can make better use of both forwarding engines and CPU on commercial network nodes.

V. RELATED WORK

A significant amount of research has been involved with enabling intelligent network control through programmable networking, ranging from networking paradigms, re-programmable hardware to application environments.

Industrial organizations such as P1520/PIN (Programming Interfaces for Networks) [16], CPIX (Common Programming Interface) [17] and Parlay [18] are working on standardization of programmable networking interfaces among hardware, network services and user applications. These standard interfaces are open, generic and have been released in their early drafts. With these standards, Openet can effectively define the boundaries of network services, and normalize service development and deployment.

The Active Networks (AN) approach [8] is a major effort in industry as well as academia to incorporate programmability into the network infrastructure. Through installing multiple active user interfaces or typical Execution Environments (EEs) on active nodes, users can flexibly compose new protocols and deploy their services for specific purposes. These EEs are referred to virtual machines, and usually come with active node OS (NodeOS). They are available for active applications to process their packets or capsules and to control the processing. Significant research projects include: MIT ANTS (Active Node Transfer System [10]), University of Pennsylvania Switchware [9], Columbia University Netscript, USC/ISI Abone (Active Backbone) [13], Active Network Encapsulation Protocol ANEP [12] and BBN Smart Packet project [14]. To date, these developments have been mainly realized in software-based hosts (e.g., Linux-based systems) that offer the required programmability but lack the performance required in real networks. An exception is the Washington University ANN (Active Network Node) [15] implementation that introduces an FPGA-based CPU module that accommodates the active code and is added to a gigabit ATM switch backplane.

Other contributions such as Darwin [21] and Phoenix [22] have investigated mechanisms for delivering programmability to end-users. Darwin develops a set of customizable resource management mechanisms that allow service providers and applications to tailor resource management optimally for the service quality they require. Phoenix, similar in part to Openet, is a framework for programmable networks that allows easy control and deployment of services toward use of re-programmable network processors. Rather, our Openet approach makes use of enabling mechanisms such as AFM to demonstrate the benefits of programmability in real networks.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, the Active Flow Manipulation mechanism, through Openet, allows network service providers to introduce, on demand, intelligent controls that adapt network node behaviors dynamically in real networks. The dynamic priority change application demonstrates this concept in real-time, with the Passport silicon-based forwarding engines.

Openet enables network service providers to quickly respond to customer requirements by introducing the sort of AFM-enabled services in the form of Oplets. Service providers can build on smaller Oplets to develop more complex network functions. Two innovative applications in active network management (i.e., AIACE IP accounting and Regatta auto supervision) have demonstrated this ability. Furthermore, Openet is an open platform that can create new opportunities for service providers in deploying third-party network services on commercial routing switches. The deployment of innovative services on network nodes leads to the differentiation of network service providers.

It is observed that the AFM-based control-plane network services enhance functionality of commercial hardware like the Passport, without impeding performance of the forwarding plane. However, as seen in the ORE ANTS application, or data-plane services, rely largely on the performance of the control CPU. We are exploring a new hardware architecture [25] in which network services can reside in a new plane, the computing plane. Attached to the commercial network devices such as Alteon [24], this computing plane is being implemented with high performance computing technology such as FPGA, and it will allow active manipulation of an increased number as well as increased granularity of content flows. In addition, the computing plane will also improve the performance of data-plane network services that perform their own packet processing, as highly demanded in active networks and content networking.

VII. REFERENCES

[1] T. Lavian, R. Jaeger, J. Hollingsworth, “Open Programmable Architecture for Java-enable Network Devices”, Stanford Hot Interconnects, August 1999.

[2] The Openet Lab, “The Oplet Runtime Environment”, http://www.openetlab.org/ore.htm, March 2000

[3] F. Travostino, “Active IP Accounting Infrastructure,” IEEE OpenArch 2000, Tel Aviv, March 2000.

[4] Nortel Networks Corp., “Networking Concepts for the Passport 8000 Series Switch”, April 2000

[5] T. Lavian and P. Wang, “Active Networking On A Programmable Networking Platform”, IEEE OpenArch’01, Anchorage, Alaska, April 2001

[6] P. Wang, R. Jaeger, R. Duncan, T. Lavian and F. Travostino, “Enabling Active Networks services on a Gigabit Routing Switch”, The 2nd Workshop on Active Middleware Services in conjunction with the 9th IEEE International Symposium on High Performance Distribued Computing (HPDC-9), Pittsburgh, Pennsylvania, August 2000

[7] P. Wang, Y. Yemini, D. Florissi and J. Zinky, “A Distributed Resource Controller for QoS Applications”, NOMS 2000-IEEE/IFIP Network Operations and Management Symposium, Honolulu, Hawaii, April 2000

[8] David L. Tennenhouse, et al, “A Survey of Active Network Research”, IEEE Communications Magazine, Vol. 35, No. 1, January 1997

[9] D. Scott Alexander, et al , “The SwitchWare Active Network Architecture”, IEEE Network Special Issue on Active and Controllable Networks, vol. 12 no. 3, July 1998

[10] David J. Wetherall, John Guttag, and David L. Tennenhouse, “ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols”, IEEE OPENARCH'98, San Francisco, CA, April 1998.

[11] Y. Yemini and S. da Silva. “Towards Programmable Networks”, IFIP/IEEE Intl. Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, October 1996.

[12] D. Scott Alexander, Bob Braden, Carl A. Gunter, Alden W. Jackson, Angelos D. Keromytis, Gary J. Minden and David Wetherall, “ANEP: Active Network Encapsulation Protocol”, Active Networks Group, Request for Comments, http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt

[13] Active Network Backbone (ABone), http://www.isi.edu/abone/

[14] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, R. Rockwell and C. Partridge, “Smart Packets for Active Networks”, IEEE OpenArch 99, New York, March 1999

[15] D. Decasper, et al, “A Scalable High Performance Active Networks Node”, IEEE Network Magazine. Vol 37, Jan/Feb 1999

[16] Biswas, J. et al, “The IEEE P1520 standards initiative for programmable network interfaces”, IEEE Communication Magzine, Vol 36, Oct. 1998

[17] The CPIX (Common Programming Interface) forum, http://www.cpixforum.org/

[18] The Parlay Group, http://www.parlay.org/

[19] Intel Internet Exchange Arcitecture (IXA), http://developer.intel.com/design/ixa/white_paper.htm

[20] Solidum, http://www.solidum.com

[21] P. Chandra et al, “Darwin: Resource Management for Value-Added Customizable Network Service”, Proc. 6th IEEE ICNP, Austin, Oct. 1998

[22] David Putzolu, Sanjay Bakshi, Satyendra Yadav and Raj Yavatkar, The Phoenix Framework: A Practical Architecture for Programmable Networks, IEEE Communications Magazine, Vol 38, No 1, March 2000

[23] V. Sethaput, A. Onart and F. Travostino, “Regatta: A Framework for Automated Supervision of Network Clouds”, IEEE OpenArch’01, Anchorage, Alaska, April 2001

[24] Nortel Networks Corp., Alteon 180 Series Web Switch White Paper - “Scaling Next Generation Web Infrastructure with Content-Intelligent Switching”, April 2000

[25] S. Subramanian, R. Durairaj, J. Rasimas, F. Travostino, P. Wang, T. Lavian and D. Hoang, Practical Active Networking Services with Content-aware Gateways, to be published, Jan 2002