Intelligent Network Services through Active Flow Manipulation

Tal Lavian, Phil Wang, Franco Travostino, Siva Subramanian
{tlavian, pywang, travos, ssiva}@nortelnetworks.com
Advanced Technology Centre, **Nortel Networks Inc**.

Doan Hoang
dhoang@eecs.berkeley.edu
Department of Electrical Engineering and Computer Sciences, **University of California at Berkeley**

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

Keywords: Programmable networking, Intelligent network control, Active Flow Manipulation (AFM), Openet

Abstract

A significant challenge in today's Internet is the ability to efficiently introduce intelligent network services into commercial high performance network devices. This paper tackles the challenge by introducing the Active Flow Manipulation (AFM) mechanism, a key enabling technology of the programmable networking platform Openet. AFM enhances the control functionality of network devices through programmability. With AFM, customer network services can exercise intelligent network control by identifying specific flows and applying particular actions thereby altering their behavior in real-time. These services are dynamically deployed in the CPU-based control plane and are closely coupled with the silicon-based forwarding plane of the network node, without negatively impacting forwarding performance. The effectiveness of our approach is demonstrated by several experimental applications on a commercial network node.

I Introduction

The Internet though ubiquitous, is fragmented 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 and network providers to develop intelligent services to differentiate their offerings. It is also important to be able to rapidly introduce these intelligent services, such as QoS (Quality of Service), rapidly on-demand, to their clients. In brief, they are in need of a comprehensive programmable framework to manage their networks to serve their customers in a satisfactory manner.

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. The trend in commercial-grade routers and switches is to accelerate performance critical 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 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 new intelligent services into 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 can 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 customized control-plane services. Openet is a platform-neutral, service-based internetworking infrastructure developed by the Nortel Networks Technology Center, aiming to deliver dynamic network programmability to network devices. Although 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 devices lack user-programmability required to deploy intelligent services. Openet provides such programmability to enable users with control of the forwarding hardware. This programmability is manifested as the ability to alter the 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 data 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.

II Related Works

A significant amount of research has involved with enabling intelligent network services 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.

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 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 that 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 which introduces an FPGA-based CPU module that accommodates the active code and is added to a gigabit ATM switch backplane.

Other works such as Darwin [21] and Phoenix [22] have investigated mechanisms for delivering programmability to end-users. Darwin develops a set of customisable 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. Our Openet approach makes use of the AFM concept to demonstrate the benefits of programmability in real networks whereas Phoenix does not demonstrate practical applications on current commercial network hardware.

III Active Flow Manipulation

Sophisticated type of controls, which require time-consuming data processing, cannot react to traffic conditions in real-time and may incur negative impacts on switching performance. It is more practical to have a simple type of dynamic control that affects a vast amount of data in real-time without affecting forwarding performance of the devices. The Passport routing switch was designed for high-speed routing performance and not intended for software processing in the data path. This enables the forwarding plane of the Passport to be able to filter packets by performing matching operations on the packet headers. However, it also limits the number of actions possible on the identified packets. The reduced flexibility of the forward plane is however, offset by the programmability provided by the CPU-based control plane. Packets identified by the forwarding plane can be "passed" to the control plane CPU for more sophisticated customer-programmed actions. This leads to the concept of Active Flow Manipulation.

The AFM mechanism has two components a monitor component and an action component. Due to the nature of the traffic being predominantly session-based, it is not appropriate to consider filters and related actions on individual packets or actions on individual packets. It is more appropriate to think in terms of flows whose characteristics can be identified and whose behaviors can be altered by some primitive actions in real-time. As stated above, switches such as the Passport, provide hardware assist for primitive actions and require the use of CPU based processing for more complex actions. Depending upon the amount of hardware-assist available within the switch, complex actions might require longer processing times. As an example of large-grained flows and primitive actions consider the following scenarios; to exercise controls over "all TCP traffic to a particular service at a particular destination IP address", "all UDP datagrams generated from a set of identifiable source IP addresses to a particular destination address" and "all traffic passing through a particular port of a router". Due to the large granularity of the flows and the simplicity of the actions available in the hardware it becomes possible to manipulate large amounts of traffic at wire speed. This combined with the ability of the control plane to change the flow filter and actions continuously under program control gives rise to the concept of Active Flow Manipulation.

The flow is the first abstraction level necessary to enable AFM. Essential properties of a flow are to have identifiable elements that can be matched and cannot be divided further. The identifiable primitive elements of a flow are matched at wire-speed using hardware assists specific to the switching platform. We therefore focus on the primitive elements supported on the Passport routing switch as shown in Table 1. It is possible that a more sophisticated hardware platform may provide a richer set. An example of a flow on the Passport is a TCP flow which matches all packets that have a protocol field set to TCP.

Table 1 – The set of identifiable primitive elements for flows on the Passport 8600

Destination Address (DA)
Range of Destination Address (RDA)
Source Address (SA)
Range of Source Address (RSA)
Exact TCP protocol match (TCP)
Exact UDP protocol match (UDP)
Exact ICMP protocol match (ICMP)
Source Port number, for both TCP and UDP (SP)
Destination Port number for both TCP and UDP (DP)
TCP connection request (TCPReg)
ICMP request (ICMPReg)
DS field of a datagram (DS)
IP Frame fragment (FrameFrag)

An example of a composite flow would be "all TCP flows through a particular port on a particular destination machine (i.e., TCP, *, *, Destination IP, Destination Port). As seen in this example, a composite flow consists of one or more flows, formed by primitive elements, combined with a simple set of operators such as AND, OR, NOT, RANGE etc. If one were to consider all traffic leading to a particular destination IP address as a flow, then Table 2 lists a subset of the possible composite flows built using that flow.

Table 2 - A subset of composite flows using Destination Address

	Destination Address (DA)
	All traffic to a particular destination machine
Range of DA	All traffic to a range of destination machines
Source Address (SA)	All traffic between 2 particular machines
Range of SAs	All traffic from many source machines to a particular destination machine
TCP	All TCP flows to a particular destination machine
UDP	All data gram packets to a particular destination machine
ICMP	All ICMP messages to a particular destination machine
ICMP Request	All ICMP requests to a particular destination machine
TCP ACK	All TCP acknowledgements to a particular destination machine
TCP RST	All TCP connection with the RST bit set
DP (TCP)	All TCP flows to a particular service in a particular server machine
DP (UDP)	All UDP datagram to a particular service in a particular server machine
SA-SP (TCP)	All TCP flows originated from a particular client of a source machine to a particular destination machine
SA-SP (UDP)	All UDP datagram originated from a client of a source machine to a particular destination machine
IP Fragments	All IP fragments to a particular destination machine
DS Field	All traffic of a particular QoS class to a particular destination machine
VLAN	All traffic from a particular VLAN to a particular destination machine
Switch-Port	All traffic on a particular switch port to a particular destination machine

It is possible that a given hardware architecture might support a set of primitive elements and a subset of composite flows. Any composite flows not supported by the hardware have to be created by operating on these flows under program control i.e. in software. It goes without saying that processing composite flows in software requires more real-time. However, the idea of controlling composite flows is very powerful and can be an enabler for new applications as will be seen later in the paper.

Monitoring primitive as well as composite flows form one portion of the AFM mechanism. The actions required for AFM can also be defined in similar fashion. Primitive actions are atomic in nature and can be associated with an identifiable flow, either primitive or composite to manipulate the traffic contained within that flow. Operators combined with primitive actions give rise to composite actions. A subset of actions of interest is shown in Table 3. Combining composite flows and composite actions generates a set of desirable flow manipulations. The ability to generate or alter flow manipulations at real time results in Active Flow Manipulation. Examples of flow manipulations may be "Change the priority of all traffic destined to a particular service on a particular machine", "Stop all traffic out of a particular link of a router".

By definition, AFM requires the ability to change the flow-action combinations at real-time. This is very easily implemented under program control, i.e. on a CPU-based plane. Thus a programmable network platform forms the basis of AFM. Openet is such a platform that can dynamically inject and control services. Such services typically reside in the control plane of a network node. By allowing AFM, Openet 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 on the switch. Typical service examples include packet filtering, firewall protection, dynamic flow classification, altering flow priority and intercepting special control messages for further processing.

IV Openet on PASSPORT Routing Switch

Openet originated from the open programmable architecture for Java-enabled network devices [1] and evolved with later works [2,5,6]. It is platform-neutral, and works closely with commercial nodes such as the Passport to efficiently use the available hardware resources.

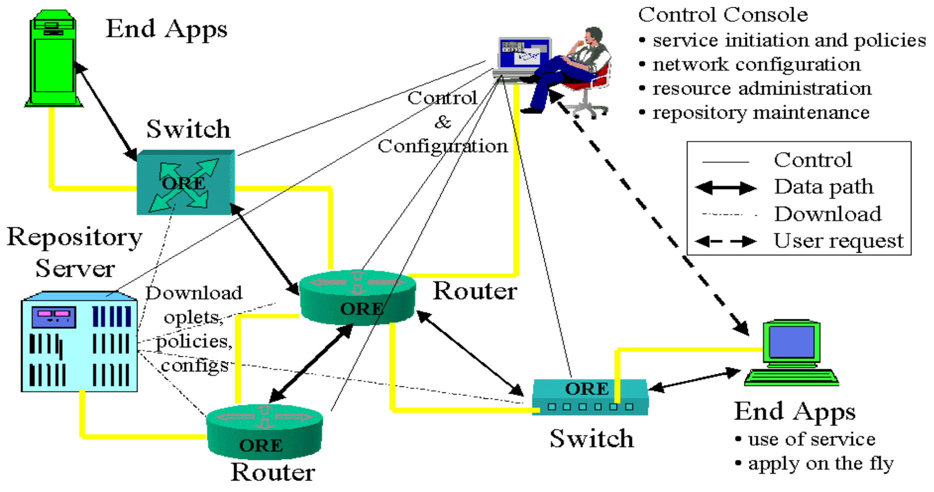


Figure 5. The Openet architecture within one network

a. Openet

Figure 1 depicts the Openet architecture in a distributed network that consists of routers switches, end hosts, repository servers and control consoles. The routers and switches download base functions and services from the repository servers, as demanded by end applications and control consoles. Repository servers and control consoles are Openet-specific. Repository servers run downloading services (e.g., HTTP) 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 consists of 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).

The Oplet Runtime Environment (ORE) is the core of the Openet infrastructure. It is an open object-oriented networking environment for customer service creation and deployment. At runtime, it supports injecting customized software of network services into a node 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 achieves a high level of performance by introducing two separated working planes control and forwarding, as depicted in Figure 2. The forwarding plane along the data path is implemented using ASICs that can forward packets at up to 256 Gbps (gigabits per seconds) without consuming any CPU resource. Conventional routers and software-based routing systems involve CPUs in both packet forwarding and forwarding control, hence reduce the level of performance that they can achieve.

The control plane, however is based on a CPU resource and contains the embedded Java VM. It executes ORE and that in turn enables execution of diversified network services. Thus, Openet introduces programmability into the otherwise rigid routing switch making it capable of supporting AFM.

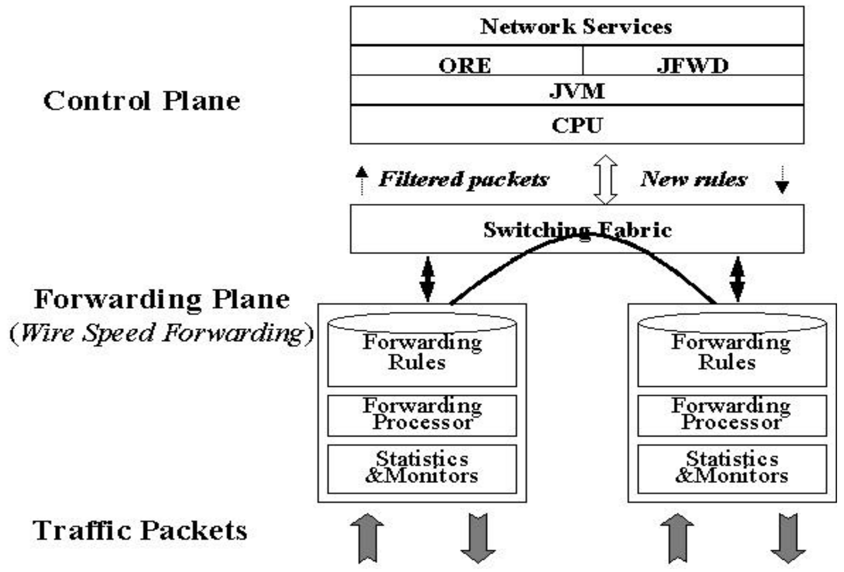


Figure 3. The Passport routing switch architecture

c. Service Deployment

Network services are composed of normal Java objects, and encapsulated by Oplets. An 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 forward. 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. System services are low-level network services that have direct access to the hardware features, e.g., JFWD that provides neutral Java APIs to alter the hardware routing and forwarding behaviors. They require particular hardware knowledge and are implemented using native programming interfaces or the hardware instrumentation. Standard Services provide the ORE standard 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. Function Services provide common functionality or utilities used to rapidly create user-level services, and are usually intermediate services coming with the ORE release or contributed by the third party. User Services are the customers' application services for particular purposes.

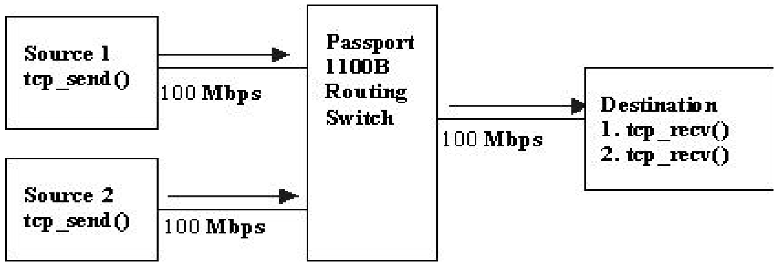
Service deployment i.e. inject network services to real networks, requires downloading and activating the service code within the ORE on commercial network hardware like the Passport. There are at least three ways to do dynamic service injection, using the ORE shell service, the ORE startup service or a user service initiation service.

IV AFM-enabled applications

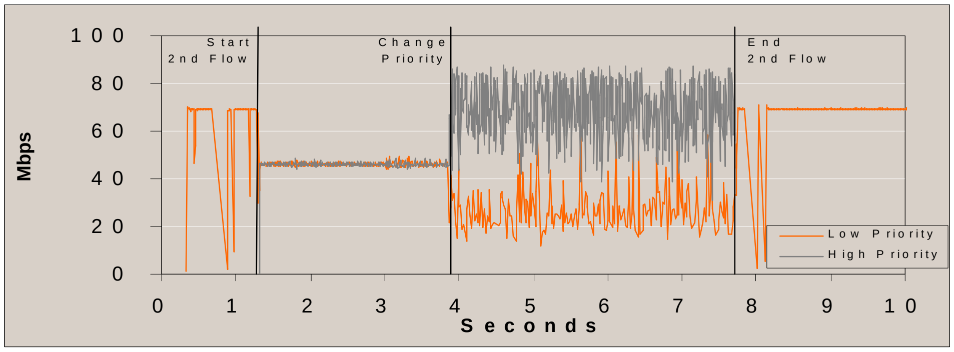
In this section three applications are described to demonstrate applications enabled by the AFM mechanism with the Openet programmability, based on the Passport routing switch and other networking platforms.

a. Dynamic Flow Priority Change in Real-time

The dynamic flow priority change is a simple and effective control-plane network service that applies AFM to alter packet forwarding priorities of particular flows in real-time. The experimental network depicted in Figure 3a is established with the Passport 1100B routing switch and three hosts that are Linux-based PC systems.



(a) Network Layout



(b) Flow thruputs at the destination

Figure 2. Two TCP flows competing for link bandwidth with dynamic priority change

The experiment procedure is as follows. 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 monitor and identify particular flows and increase the packet priority of a specific flow. 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).

The bandwidth jumping of the second flow at time 3.8 seconds shows that the Passport forwarding plane carries out packet filtering at the wire-speed, without any performance reduction. The reason for this being, the control services do not require packet processing in the forward plane. The forwarding engine processes and forwards packets while the control CPU executes the Java code implementing the AFM mechanism.

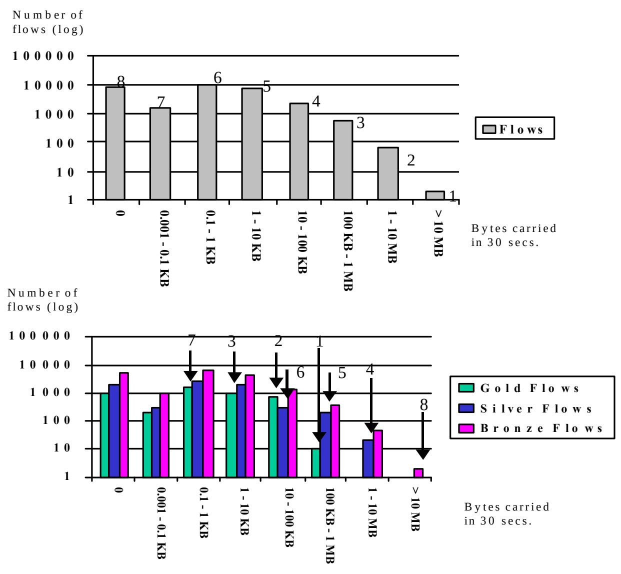
Even though the experiment and results are not groundbreaking, this service indicates an immediate benefit of active detection of flows and dynamic adjustment of packet priorities on commercial-grade nodes. It can be used widely in traffic control such as end-to-end video and audio traffic, and QoS mechanisms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



a) In this example, a network-node organizes about 2 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.

b) The node now structures the same accounting data into QoS-flavored flows (same X and Y axis 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).

Figure 3. Results of a flow monitoring scenario under AIACE

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.

- How much accounting data the application wants to handle from a given network node;
- How the network node should weight its accounting data for flows, decide what to mine out, and package it within the aforementioned limit;

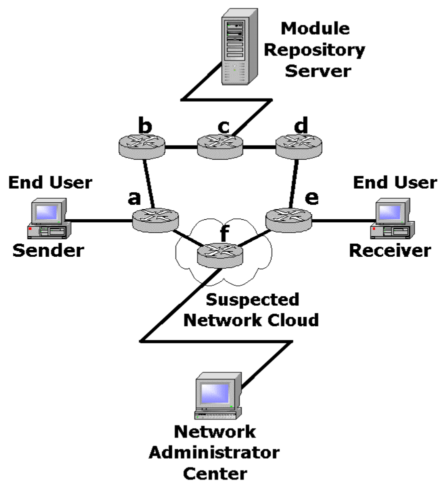
- Which accuracy is expected from the network node while performing this accounting data mining.

The opportunities in Step 2 become evident in the example showed in Figure 4. The network under analysis is QoS-enabled and three QoS classes-gold, silver, and bronze-are defined. 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.

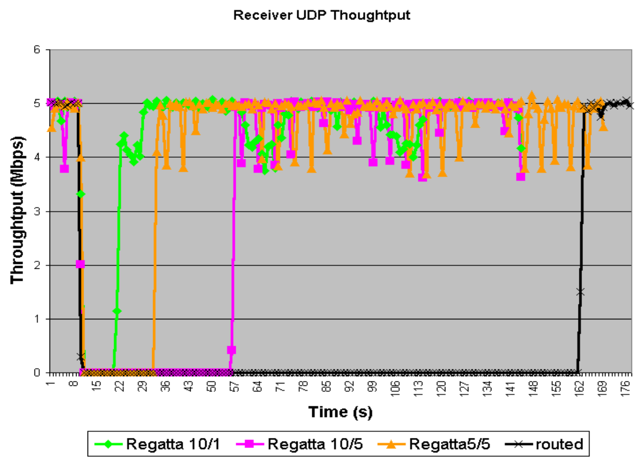
AIACE's accounting plug-ins realize Steps 1 through 3 by operating at both network nodes and accounting server. Some plug-ins that define the weighting algorithms are loaded and executed only at network nodes. Other plug-ins that implement the accounting wire protocol and its capability to drive the 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 failure semantics that can be described as "the link layer is always up, but the IP layer sometimes suddenly fails to forward PDUs".



(a) Experimental Setup



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

Figure 4. Dynamic flow bypass using Regatta

The network operator can thus aims 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.

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 the JFWD IP 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 get 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.

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).

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

V Conclusions and Future Work

The Active Flow Manipulation mechanism, built into Openet, allows network service providers to introduce, on demand, innovative services 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 allows network service providers to quickly respond to customer requirements by introducing new 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 network services enhance functionality of commercial hardware like the Passport, without impeding performance of the forwarding plane. However, sophisticated data-plane services rely largely on the performance of the control CPU. We are exploring a new hardware architecture in which intelligent network services can be deployed in a new plane, the computing plane. On the Passport routing switch, this plane which is being implemented with high performance computing technology, will allow active manipulation of an increased number as well as increased granularity of flows.

VI References

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

[2] R. Duncan, "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] R Jaeger, S. Bhattacharjiee, Hollingsworth, R. Duncan, T. Lavian and F. Travostino, "Integrating Active Networking and Commercial-Grade Routing Integrated Active Networking and Commercial-Grade Routing Platforms", Usenix: Intelligence at the Network Edge, San Francisco, March 2000

[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 Programma-

ble Networks", IFIP/IEEE Intl. Workshop on Distributed Systems: Operations and Management, L'Aquila, Italy, October 1996.

[12] D. Scott Alexander, et al, "ANEP: Active Network Encapsulation Protocol", Active Networks Group, Request for Comments, http://www.cis.upenn.edu/~switchware/ANEP/docs/ANEP.txt, July 1977

[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" , submitted in publication, Oct. 2000.