

Faculty of Informatics

Faculty of Informatics - Papers

University of Wollongong

Year 2002

An architecture for carrier grade
programmable networks

T. V. Nguyen*

P. Boustead†

F. Safaei‡

*University of Wollongong

†University of Wollongong, boustead@uow.edu.au

‡University of Wollongong, farzad@uow.edu.au

This article was originally published as: Nguyen, TV, Boustead P & Safaei, F, An architecture for carrier grade programmable networks, GLOBECOM '02 IEEE Global Telecommunications Conference, 17-21 November 2002, vol 2, 2016-2020. Copyright IEEE 2002.

This paper is posted at Research Online.

<http://ro.uow.edu.au/infopapers/187>

An Architecture for Carrier Grade Programmable Networks

Thanh Vinh Nguyen, Paul Boustead, Farzad Safaei
Telecommunications and Information Technology Research Institute
University of Wollongong, Australia
email: {vinh,paul}@ittr.uow.edu.au; farzad@uow.edu.au

Abstract-Active and programmable networks allow innovative new services to be deployed rapidly. However, in a carrier grade network, it is imperative to maintain a scalable fast path mechanism so that the delay and throughput requirements are met. This is particularly important since in many cases network-level processing is only needed for a subset of packets and the remainder of traffic must be forwarded on the fast path. It is a challenge to design a cost effective node architecture that can satisfy this requirement. Current models are often 'revolutionary' and may not scale to the required performance levels of a carrier grade network. In this paper we develop an evolutionary architecture that uses MPLS label stacks to enable efficient and scalable extraction of active packets from the fast path without penalising the rest of the traffic. Our model can be used to extend the reach of programmable networks over legacy infrastructure and provide a migratory path for existing carriers towards a programmable infrastructure.

I. INTRODUCTION

Programmable, or active¹, networking has been widely promoted as a future solution for fast and easy deployment of new innovative services over the Internet. The vision is that by offering programmability at network nodes customised code can be injected into the network to enable service provisioning without going through any standardisation process. Since research in this area was initiated a few years ago, there has been much research effort, resulting in a large amount of published literature ([1], [2], [3], [4] and references therein). However, we believe that even with the current state of the art, introducing programmability in carrier networks without significantly changing existing high-speed packet forwarding mechanisms is still a challenge that has not been totally addressed.

The challenge in carrier-grade programmable networking is to achieve scalability while having enough flexibility to support a wide range of services. In this environment, there is little processing time available for each packet, which places a constraint on the amount of processing that can be performed. The bottle neck is expected to persist in the foreseeable future, since currently transmission speed is growing at 200% per year, outpacing the growth of processing power, which still keeps up with the Moore's law [5]. This is the reason why 'revolutionary' software-based active router models such as ANTS [1], which requires processing for every packet, are not suitable [6]. There have been more practical approaches, such as the Tempest [7], which aims at a less flexible architecture by introducing programmability only in the node's control plane. Tempest, therefore, only supports applications that does not require data plane programmability. Section V will briefly review existing related architectures.

In this paper we propose an architecture that we believe is practical and suitable for the carrier grade environment. This architecture is designed for Multi-Protocol Label Switching (MPLS)

¹In this paper, we use the terms 'programmable' and 'active' interchangeably

networks - an increasingly popular carrier-grade networking platform. Utilising MPLS traffic engineering capabilities, we have developed an architecture that is scalable, flexible enough to support a wide spectrum of applications and is 'evolutionary', i.e. allowing carrier-grade programmable networking in the *near future* using legacy equipment.

The rest of the paper is organised as follows: Section II will specify the requirements of a carrier-grade programmable network architecture, Section III will describe the proposed architecture and the advantages ensued, Section IV will describe the current development and implementation status of the architecture, and illustrate the architecture's feasibility through an example application scenario. Related literature will be discussed in Section V and we will conclude in Section VI

II. PROGRAMMABLE NETWORKING IN A CARRIER-GRADE ENVIRONMENT

A key characteristic of the carrier network environment is extremely high transmission speed, typically in the order of tens of gigabits per second. Coupled with a mismatch in growth of transmission speed and processing power, such environment does not allow a practical architecture to afford active processing for every packet.

Fortunately, our observation is that in practice there is a large class of applications that require network-based processing for only a subset of packets, for example fault detection and recovery, performance monitoring or customised routing. In such applications the active packets are used for control purpose, while the remaining are data packets. Though there are applications that require processing for 100% of packets, such as encryption or content adaptation, they can only comprise a small portion of traffic since the predominant function of networks is transport. We therefore believe it is practical and desirable to develop an architecture that exploits this trait to maintain a fast data path and support selective processing for a portion of traffic. Processing intensive applications can also be supported but to a lesser degree, e.g. not at every network node but at special-purpose processor farms (nodes with very high processing power) deployed inside the network.

Having relaxed the processing requirements to a subset of packets, it is imperative to have an efficient and robust mechanism for identifying and extracting active packets from the data path. The challenge lies in the fact that such process may impose extra overhead on other non-active packets. Furthermore, a packet may even be active at some nodes and follow the fast path at others. Current approaches, e.g. those described in [8] and [9], are usually based on filtering to extract active packets. Active packets are generally selected according to (a combination of) header information, such as IP addresses and TCP/UDP port

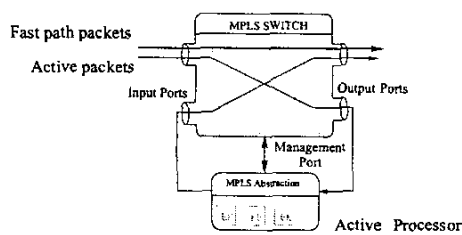


Fig. 1. Active node architecture

numbers. A drawback is filtering may affect every packet. The data path, while having high performance, needs to be flexible enough so that it can be affected by (the processing of) active packets. The ability to accommodate active packets in a portion of traffic will only be useful if active packets are able to affect what happens in the fast path. This is required to allow applications with customised routing and forwarding mechanisms to be deployed with only a few active packets per flow. Finally, to make carrier-grade programmable networking possible in the *near future*, an evolutionary approach with a clear migration path from current networks is vital. In particular, it is important to ensure that the architecture will be able to integrate with legacy non-active networks and non active packet flows.

In summary, the following is desired:

1. *Flexible fast path:* Non-active packets must be forwarded on a fast path, which has minimal extra overhead and can be influenced by active applications.
2. *Robust active packet identification and extraction:* An active packet should only be processed where necessary. Otherwise it should be forwarded on the fast path and not incur any additional penalty.
3. *Processing intensive application support:* To support these applications with processor farms a mechanism to divert traffic there seamlessly when necessary is needed.
4. *A clear migration path.*

III. AN MPLS-BASED CARRIER-GRADE PROGRAMMABLE NETWORK ARCHITECTURE

A. Node Architecture

In our architecture, each programmable node has two main components - a fast path and an active processor. The fast path is an MPLS switch. An active processor is connected to an output port and an input port of the switch (Figure 1), or may be integrated in future generation switches, to provide processing power. If a packet is to be processed at a node, it will be placed on a label switched path (LSP) that goes to the active processor at that node. Otherwise it is forwarded directly through the switching fabric on a separate LSP. Therefore, at each node, each packet will require only one label look-up, regardless of it being active or non-active and the fast path does not experience any extra overhead due to the presence of active packets. The active processor is also connected to the switch management port. A flexible fast path is achieved by allowing the active processor control the switch through this interface. Active applications residing in the processor thus can alter fast path

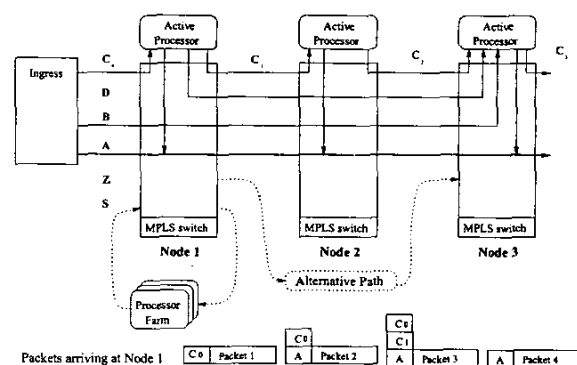


Fig. 2. Active and non-active label switched paths

behaviour, e.g. updating the MPLS switch's label mappings. The active processor has an MPLS abstraction layer that acts as a middleware between the MPLS fast path and the processor's execution environments (EE). This layer provides fast path information to the EEs and/or polices their actions on the switch. This node architecture allows programmable nodes to be constructed from legacy equipment by adding adjunct processors to high-speed MPLS switches.

B. Using MPLS Label Stack for Packet Forwarding

The architecture defines two types of LSPs: active and non-active. An active LSP starts and/or ends at active processors, but may go through the fast path of intermediate nodes. Note that active LSPs terminate at active processors to enable delivery of active packets to the processors and do not necessarily mean active packets will stop there. In fact, packets can be recycled back to the switch through the input port for further transmission. A non-active LSP carries only non-active packets (i.e. it goes through the fast path of the nodes it traverses). Figure 2 shows an example of three active nodes connecting by LSPs A, B, C_0 to C_3 , D, S and Z. By our definitions, LSPs C_0 to C_3 , B and D are active. It is important to note that packets on LSP D are "active" in node 1 and node 3 and "non-active" in node 2. LSP S is a path for diverting traffic from node 1 to a processor farm, and LSP Z represents an alternative path from node 1 to node 3. LSP A is non-active; it is a multipoint-to-point LSP since packets from immediate active processors are able to join the flow, which aims to reduce the total number of LSPs required.

We will now describe the architecture's capabilities and underlying mechanisms using examples based on this network. The letters (A, B, C_0 ...) are used to represent labels on these LSPs, but the actual label values will be different on each link due to MPLS label swapping.

• **Active packet selective delivery:** By using the appropriate label stack to forward an active packet along a suitable LSP, a packet can be delivered to active processors only at selected nodes, while bypassing processing at others.

As an example, consider Packet 1 in Figure 2. It arrives at Node 1 with label C_0 and is delivered to the active processor. The active application in Node 1 then may swap the current la-

bel for C_1 to enable further processing in Node 2, D to enable processing at Node 3 but not Node 2, or A to forward the packet on along the fast path.

It is also possible for the ingress to manipulate label stacks and decide where packets should be processed. An example is Packet 2 in the same figure. This packet arrives at Node 1 with labels C_0 and A. After processing in Node 1's active processor, the top label (C_0) is popped and the packets is forwarded using the second label (A). It now follows the fast path for the rest of the route. Packet 3, on the other hand, needs processing at both Node 1 and 2, and is thus labelled with C_0 , C_1 and A. The top two labels will cause the packet to be processed at both Nodes 1 and 2 as required. After that it will also follow the fast path.

The selection of processing locations depends on two factors - specific application requirements and available processing power at nodes. The ability to perform active processing at pinpoint selective locations is important in making the architecture scalable, since it allows efficient use of this limited resource.

- **Influencing the fast path:** The following example demonstrates why active processors are required to be able to affect fast path behaviours. Consider a packet flow in LSP "A" (e.g. Packet 4), normally such packets are forwarded along fast path LSP A from Node 1 to Node 3. Suppose that a monitoring application in Node 1 decides that the flow needs to be diverted to the alternative path (e.g. due to a link failure or congestion), it can request the active processor to modify the switch's MPLS table so that an incoming packet with label A has its label swapped with the correct label for this LSP at the end of the alternative path and Z is then pushed onto the stack. Such packets are now transmitted to Node 3 via the alternative route. LSP Z can either be pre-setup or created on demand.

Such flexibility is achieved by allowing the active processor to access the switch management port and alter switch settings (e.g. MPLS mapping table or QoS settings). This enables support for applications that have some active packets controlling the forwarding of the remaining traffic, such as application-specific routing or failure recovery.

- **Incorporating processor farms:** The range of applications supported by the architecture can be broadened to include those requiring intensive processing by deploying processor farms inside the network. Utilising the traffic diversion ability described above, processor-intensive traffic can be seamlessly diverted to processor farms and back.

As an example, imagine the flow in LSP C_0 in the network above belongs to such application, e.g. encryption or multimedia transcoding. If a processor farm is needed to meet processing requirement for this application, an LSP (LSP S) is created to connect Node 1 to that processor farm, and then the fast path label mappings are modified so that label S is pushed onto all the packets in the flow. The flow will now get diverted to the processor farm for processing, then come back for further forwarding.

- **Packet labelling options:** The use of LSPs and label stacks in the mechanisms described above makes an important assumptions on the edge (of the carrier network) node's capability - it must be able to classify incoming packets and add the ap-

propriate labels so that packets are placed on the suitable LSPs. Our solution is to use either an active node at the edge or a content switch that is capable of filtering packets based on a combination of Layer 2 to 7 headers. In effect, instead of classifying packets at each node such as in [8], we have pushed the requirement to the network edge. The reason is the traffic load at such locations are expected to be less than in the core.

- **Creating LSPs:** We may need to develop new methods to create active and non-active LSPs required for an application. A solution is for the first active packet in the stream to negotiate which active processors to be used on the route and create the LSPs "on the fly" between them. This packet would follow pre-setup LSPs between the processors - C_0 , C_1 , C_2 , and C_3 in the above example. It examines capabilities of each node, determines what operation is to be performed in that node and configures LSPs for the remaining packets. The result will be creation of LSPs that bypass active processing in some nodes and obtain active processing in others. LSPs may also be created on demand by an active application, such as LSP S in the previous example. The ability of active applications to modify the MPLS switch settings would facilitate such operations.

IV. DEVELOPMENT AND IMPLEMENTATION

A. Development direction

Our aim is to continue developing this architecture as a platform for integrating multiple services using the programmable virtual network (PVN) concept. A PVN is an overlay topology of virtual nodes and links created over a physical programmable network infrastructure. It is envisioned to be commonly offered as generic service by a network provider to customers that might be service providers themselves. Service providers (or PVN owners) will lease a virtual network topology from the network provider (or PVN provider) that includes both transmission and processing resources. The aim is to provide a customer with a virtual network which is also *programmable* and can be adapted to its business and operational requirements. For example, each service provider may instantiate its own routing protocol, billing, accounting, management, and custom services independently and securely.

We see this as a desirable development due to number of reasons. Firstly, it will be an efficient and convenient means of hiring and managing network resources (bandwidth, label space, storage and computation power) between network owners and service providers, which is particularly desirable in a carrier network environment. Secondly, the PVN is well suited to our programmable architecture since it is an application for which there is only a small percentage of active packets (service provider's traffic) and the majority of packets (end users' traffic) follow the fast path. Furthermore, the use of MPLS LSPs in our architecture will provide a convenient means of isolating and managing traffic in different PVNs.

Although similar concepts have been discussed in the literature ([7], [10] and [11]), our goal is a complete architecture that can support a large range of applications and fully address research issues including scalability, migratory path, multiple network spanning (PVN peering), quick PVN provisioning.

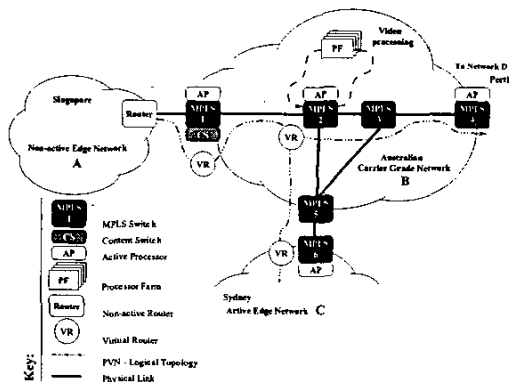


Fig. 3. Application scenario

B. An Application Scenario

This section will demonstrate how the architecture will enable deployment of useful services in a network consisting of legacy equipment though an example application.

In this example a fictitious Singapore company “SingVideo” is providing a video chat room service in Singapore that scales to a large number of members cost-effectively. As we assume that the service implements a multimedia multicast mechanism and supports features like transcoding and addition of local or targeted advertising. The service is being extended to Australia using an Australian carrier. Requirements include network reach to customer bases in several Australian metropolitan areas, the ability to deploy proprietary multicasting and billing protocols, multimedia processing capabilities.

Figure 3 summarizes the scenario. Network “A” in Singapore is designed and implemented (using specialised hardware) to support SingVideo’s proprietary service. The new Australian customer bases are networks “C” and “D”. Service from an Australian carrier (network “B”) is used to enable SingVideo to extend their network to the new customers without installing their own hardware.

In order to support the service over the Australian carrier grade network, active processors are connected to several legacy MPLS switches: the one peering the Singapore network, and some located in networks C and D. Suppose that for efficient multicast, a processor is also added to a switch within the core of the carrier network to support an intermediate node between Sydney and Perth. A virtual topology (a PVN) is created over these nodes to provide the necessary network reach. Resources at these nodes (computation, storage, label space) are allocated to SingVideo in the form of “virtual nodes”. LSPs similar to example in Figure 2 are created to connect these nodes. Customised protocols are deployed in the processors to process protocol messages forwarded using active LSPs.

Since “A” is a proprietary network a content switch is placed at the edge router of the Australian carrier grade network. It uses filters to extract packets belonging to the “SingVideo” PVN and push the appropriate MPLS labels onto the stack to forward them on suitable LSPs. In the remainder of the carrier grade

network content switches are not required since the edge nodes in C and D are programmable and can place packets onto the appropriate LSPs.

For processing intensive work that can not be supported by the active processors, processor farms may be needed. We assume that the router in network “A” wishes to multicast multimedia packets to nodes in “C” and “D”. These packets are extracted by the content switch in node 1 and placed on an LSP destined for node 2. At node 2 they are forwarded by a custom multicast protocol to networks “D” and “C”. Suppose the virtual router in node 2 determines that the multicast branch destined for network “D” requires a high level of processing (e.g. addition of localised advertising). If there is insufficient processing power in node 2, the label for a LSP linking a nearby processor farm (PF) is pushed onto the stack for all packets in this media stream to divert them to that node. The processing intensive work is performed in the processor farm and then packets are returned to the node 2 to be forwarded on to the destination.

C. Test Bed

We have implemented a testbed over which we demonstrated the feasibility of our carrier-grade programmable network architecture and the PVN concept. In our testbed each active node is constructed from an emulated MPLS switch, based on the Cambridge University NetX² MPLS implementation on Linux kernel, connected to an active processor. We created a management interface for the MPLS switch which is similar to a basic Cisco IOS management interface and is accessible to the active processor. The testbed currently has 5 active nodes and is being extended to contain a commercial grade Cisco 7200 MPLS switch.

We have demonstrated the key mechanisms including active packet identification and extraction, using active packets to create label switched paths (LSPs) or change existing label mappings. We have also successfully created multiple PVNs running different routing protocols. We achieved this by maintaining separate routing databases and tables for each PVN in the active processor. However, since forwarding in the data path uses the MPLS switch’s table, these per-PVN tables are merged into the common routing table via the management interface.

V. RELATED WORK

A. Software-based Architectures

A number of prominent pioneering active network architectures are ANTS [1] developed at Massachusetts Institute of Technology, SwitchWare [3] developed at the University of Pennsylvania, Genesis kernel [12] and NetScript [2] at Columbia University.

These early works share a common characteristic: they are all software based models that consider the network as a distributed computer and active nodes as general purpose processors, and which require active processing for every packets. Although such design allows maximum flexibility, the resulting overhead adversely affects their performance and scalability. In

²<http://www.cl.cam.ac.uk/Research/SRG/netos/netx/>

comparison, our approach of using MPLS label stacking has removed the overhead required for active packet extraction.

B. Scalable Active Node Architectures

The Active Network Node (ANN) [13], and similarly work in [14], proposed architectures that have multiple extensible processing engines plugged into a high speed backbone to provide processing power. Issues dealt with were mainly at node level and focused on computation resource management. Works in [15], [16] and [8] introduced active networking into Nortel Networks' commercial-grade multi-gigabit Passport routers using Openet technology. In these devices the forwarding plane is implemented using ASICs while the control plane is CPU-based and contains an embedded Java virtual machine (JVM). Services are deployed in the programmable control plane in form of applets running inside the JVM. Incoming packets are classified using combinations of hardware filters to identify flows or to distinguish active and non-active packets. Although the approach has the advantage of having high performance, the use of customised hardware filters are essential.

C. Programmable Virtual Network

The Virtual Active Network (VAN) concept [10] is very similar to the PVN. The main contribution of this work is a framework that splits responsibilities between network providers and service providers. The authors envisioned an active network shared by multiple customers, each allowed to install, run and manage services independently using VANs. However, in this work an underlying active network architecture is not yet defined and the issues related to the carrier environment, especially scalability problems, have not been considered. Tempest [7] provides a framework in which virtual private networks (VPN) are dynamically created and assigned dedicated sets of network resources in the form of switchlets. It has a programmable control plane where users can inject customised control architecture. The Tempest approach has limited flexibility because active packets are not allowed. The architecture also excludes processor intensive applications. Our aim, on the other hand, is to support a wider range of applications, including some processor intensive applications, without penalty on performance or scalability.

Work in [11] addressed a resource optimisation problem seen by the network provider in provisioning multiple PVNs with different topology and resource requirements over a physical network. This work is compatible and complementary to ours.

VI. CONCLUSION AND FUTURE WORK

The migration of carriers' IP networks to MPLS is well underway. The widespread deployment of programmable networks in the near-term would be more realistic if it is consistent with this trend. In this paper we have proposed an evolutionary approach to programmable carrier-grade networking based on MPLS traffic management capabilities that offers the following

advantages:

1. **Scalability:** achieved through a fast data path, an efficient means of active packet identification and delivery and moving packet classification to network edge.
2. **Flexibility:** although most applications are expected to have a few active packets, those requiring heavy processing can still be supported by incorporating processor farms, without compromising scalability.
3. **Migratory path:** gradual deployment of processing capability and expansion of active network applications over a legacy infrastructure is possible.

VII. ACKNOWLEDGEMENT

The support of the Cooperative Research Centre for Smart Internet Technology (<http://www.smartinternet.com.au>) for this work is hereby acknowledged.

REFERENCES

- [1] D. Wetherall, J. Guttag, and D. Tennenhouse. Ants: network services without the red tape. *Computer*, 32(4):42–48, April 1999.
- [2] Y. Yemini and S. da Silva. Towards programmable networks. In *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy, October 1996.
- [3] D. Alexander, W. Arbaugh, M. Hicks, P. Kakkar, A. Keromytis, J. Moore, C. Gunder, S. Nettles, and J. Smit. The switchware active network architecture. *IEEE Network*, May/June 1998.
- [4] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, 1997.
- [5] Werner Bux, Wolfgang Denzel, Ton Engbersen, and Ronald Luijten. Technologies and building blocks for fast packet forwarding. *IEEE Communications Magazine*, 39(1):70–77, January 2001.
- [6] David W. Active network vision and reality: lessons from a capsule-based system. In *Symposium on Operating Systems Principles*, pages 64–79, 1999.
- [7] S. Rooney, J. E. van der Merwe, S. A. Crosby, and I. M. Leslie. The tempest, a framework for safe, resource assured, programmable networks. *IEEE Communications Magazine*, 36:42–53, October 1998.
- [8] T. Lavian and P. Wang. Active networking on a programmable networking platform. In *OPENARCH 2001. 2001 IEEE Fourth Conference on*, pages 98–107, March 2001.
- [9] G. Hjalmtsson. Flexible toolkit for programming networks using a commodity operating system. In *OPENARCH 2000. IEEE Third Conference on*, pages 98–107, March 2000.
- [10] T. Brunner and R. Stadler. Service management in multiparty active networks. *IEEE Communications Magazine*, 38(3):144–151, 2000.
- [11] F. Safaei, I. Ouveysi, M. Zukerman, and R. Pattie. Carrier-scale programmable networks: wholesaler platform and resource optimization. *Selected Areas in Communications, IEEE Journal on*, 19(3):566–573, March 2001.
- [12] A. Campbell, M. Kounavis, D. Villela, J. Vicente, H. De Meer, K. Miki, and K. Kalaichelva. Spawning networks. *IEEE Network*, July/August:16–29, 1999.
- [13] Parulkar G. Choi S. DeHart J. Wolf T. Decasper, D. and B. Plattner. A scalable high-performance active network node. *IEEE Network*, 13(1):8–19, January/February 1999.
- [14] T. Wolf and J. Turner. Design issues for high-performance active routers. *IEEE Journal on Selected Areas in Communications*, 19(3):404–409, March 2001.
- [15] T. Lavian, P. Wang, F. Travostino, S. Subramanian, V. Sethapat D. Hoang, and D. Culler. Enabling active flow manipulation in silicon-based network forwarding engine. *IEEE Journal of Communications and Networks*, pages 78–87, March 2001.
- [16] T. Lavian, P. Wang F. Travostino, S. Subramanian, D. Hoang, and V. Sethapat. Intelligent network services through active flow manipulation. In *Intelligent Network Workshop*, pages 73–82, 2001.