Internet Engineering Task Force
INTERNET DRAFT

Authors
Ashish Mehra
Dinesh Verma
Renu Tewari
Tsipora Barzilai
Douglas Freimuth
Mandis Beigi
IBM T J Watson Research Center
14 July 2000

Policy-Based Differentiated Services on AIX
<draft-mehra-diffserv-aix-00.txt>

Status of Memo

Abstract

This draft presents a policy-based Differentiated Services [DSARCH] architecture that has been realized on the AIX operating system. We briefly motivate the benefits of supporting DiffServ functions on servers using policy-based networking, and outline the requirements for DiffServ-enabled servers. We then describe the policy-based DiffServ infrastructure we have developed for AIX. Experimental results using Web server workloads demonstrate the efficacy of the DiffServ mechanisms provided. We also outline ongoing work in deploying diffserv-capable AIX servers on Internet2.

1. Introduction

The differentiated services (DiffServ) architecture and model [DSARCH, DSFRAME] provides the framework on which network operators can support different classes of service in a network. In this framework, an access (or edge) device determines the class of service of data packets passing through it and modifies the Type of Service (TOS) byte of the IP header [DSHEAD]. Routers offer different classes of service in the network by supporting a variety of Per Hop Behaviors (PHBs)(e.g., Assured Forwarding [ASREF], Expedited Forwarding [EFREF]). In addition to standardizing service classes and their PHB´s, a continued focus remains on exploring how various components within the framework can together meet the end-to-end quality of service requirements of network applications. Towards that end, there have been proposals on the architecture of DiffServ access boxes [DIFFEDGE] as well as discussion of how differentiated services can inter-operate with integrated services and signaled quality of service (QoS) [DIFFEDGE] [RSVPDIFF].

```
        / Stub   \       /   Transit   \       /  Stub  \
       / Network  \     /    Network    \     /  Network \
|---| |        |---|   |---|          |---|   |---|       | |---|
|Tx |-|        |ER1|---|BR1|          |BR2|---|ER2|       |-|Rx |
|---| |        |-- |   |---|          |---|   |---|       | |---|
       \         /     \              /      \         /
        \       /       \            /        \       /
```

Figure 1: Sample Network Configuration

1.1. DiffServ Servers

DiffServ deployment is typically perceived to be carried out as in Figure 1, which shows two stub networks connected together with a transit network. The working assumption has been that the stub networks would support integrated services while the transit network(s) would support differentiated services. We have considered a variety of deployment scenarios for IntServ and DiffServ [DSERVERS] and argued in favor of DiffServ functionality in the stub networks, especially on end hosts such as servers. Recent developments have clearly highlighted the need for diffserv-capable servers. Large multi-tiered Web sites frequently deploy frontend servers and devices to manage the flow of traffic into and out of the inner tiers. With the successful deployment of content distribution networks and services, ``edge´´ servers such as caching proxies are increasingly being placed at different points inside the network. This is in addition to the existence of more traditional proxies, e.g., application-level gateways/delays, SOCKS proxies, and voice call aggregation (H.323) proxies.

More often than not, proxies reside at the boundary of two separate networks, which are likely to represent different DiffServ domains. In general, such proxy servers need to consistently manage the traffic flowing through by enforcing bandwidth controls, preserving TOS markings, or translating QoS requirements (e.g., re-marking) if necessary. These developments strongly motivate the need to support the DiffServ architectural framework at end hosts - the sources and sinks of network connections. The DiffServ support in Linux [LINUX] is an effort in that direction.

1.2. Policy-Based Networking

As networking equipment becomes more complex in an effort to provide more complex QoS features, it is necessary to standardize the network configuration parameters used to establish a network policy. In the absence of such standards, it is generally not possible for networking equipment from different vendors to interoperate successfully, making realization of end-to-end QoS extremely difficult. Policy-based networking has grown out of the efforts of the Directory Enabled Networking (DEN) Ad Hoc working group [DEN] operating under the auspices of the Desktop Management Task Force (DMTF). The DMTF is an organization with much broader concerns of configuration and resources far beyond the network.

Policy-based networking enables administration of network policies from a centralized policy server, thus reducing some of the complexity surrounding network policy configuration for QoS. The underlying model behind network policies, along with an annotated list of the terminology, is discussed in [POLICY]. A policy schema for QoS being considered by the IETF´s policy working group is described in [SCHEMA]. We are concerned only with policy-based networking to the extent it can be used to define and manage policies that influence QoS.

In this draft we present a policy-based differentiated services architecture that has been implemented on the AIX operating system. This architecture satisfies the requirements outlined in [DSERVERS] and leverages policy-based networking to enable provisioning of DiffServ services via platform-independent, high-level policies. The goal of this Internet Draft is to describe the current support for Differentiated Services on AIX. We first summarize key requirements for provisioning of DiffServ functions on servers. We then present the policy-based architecture that provides DiffServ services to QoS-unaware (e.g., standard/existing) as well as QoS aware applications. The components constituting the architecture are described next. We report experimental results demonstrating the efficacy of the underlying DiffServ mechanisms and outline an ongoing effort to deploy diffserv-capable AIX servers on the Internet2 network.

2. Policy-Based DiffServ Architecture

We have realized a policy-based architecture for Differentiated Services on AIX. Our choice of such an architecture is motivated by the need to satisfy a number of essential requirements [DSERVERS]. We briefly describe these requirements before presenting the architecture.

2.1. Requirements

An architecture for diffserv enabled servers must satisfy the following requirements:

- It should allow standard/existing networked applications (that are QoS unaware) to be classified into a PHB other than best effort.

  Many existing networked applications, e.g. databases, transaction services, CICS, MQSeries, SAP etc., and middleware technologies (e.g., Java RMI, CORBA IIOP, HTTP) have been designed to run over best effort networks. For a staged deployment of DiffServ, these applications should not have to be modified to take advantage of the different classes of service.

Thus, an architecture for diffserv servers must support the ability to be application-unaware.

- It should allow classification at different granularities.

  The differentiated services architecture is based on classifying 5-tuples (the source and destination IP addresses, the source and destination port numbers, and the protocol) or 6-tuples (if we include the incoming DS field) into an outgoing DS field at each access router. While classification at this granularity is sufficient to meet may needs, classification should be possible at the granularity of an application, or the identity of a user.

- It should allow policy decisions to be applied uniformly across existing applications as well as new applications that signal the desired service level. A similar policy enforcement should occur for applications that use a signaled interface (e.g. RAPI) to signal their performance needs.

  Since both types of applications would require access to the same network resources, they should be controlled and administered by the same policy information.

- It should allow differentiated service support to coexist with secure communications using IP-sec or SSL.

  Secure communications using IP-sec or SSL is a reality in current Internet communications. Since IP-sec encrypts transport layer headers, the differentiated services architecture must perform its encryption functions prior to the IP-sec functions. SSL offers no hindrances as long as classification is based solely on contents of transport and network layer headers.

- It should allow functions needed in order to conform with the rate requirements enforced by network layer SLSs.

  The end-host must not violate the terms of the contracts negotiated with the network provider. It must put into place appropriate mechanisms to ensure that applications comply with network level SLSs.

- It must coexist with the support for integrated services in the end-host.

  Integrated Services and Differentiated Services offer a complementary set of QoS features, which may each be most appropriate to meet a different set of application requirements. Both methods should coexist in the end-host.

2.2. Architecture Overview

The DiffServ architecture on AIX enables server-based implementation of the core differentiated services requirements. Figure 2 shows the main components of the architecture that consists of the Policy Agent, the QoS Manager and the DiffServ API.

One of the key features of the architecture is that it supports policy based networking that can enable different vendors to inter-operate easily. With policy-based controls there is no need to modify existing applications, thereby, making application-unaware DiffServ support a reality. Most other implementations require the applications to request for a particular traffic profile.

The policy agent resides at the user-level and provides policy management while interfacing with the kernel-resident QoS Manager. It provides support for adding, deleting, and modifying policies that can either be defined by a local configuration file or can be retrieved from a centralized repository using LDAP. A policy specification consist of multiple rules and parameters including permissions (accept, deny), rate control parameters for the in-profile traffic, TOS marking values for in-profile and out-of-profile traffic, etc. A policy schema to describe policies in a standardized fashion has been defined in [POLICY]; the AIX policy agent conforms to these schema definitions. Along with the policy agent there exists command line support for applications or system administrators to directly set up rules, filters and classes.

The QoS Manager is a kernel-based module that implements the core DiffServ functionality. The two main functions of the QoS Manager are classification and traffic conditioning. The classifier determines the class a packet or connection belongs to based on its network 5-tuple and the instantiated filter specification. The traffic conditioning support includes packet marking, policing with drop or mark and traffic shaping based on the flow specification. The QoS Manager interacts with the transport layer of the stack to control the flow of outgoing traffic.

The interactions between the user level agents and the kernel-based QoS manager happens via different types of control messages over special control sockets. The messages include adding, deleting, and modifying traffic conditioning rules and filters.

For applications that are aware of DiffServ specifications, there is also a supported DiffServ API to communicate with the policy agent in order to set up the rules. Refer to [DSERVERS] for details.

The following section provides the implementation details on each of the components of the architecture, focusing on the Policy Agent and the QoS Manager.

```
                                      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   +------------+ +--------------+    ~  +--------+ Policy-Based Ctrl ~
   | QoS unaware| | QoS aware    |    ~  |Command |                   ~
   |Application | | Application  |    ~  |Line    |   +---------+     ~
   +------------+ +--------------+    ~  +--------+   |Policy   |     ~
           |       |     | control    ~       |       |Data Base|     ~
           | data  |     | path       ~  +-------+    | (LDAP)  |     ~
           |/ path\|  +------------+  ~  |       |<-->+---------+     ~
           |       |  |DiffServ API|<-~->|Policy |    +---------+---+ ~
           |       |  +------------+  ~  |Agent  |<-->|Local        | ~
           |       |                  ~  +-------+    |Configuration| ~
           |       |                  ~       |       +-------------+ ~
           |       |                  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
           |       |                          |
     +------------------------------------------------------------+
     |                  socket layer                              |
     +------------------------------------------------------------+
                  |                           |
                  |                           |
                +----------------+            |
                |Transport Layer |           +--------------------+
                | TCP / UDP      |<--------->|  QoS Manager       |
                +----------------+           |                    |
                  |                          | Rules - flow spec  |
                  |                          |         filter spec|
                +---------------+            | Actions-Marking    |
                |  IP           |            |         Policing   |
                +---------------+            |         Shaping    |
                  |                          +--------------------+
                  |
                  v
```

Figure 2. Architecture of DiffServ Enabled Server

3. Architectural Components

3.1. Policy Database and System Agent

The policy agent is a user level daemon provides policy management support by translating QoS policies and interfacing with the QoS kernel extension to install, modify, and delete policy rules. Policy rules may be defined in a local configuration file (/etc/policyd.conf), retrieved from a central policy database server using LDAP [LDAP], or created via command line programs (see Figure 2).

3.1.1. QoS Policy

The IETF´s policy working group is attempting to define a QoS schema in which QoS policy can be succinctly described such that the various QoS policy parameters have a standardized definition so that networking equipment supporting QoS functions can interoperate effectively and deterministically. We use the QoS schema proposed and described in [SCHEMA] to represent QoS policies.

As currently defined, a policy (or policy rule) consists of two components, a ***policy condition*** and a ***policy action***. The mechanisms enforcing the policy are in the form: if ***policy condition*** then ***policy action***. The policy condition component is essentially a traffic classifier (see section 3.2.1). The policy action specifies the permissions and restrictions that apply to the packets matched by the policy condition. A policy action contains specifications: permission (accept or deny), token bucket parameters defining in-profile traffic, TOS byte value for in-profile traffic, and TOS byte value for out-of-profile traffic. Refer to [SCHEMA] for more details on policy rules.

3.1.2. Policy Configuration

A policy expressed in the configuration file has two primary statements, ServiceCategories and ServicePolicyRule, that contain the attributes to express the diffserv policy. The ServiceCategories statement specifies the type of service that a flow of IP packets should receive as they traverse the network. The ServicePolicyRule statement specifies the characteristics of IP packets that are used to match to a corresponding service category, and receive the specified class of service. The policy agent can also be configured to translate diffserv policies from an LDAP database. The Policy Agent periodically updates policies from the configuration file or an LDAP server.

Policies can also be input from the command line via system commands. The AIX QoS implementation provides several system commands to manage and monitor policies [AIXDOC]. These commands ease the burden of managing policies for the QoS administrator and provide a way to monitor the amount of traffic matched with each installed policy. The available system commands for policy management are briefly described below.

1. Add Policy: The qosadd command adds the specified ServiceCategory or PolicyRule entry to the default policy configuration file /etc/policyd.conf and installs the changes in the QoS Manager kernel. The qosadd command communicates with the Policy Agent and specifies the action to add a policy. The flags on the qosadd command are passed in to the Policy Agent to specify the QoS attributes for the policy. The Policy Agent writes the appropriate policy out to the configuration file and then copies the new policy into the QoS Manager kernel for installation.

2. Remove Policy: The qosremove command removes the specified ServiceCategory or PolicyRule entry from the configuration file and removes the entry in the QoS Manager kernel. The qosremove command communicates with the Policy Agent and specifies the action to remove a policy. The flags on the qosremove command are passed in to the Policy Agent to specify the QoS ServiceCategory or PolicyRule to remove. The policy is removed from the configuration file and then the Policy Agent makes a call to remove the policy from the QoS Manager kernel.

3. Modify Policy: The qosmod command modifies the specified ServiceCategory or PolicyRule entry in the configuration file and installs the changes in the QoS Manager kernel. The qosmod command communicates with the Policy Agent and specifies the action to modify a policy. The flags on the qosmod command are passed into the Policy Agent to specify which ServiceCategory or PolicyRule to modify and to specify the attributes to modify. The Policy Agent modifies the configuration file and the makes a call to the QoS Manager kernel to modify the appropriate installed policies.

4. List Policy: The qoslist command lists the specified ServiceCategory or PolicyRule. The command will list all ServiceCategories or PolicyRules if none are specified. The qoslist command communicates with the Policy Agent and specifies the action to list a policy or list all policies. The flags on the qoslist command are used to specify the ServiceCategory or PolicyRule to list. The Policy Agent queries the list of installed policies and creates the appropriate output.

5. Display Policy Statistics: The qosstat command displays information about the installed QoS policies. The qosstat command communicates directly with the QoS Manager kernel to query the structures that maintain the QoS statistics. The structures in the QoS Manager kernel are updated dynamically upon the completion of each QoS action. Information such as the filter specification, flow class and data statistics are displayed.

3.1.3. Priorities for Overlapping Policies

The Policy agent contains a priority queue used for the installation of policies in the QoS Manager kernel. This functionality is important since the QoS Manager permits installation of overlapping policies. The priority queue in the Policy Agent can be utilized to determine the matching priority in the event of overlapping policies. The ServicePolicyRules statement in the policy definition allows the specification of the PolicyRulePriority attribute to set the priority of matching in the overlapping case.

3.2. QoS Manager Kernel Extension

The QoS Manager is a kernel module that implements the basic QoS mechanisms that can be used to deliver a particular quality of service to an IP flow or flow aggregate. It is used to realize both the IntServ and DiffServ models for QoS on AIX.

The QoS Manager is defined as a new protocol module in the networking subsystem, with a protocol switch table entry to represent the various functions it exports to other modules. Applications access the QoS Manager from user space by exchanging control messages over special control sockets, which are distinct from data sockets. A similar paradigm is used for passing routing information in 4.4 BSD. We modified the socket and transport layers to invoke the classification and policy enforcement functions exported by the QoS Manager before sending out packets. These functions are briefly discussed next.

3.2.1. Classification

The QoS Manager currently classifies data sockets based on the five tuple <source IP address, source port number, destination IP address, destination port number, protocol type>. In general, it can also classify data sockets based on other information available only at hosts, e.g., user id, group id, and application. Each installed policy maintains a list of data sockets that match the filter associated with the policy. This list is updated during connection establishment/teardown as well as policy creation/deletion.

3.2.2. Policy Enforcement

Policies are enforced using traffic conditioning mechanisms that actively regulate the flow of out-bound packets. The QoS Manager implements several traffic conditioning mechanisms: marking, policing, dropping, and shaping.

Policing

Policing monitors a specific flow to determine if it conforms to the traffic profile given by its token bucket specification. QoS Manager supports a token bucket specification with two parameters: the average rate and the burst size. Precisely what should be done with both in-and out-of-profile traffic is governed by policy. For example, out-of-profile packets could be dropped immediately, delayed until they conform (shaped), or delivered at a lower quality of service.

Dropping

Dropping is the simplest form of traffic conditioning where non-compliant packets are discarded.

Shaping

Shaping involves holding back out-of-profile packets until they conform to the specified traffic profile. QoS Manager implements shaping by either witholding packets in the socket send buffers (for TCP) or blocking the kernel (or application) thread shepherding the packet (for UDP). Applications sending data on non-blocking sockets are returned an error for out-of-profile traffic. To continue transmitting out-of-profile packets when they become compliant, QoS Manager relies upon a system shaping timer and other triggers (such as ACKs). The granularity of the shaping timer is configurable at user level, which allows it to be tuned for correct and efficient traffic shaping. QoS Manager provides max-min fairness between traffic originating on the different data sockets associated with (i.e., controlled by) a given policy.

Marking

Marking, which involves setting the TOS bits in the IP header for each outgoing packet, is implemented by updating the TOS byte in the protocol control block associated with a data socket. Each transport protocol automatically copies the newly generated marking to the outgoing packet´s header. When combined with policing, in-profile packets are marked with one DS codepoint while out-of-profile packets are marked with another, as specified in the policy.

4. Experimental Results

The goal of these experimental results is to demonstrate the efficacy of policy enforcement using the AIX diffserv architecture. The experiments determine if the traffic conditioning techniques used result in delivering the specified classes of service.

The test setup is chosen to represent how this implementation might be used in an enterprise LAN. The AIX 5.0 (beta version) QoS implementation is configured on an RS/6000 4-way SMP, with 166 MHz processors and a total physical memory of 512 MB. The server runs the publicly available Apache Web Server 3.1 with the appropriate diffserv policies configured by the policy agent. Client workload is generated by the publicly available Webstone 2.5 benchmark. There are four Webstone clients, each running on a RS/6000 200 MHz uniprocessor with 128 MB physical memory. The four clients are attached to a 100Mbps Ethernet switch with a 1 Gbps uplink to the server. Each Webstone client repeatedly makes 100 concurrent HTTP requests for a 100 KB file.

The following traffic conditioning techniques are exercised during the experiments.

- Marking is performed standalone to simply mark the IP TOS byte of each packet with a DS codepoint for further processing in external network routers.

- Policing is performed with marking so that the in-profile packets are marked with a DS codepoint to receive the desired class of service using the IP TOS field. The out-of-profile packets are left unmarked. The token bucket parameters of burst size and average rate are expressed in the policy to determine the traffic profile.

- Shaping is performed with marking to only transmit in-profile packets. The out-of-profile packets are held back until they are compliant with the traffic profile expressed in the QoS policy.

The shaping timer value is configurable via the no (network option) system command and is set at 10 milliseconds.

For the policing and shaping experiments the token bucket burst size is set to 60 KB.

Action	M	P	P	P	S	S	S
Rate	None	64Kbps	1.5Mpbs	45Mpbs	64Kbps	1.5Mbps	45Mbps
Thruput (total)	137	130	130	130	0.064	1.44	45.09
Thrhput (in-prof.)		0.064	1.5	45	0.064	1.44	45.09

M: Marking only
P: Policing with marking
S: Shaping with marking

Table 1: Throughput (Mbps) of Webstone Clients with Different QoS Policies

Table 1 shows the total throughput observed by Webstone clients for different traffic conditioning policies. The values presented are the total and in-profile throughput for: (i) marking only with no rate control, (ii) policing with 3 different bandwidth values, and (iii) shaping at 3 different values. The total throughput achieved with policing is slightly less than that of marking only due to the additional overhead of policing checks in the data path. The in-profile throughput conforms to the specified value.

Table 2 shows the commands for marking and policing. In case I, line 1 sets up marking with a DSCP value 00001000. The second line specifies that the policy applies to TCP traffic with source IP address 10.1.1.151 and source port 80. In case II, line 1 sets up a policing rule for 63 kbps with a burst size of 469 kb.

I. Policy for marking only:

```
qosadd -s 64k -t 00001000 service
qosadd -s 64k -r 64k-policy -n 6 -A 10.1.1.151-10.1.1.151 -P 80 policy
```

II. Policy for policing and marking:

```
qosadd -s 64k -t 00001000 -b 469 -m 63 service
qosadd -s 64k -r 64k-policy -n 6 -A 10.1.1.151-10.1.1.151 -P 80 policy
```

Table 2: AIX commands for setting up marking and policing rules

5. Deployment

We are collaborating with the International Center for Advanced Internet Research (ICAIR) to deploy the AIX policy-based diffserv functionality within the context of the I2DVN and QBONE projects of Internet2 [ICAIR]. The goal is to study network-intensive applications by performing end-to-end QoS experiments over a network providing differentiated services. The end points exist at the ICAIR Center at Northwestern University and IBM Hawthorne, NY. Once the Internet2 connections are configured for the end points, experiments will be conducted across the Abilene network connecting the IBM Hawthorne site and the ICAIR center.

The initial application is a streaming video application called VideoCharger installed on an RS/6000 SMP at each end-point. QoS policies are defined on an LDAP server at each end-point, and retrieved by the AIX Policy Agent for installation in the QoS Manager. Tests have been performed to assure proper interaction between the video application and the QoS Manager so that packets are properly marked with the desired DS codepoint. The application scenario being considered is one in which medium bandwidth video (1.5 Mbps MPEG-1) will be generated in unicast mode from a VideoCharger server to VideoCharger clients and to a special measurement client. The measurement client will be used to measure the latency and throughput between the server and clients. Those statistics will then be compared with the traffic profile expressed in the QoS Policy.

6. Security Considerations

We have highlighted a number of security considerations for servers supporting differentiated services in an earlier draft [DSERVERS].

Acknowledgments

The authors would like to acknowledge the helpful comments and suggestions of the following individuals: Brian Carpenter, Edward Ellesson, Ray Jennings, Dilip Kandlur, Arvind Krishna, and Ken White.

References

[AFREF] J Heinanen et. al., `` Assured Forwarding PHB Group´´, Internet Draft draft-ietf-diffserv-af-04.txt, January 1999, Work in progress.

[EFREF] V. Jacobson et. al, ``An Expedited Forwarding PHB´´, Internet Draft draft-ietf-diffserv-phb-ef-01.txt, November 1998, Work in progress.

[DIFFEDGE] Y. Bernet, D. Durham and F. Reichmeyer, ``Requirements of Diff-Serv Boundary Routers´´, Internet Draft draft-bernet-diffedge-01.txt, November 1998, Work in progress.

[DSARCH] S. Blake, et. al. "An Architecture for Differentiated Services", Internet RFC 2475, December 1998.

[DSFRAME] Y. Bernet, J. Binder, S. Blake, et. al. " A Framework for Differentiated Services", Internet Draft <draft-ietf-diffserv-framework-01.txt>, November 1998, Work in progress.

[DSHEAD] K. Nichols et. al., "Definition of the Differentiated Services Field (DS Byte) in the IPv4 and IPv6 Headers", Internet RFC 2474, December 1998.

[POLICY] J. Strassner and E. Ellesson, ``Policy Framework Core Information Model´´, IETF Draft draft-ietf-policy-core-schema-00.txt, November 1998, Work in progress.

[RSVPDIFF] Y. Bernet, R. Yavatkar et. al., ``A Framework for use of RSVP with DiffServ Networks´´, Internet Draft draft-ietf-diffserv-rsvp-01.txt, November 1998, Work in progress.

[DSERVERS] A. Mehra et. al., ``Architectural Considerations for DiffServ Servers´´, Internet Draft draft-mehra-diffserv-servers-01.txt, February 1999, Work in progress.

[LDAP] M. Wahl et. al, ``Lightweight Directory Access Protocol(v3)´´, Internet RFC 2251, December 1997.

[LINUX] Werner Almesberger et. al, ``Differentiated Services on Linux´´, Internet Draft draft-almesberger-wajhak-diffserv-linux-01.txt, June 1999, Work in progress.

[ICAIR] Brian Carpenter et. al, ``Experiments with Differentiated Services at ICAIR´´, Internet2 Workshop Presentation http://sss.advanced.org/meetings/houston2000, February 2000.

[AIXDOC] IBM Publications Austin, TX, ``AIX Version 5.0 System Management Guide:Communications and Networks´´, IBM Publication http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/topnav/topnav.htm, Future Release.

[SCHEMA] R. Rajan et. al, ``Schema for Differentiated Services and Integrated Services in Networks´´, Internet Draft draft-rajan-policy-qosschema-00.txt, October 1998, Work in progress.

[DEN] S. Judd and J. Strassner, ``Directory Enabled Networks - Information Model and Base Schema´´, Draft v3.0c5 DEN Specifications, September 1998.

Authors´ Address

Ashish Mehra Phone: (914) 784-7628
Dinesh Verma Phone: (914) 784-7466
Renu Tewari Phone: (914) 784-7154
Tsipora Barzilai Phone: (914) 697-8326
Douglas Freimuth Phone: (914) 784-6221
Mandis Beigi Phone: (914) 784-3277
IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
Email: mehraa,dverma,tewarir,tsipora,freimuth,mandis@watson.ibm.com