Distributed Edge-to-Edge QoS Monitoring Methods to Manage Traffic Flows in IP Networks Supporting Differentiated Services

October 22, 2001

Jae-Young Kim
jay@postech.ac.kr

Distributed Processing & Network Management Lab.
Dept. of Computer Science and Engineering
POSTECH
Outline

• Introduction
• DiffServ Framework and Its Limitations
• Monitoring Methods for Edge-to-Edge DiffServ Flows
• Applying Proposed Methods to Managing DiffServ Networks
• Edge-to-Edge DiffServ Flow Monitoring System
• Conclusions
Current Internet and QoS Support

• Simple but efficient IP technology
  – connectionless protocol
  – single FIFO queuing / best-effort packet forwarding
  – statistical multiplexing $\rightarrow$ efficient link utilization
  – shortest path routing $\rightarrow$ only one route at a time
  – IP over everything
  – Internet users and traffic volumes drastically increased

• No QoS support
  – same service quality for every packet
  – explosive traffic increase / burst traffic patterns
    $\rightarrow$ congestions at bottleneck points $\rightarrow$ unexpected delay / loss / jitter
  – no quality guarantee
Internet QoS Frameworks

• Need for supporting QoS
  – need for guaranteeing QoS
  – application characteristics change (elastic $\rightarrow$ QoS-sensitive)
  – users requirements / providers requirements
  – need to provide different service quality to different users & applications

• Current Internet QoS frameworks
  – answers for providing different QoS to different users & applications
    – IntServ : 1993, RSVP signaling, fine-grained, not scalable
    – DiffServ : 1997, no signaling, edge/core model, coarse-grained, scalable
    – MPLS : 1998, label switching, tool for traffic engineering, LSP, LDP

  – each QoS framework has pros and cons to each other
  – integration of above frameworks
QoS-enabled IP Network Model

QoS Domain

Access Network

Core Network

Neighbor QoS Domain

Peering Network

Peering Network

Neighbor QoS Domain

QoS Provisioning

QoS Classification

QoS Negotiation
Integration of Internet QoS Frameworks

- Access Network (IntServ)
- Access Network (IntServ)
- Access Network (IntServ)
- Access Network (IntServ)
- Backbone Network (DiffServ)
- Underlying Networks (MPLS)
Need for QoS Monitoring

- Dynamic nature of network status
  - traffic burst and congestion
  - dynamic routing change

- Need to keep up the QoS status change
  - snapshot current QoS status
  - ensure QoS provisioning
  - verify expected QoS
  - understand historical trends

- Need to have information for solving QoS problems
  - report QoS degradation
  - pinpoint QoS degradation locations
  - help to find possible solutions

- Few research work on QoS monitoring
  - most existing work is done on QoS configuration and provisioning
QoS Management Process Cycle

1. QoS Requirements
2. QoS Provisioning
3. Configuration
4. QoS Monitoring
5. QoS Status Analysis
6. Problem Detection
7. QoS Requirements
Motivation and Key Ideas

- **DiffServ QoS framework**
  - edge / core model
  - small number of QoS classes (maximum 64) for aggregated traffic
  - separation of QoS control and routing control

- **Edge-to-Edge QoS monitoring**
  - E-to-E QoS is useful for managing DiffServ domains in various ways
  - each DiffServ router can monitor QoS classes it handles
  - however, local QoS monitoring doesn’t provide E-to-E QoS status
  - how about combining routing and local QoS monitoring together?

- **Key ideas**
  - E-to-E QoS = routing topology + local QoS monitoring
  - local QoS monitoring (throughput, drop) at each DiffServ router
  - construct E-to-E QoS by combining local QoS and E-to-E routing path
  - modeling E-to-E DiffServ flow and hop-by-hop QoS concatenation rules
  - finer-grained DiffServ : enable different control on different E-to-E flows
    (number of supported QoS classes) X (number of edge routers)^2
Related Work

- **Basic IP traffic monitoring**
  - various low-level monitoring tools: tcpdump, ntop, ethereal, …
  - SNMP-based monitoring: MIB2, RMON (from IETF)
  - traffic flow monitoring: RTFM (RFC2720-2724), NetFlow (Cisco)
  - IP performance metric (IPPM): RFC2330

- **End-to-End QoS monitoring**
  - passive monitoring
    - Jiang et al. (University of Singapore), NetScope (AT&T Lab)
  - active probing
    - PINGer (Stanford IEPM), MBAC (Measurement-Based Admission Control)

- **DiffServ QoS management approach**
  - primitive DiffServ management: DSMIB, DSPIB, DSMON (from IETF)
  - Bandwidth Broker architecture (BB): IETF Internet draft, 1997
  - Per-Domain Behavior (PDB): IETF RFC3086
  - Resource Management in DiffServ (RMD) framework: IETF Internet draft, 2001
  - TEQUILA project: European Commission IST, 2000~2002
Outline

• Introduction
• **DiffServ Framework and Its Limitations**
  • Monitoring Methods for Edge-to-Edge DiffServ Flows
  • Applying Proposed Methods to Managing DiffServ Networks
• Edge-to-Edge DiffServ Flow Monitoring System
• Conclusions
Concepts of DiffServ Framework

- Edge / Core model
- Per-Hop Behavior (PHB)
- Limited number of QoS classes for aggregated traffic
Functional Blocks of a DiffServ Router

- **Management (SNMP/COPS)**
- **DiffServ Configuration and Monitoring Interface**
- **Ingress TCB**
- **Routing**
- **Egress TCB**
- **Queuing**

TCB: Traffic Conditioning Block
Traffic Conditioning Block

TCB

packet in

Classifier

Marker

Meter

Shaper Dropper

packet out
Conceptual Model of a TCB

Classifier

Multiple Priority Queues

Scheduler

packet in

packet out

classification rules

packet drop algorithms

scheduling algorithms
Why Monitoring E-to-E DiffServ QoS?

- Tool for deciding QoS provisioning parameters
  - it is hard to decide proper QoS parameters when provisioning DiffServ network
  - multiple ingress / egress situation results in complicated merge / split points
  - periodic monitoring results help to decide proper parameters

- Report detailed network status
  - E-to-E DiffServ QoS information provides detailed snapshot of network status
  - network operators can understand the QoS of the specific E-to-E connection
  - PDB can be checked by the E-to-E QoS information

- Pinpoint reasons of bottleneck situation
  - when bottleneck occur, the bandwidth-consuming E-to-E flow can be identified
  - this information can be used for traffic engineering decision

- Accounting and pricing
  - usage accounting needs E-to-E QoS information
  - pricing model can be built by the monitored information

- End-to-end QoS
  - edge-to-edge QoS can be a building block for end-to-end QoS
  - used for intradomain SLA negotiation
Key Ideas

- E-to-E DiffServ Flow Modeling
- Local QoS Monitoring
- Routing Information
- QoS Concatenation Methods

DiffServ Domain

Edge-to-Edge QoS (throughput / drop)

- Tool for Provisioning
- Detailed Network Status Report
- Bottleneck Detection and Resolution
- Finer-Grained DiffServ
Proposed Monitoring Architecture

- Edge-to-Edge Monitoring
- Per-Hop QoS Concatenation Rule / Method
  - Topology Data
  - Performance Data
    - SNMP-based Polling and Notification

Differentiated Services Networks
Outline

- Introduction
- DiffServ Framework and Its Limitations
- **Monitoring Methods for Edge-to-Edge DiffServ Flows**
  - Applying Proposed Methods to Managing DiffServ Networks
  - Edge-to-Edge DiffServ Flow Monitoring System
- Conclusions
Modeling Topology / Flow / E-to-E Flow

- Edge-to-Edge Flow of Class i at time T
- Flow of Class i at time T
- Topology at time T
Graphical Representation of DiffServ Nodes

- **Direct**: 1 ingress / 1 egress
- **Split**: 1 ingress / n egress
- **Merge**: m ingress / 1 egress
- **Switch**: m ingress / n egress

**Switch Node Elimination**

- m ingress / n egress
- virtual link
- m ingress
- n egress
Representation of an E-to-E DiffServ Flow

Es : source (ingress) edge router
Ed : destination (egress) edge router
Ri : core router
Pi : path link
Mk : merge link
Sj : split link

- each link has QoS information Q(Ci, Li), where Ci is an aggregate class and Li is a link
- Q(Ci, Li) contains throughput and drop rate of traffic in aggregate class Ci in the link Li within monitoring interval T
- In steady state (throughput from sources + throughput from merge links) equals to (throughput into sinks + throughput into split links + drops in transit)
Distributed QoS Monitoring and Concatenation

\[ E_s \xrightarrow{\text{flow}} R_1 \xrightarrow{} R_2 \xrightarrow{\ddots} R_i \xrightarrow{} \ldots \xrightarrow{} R_n \xrightarrow{} E_d \]

\[ Q_1 \xrightarrow{} Q_2 \xrightarrow{} \ldots \xrightarrow{} Q_i \xrightarrow{} \ldots \xrightarrow{} Q_n \]

QoS Concatenation

\[ Q_{e-to-e} \]
E-to-E QoS Concatenation Methods

• **Purpose**
  – methods for combining locally-observed QoS parameters
  – local QoS parameters + edge-to-edge routing path
  – simple and scalable methods

• **Two different methods**
  – edge-to-edge QoS with min/max bound
  – edge-to-edge QoS with ingress marking / egress counting

• **Two phases**
  – first phase: calculate edge-to-edge throughput (forward routing path)
  – second phase: calculate edge-to-edge drop by using BSR (backward routing path)

Bandwidth Share Ratio (BSR) = \[
\frac{\text{amount of edge-to-edge throughput}}{\text{amount of aggregated throughput of a link}}
\]
**Min/Max Bound for E-to-E DiffServ Flow**

- From the local QoS information, the max/min throughput of an edge-to-edge DiffServ flow can be obtained by following rules.

Max E-to-E Throughput = \( \min \{ \text{throughput at } P_i \} \)

Min E-to-E Throughput = \( \text{throughput at } P_1 - \sum (\text{throughput at } S_j) \)
QoS Concatenation Algorithm (Min/Max)

- First phase: calculate edge-to-edge throughput

\[
\text{TH}_{\text{max}} = \text{TH}_{\text{min}} = \text{throughput of } P_1
\]

\[
\text{for ( each path link, } P_i \text{ ) } \{
\]
\[
\text{BSR}_{\text{max}} = \text{TH}_{\text{max}} / \text{throughput of all merged links}
\]
\[
\text{DROP}_{\text{max}} = \text{BSR}_{\text{max}} \times D_i
\]
\[
\text{TH}_{\text{max}} = \text{TH}_{\text{max}} - \text{DROP}_{\text{max}}
\]

\[
\text{BSR}_{\text{min}} = \text{TH}_{\text{min}} / \text{throughput of all merged links}
\]
\[
\text{DROP}_{\text{min}} = \text{BSR}_{\text{min}} \times D_i
\]
\[
\text{TH}_{\text{min}} = \text{TH}_{\text{min}} - \text{DROP}_{\text{max}}
\]

\[
\text{TH}_{\text{max}} = \min( \text{TH}_{\text{max}}, \text{throughput of } P_{i+1} )
\]
\[
\text{if ( } R_i \text{ is a split node ) } \{
\]
\[
\text{TH}_{\text{min}} = \text{TH}_{\text{min}} - \text{throughput of all split links}
\]
\[
\} \text{ else } \{
\]
\[
\text{TH}_{\text{min}} = \min( \text{TH}_{\text{min}}, \text{throughput of } P_{i+1} )
\]
\[
\}
\]

\[
\text{final throughput}_{\text{max}} = \text{TH}_{\text{max}}
\]
\[
\text{final throughput}_{\text{min}} = \text{TH}_{\text{min}}
\]
QoS Concatenation Algorithm (Min/Max)

- Second phase: calculate edge-to-edge drop

\[
\begin{align*}
\text{TH\_max} &= \text{final\_throughput\_max} \\
\text{TH\_min} &= \text{final\_throughput\_min} \\

\text{for ( each path link, } P_i, \text{ in reverse order )} \{ \\
&\quad \text{BSR\_max} = \text{TH\_max} / \text{throughput of } P_i \\
&\quad \text{DROP\_max} = \text{BSR\_max} \times D_i \\
&\quad \text{TH\_max} = \text{TH\_max} + \text{DROP\_max} \\

&\quad \text{BSR\_min} = \text{TH\_min} / \text{throughput of } P_i \\
&\quad \text{DROP\_min} = \text{BSR\_min} \times D_i \\
&\quad \text{TH\_min} = \text{TH\_min} + \text{DROP\_min} \\
\}
\]

\[
\begin{align*}
\text{initial\_throughput\_max} &= \text{TH\_max} \\
\text{initial\_throughput\_min} &= \text{TH\_min}
\end{align*}
\]
Example of E-to-E DiffServ Flows

Example Topology

Example Cases

- Case 1: with no drop rate
  - there is no packet drop in every link, non-congested situation

- Case 2: with drop rate
  - some of links have packet drops due to congestion
Case 1: with no drop rate

Min Throughput

<table>
<thead>
<tr>
<th>Min Throughput</th>
<th>80</th>
<th>80</th>
<th>50</th>
<th>50</th>
<th>40</th>
<th>final minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min BSR</td>
<td>40/90</td>
<td>40/60</td>
<td>40/80</td>
<td>40/70</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Max Throughput

<table>
<thead>
<tr>
<th>Max Throughput</th>
<th>80</th>
<th>80</th>
<th>60</th>
<th>60</th>
<th>60</th>
<th>final maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max BSR</td>
<td>60/90</td>
<td>60/60</td>
<td>60/80</td>
<td>60/70</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case 2: with drop rate (1st phase)

<table>
<thead>
<tr>
<th></th>
<th>Min Throughput</th>
<th>Max Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>42</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>52</td>
</tr>
<tr>
<td>Min BSR</td>
<td>80/100</td>
<td>80/100</td>
</tr>
<tr>
<td></td>
<td>42/80</td>
<td>60/80</td>
</tr>
<tr>
<td>Drop at Link</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4/5 × 10</td>
<td>4/5 × 10</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>6/8 × 10</td>
</tr>
<tr>
<td>Total Drop for Min</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>Total Drop for Max</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>16</td>
</tr>
</tbody>
</table>
## Case 2: with drop rate (2nd phase)

<table>
<thead>
<tr>
<th></th>
<th>Initial Minimum</th>
<th>Initial Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min Throughput</strong></td>
<td>34, 31, 31, 27, 27</td>
<td>66, 59, 59, 52, 52</td>
</tr>
<tr>
<td><strong>Min BSR</strong></td>
<td>31/90, 27/70</td>
<td>59/90, 52/70</td>
</tr>
<tr>
<td><strong>Drop at Link</strong></td>
<td>0, 31/90 × 10, 0, 27/70 × 10, 0</td>
<td>0, 59/90 × 10, 0, 52/70 × 10, 0</td>
</tr>
<tr>
<td><strong>Total Drop for Min</strong></td>
<td>7, 7, 4, 4, 0</td>
<td>14, 14, 7, 7, 0</td>
</tr>
</tbody>
</table>
E-to-E QoS with Ingress Marking / Egress Counting
• If every ingress edge router can distinguish the amount of traffic to every egress edge router, we can monitor the status of real E-to-E flows in more detail.
Ingress Marking / Egress Counting (Cont.)

• Mechanism
  – additional marking in IP packet header
  – every ingress edge router marks its ID on packets leaving
  – every egress edge router checks the ID of incoming packets and counts the number of packets/bytes for each ingress edge
  – each egress edge router can build an E-to-E throughput matrix for E-to-E flows from every ingress edge router to the egress edge router
  – with this E-to-E throughput matrix, E-to-E throughput is easily obtained

• Benefits
  – easy and correct way to measure real E-to-E throughput
  – simple marking at ingress edge routers
  – affordable load of counting at egress edge routers
  – no modification at core routers
Ingress Marking / Egress Counting (Cont.)

- Implementation Consideration

1) Modified DS field
   - the number of current standard DSCP is only 21 of 64 available
   - if we use currently-unused 2-bit in DS field with a proper encoding rule, 8 edge ID can be incorporated in DS field
   - if we limit the number of service classes in a DiffServ domain, the number of available edge ID can be increased

2) Use of MPLS header
   - for general implementation with unlimited number of DiffServ classes and edge routers, MPLS shim header can be used
   - this needs MPLS capability in every DiffServ router
QoS Concatenation Algorithm (IM/EC)

- First phase

\[ \text{TH} = \text{edge-to-edge throughput calculated from the e-to-e matrix} \]

- Second phase

\[
\text{for ( each path link, } P_i, \text{ in reverse order ) } \{ \\
\text{BSR} = \frac{\text{TH}}{\text{throughput of } P_i} \\
\text{DROP} = \text{BSR} \times D_i \\
\text{TH} = \text{TH} + \text{DROP} \\
\} \\
\text{initial_throughput} = \text{TH}
\]
Case 1: with no packet drop

<table>
<thead>
<tr>
<th>Throughput</th>
<th>30</th>
<th>30</th>
<th>30</th>
<th>30</th>
<th>30</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSR</td>
<td>30/80</td>
<td>30/90</td>
<td>30/60</td>
<td>30/80</td>
<td>30/70</td>
<td></td>
</tr>
</tbody>
</table>

Throughput:
- From Es: 30

Diagram:
- Es → R1: 80 → 90
- R1 → R2: 90 → 60
- R2 → R3: 60 → 80
- R3 → R4: 80 → 70
- R4 → Ed: 70
Case 2: with packet drops

<table>
<thead>
<tr>
<th></th>
<th>Throughput</th>
<th>BSR</th>
<th>Drop at Link</th>
<th>Total Drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>50</td>
<td>64/80</td>
<td>7</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>44/80</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>44/50</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>38</td>
<td>38/60</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>30/40</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Initial throughput: 57

From Es: 30
Assumptions for the Monitoring Methods

- **Routing information**
  - each DiffServ router has a routing table
  - dynamic routing change is reflected on the routing table

- **Per-class statistics**
  - each DiffServ router keeps records of DiffServ traffic of each class
  - counter for each packet transmitted and dropped

- **Monitoring interval**
  - longer than edge-to-edge delay
  - shorter than routing change

- **Proportional drop**
  - drop probability of a connection should be proportional to the amount of traffic share of the total throughput
  - for calculating edge-to-edge drop rates by following the proposed methods
Simulation for Checking the Proportional Drop Characteristic at DiffServ RED Queue

- **Purpose**
  - to check if an RED queue has proportional drop characteristics
  - S. Floyd and V. Jacobson, IEEE/ACM Transactions on Networking, August 1993

- **Simulation**
  - two traffic sources are merged at an RED queue
  - change BSR of two sources and count the drop rates of each traffic flow
  - total amount of traffic flows is 1.2Mbps, while the bottleneck link capacity is 1Mbps
**RED Queue Drop Characteristics**

- **Drop Probability** vs. **Average Queue Occupancy**

- **minth**: minimum threshold
- **maxth**: maximum threshold
- **maxp**: never drop
- Guaranteed drop when queue occupancy reaches **maxth**
- Non-zero and increasing drop rate from **minth** to **maxth**

Graph shows a linear increase in drop probability as queue occupancy increases from **minth** to **maxth**, with guaranteed drop at **maxth**.
## Proportional Drop Simulation Result

<table>
<thead>
<tr>
<th>BSR</th>
<th>E1→E3 (Mbps)</th>
<th>E2→E3 (Mbps)</th>
<th>Total RED Drop (packet)</th>
<th>Drop from E1 (packet)</th>
<th>Drop from E2 (packet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:9</td>
<td>0.12</td>
<td>1.08</td>
<td>3238</td>
<td>325 (10.04%)</td>
<td>2913 (89.06%)</td>
</tr>
<tr>
<td>2:8</td>
<td>0.24</td>
<td>0.96</td>
<td>3219</td>
<td>649 (20.16%)</td>
<td>2570 (79.84%)</td>
</tr>
<tr>
<td>3:7</td>
<td>0.36</td>
<td>0.84</td>
<td>3246</td>
<td>956 (29.45%)</td>
<td>2290 (70.55%)</td>
</tr>
<tr>
<td>4:6</td>
<td>0.48</td>
<td>0.72</td>
<td>3219</td>
<td>1246 (38.71%)</td>
<td>1973 (61.29%)</td>
</tr>
<tr>
<td>5:5</td>
<td>0.6</td>
<td>0.6</td>
<td>3267</td>
<td>1658 (50.75%)</td>
<td>1609 (49.25%)</td>
</tr>
<tr>
<td>6:4</td>
<td>0.72</td>
<td>0.48</td>
<td>3229</td>
<td>1944 (60.20%)</td>
<td>1285 (39.80%)</td>
</tr>
<tr>
<td>7:3</td>
<td>0.84</td>
<td>0.36</td>
<td>3227</td>
<td>2271 (70.37%)</td>
<td>956 (29.63%)</td>
</tr>
<tr>
<td>8:2</td>
<td>0.96</td>
<td>0.24</td>
<td>3210</td>
<td>2581 (80.40%)</td>
<td>629 (19.60%)</td>
</tr>
<tr>
<td>9:1</td>
<td>1.08</td>
<td>0.12</td>
<td>3391</td>
<td>3065 (90.39%)</td>
<td>326 (9.61%)</td>
</tr>
</tbody>
</table>
- **Result**
  - the RED queue proportionally drops packets from different traffic sources
Outline

• Introduction
• DiffServ Framework and Its Limitations
• Monitoring Methods for Edge-to-Edge DiffServ Flows
• **Applying Proposed Methods to Managing DiffServ Networks**
  • Edge-to-Edge DiffServ Flow Monitoring System
  • Conclusions
Applications of Edge-to-Edge QoS in Managing DiffServ Networks

- Network Status Reporting
  - new information: E-to-E QoS status

- Dynamic Adaptive Provisioning
  - decide appropriate provisioning parameters periodically

- Bottleneck Detection and Resolution
  - pinpoint bottleneck link
  - search for bandwidth-consuming E-to-E flows

- Others
  - monitoring Per-Domain Behavior (PDB)
  - intradomain management
  - accounting / pricing
Simulation Model for Monitoring DiffServ E-to-E Flows

- Each source set has 5 independent CBR traffic generators (64bps ~ 1Mbps).
- There are 4 different E-to-E DiffServ flows (S1→D1, S1→D2, S2→D1, S2→D2)
- There is no drop in transit routing paths (non-congested situation)

- Example 1: similar amount of bandwidths are shared among four E-to-E flows
- Example 2: one dominant bandwidth-consuming E-to-E flow (S1→D2)
Ex. 1: R1 → R2 Aggregated Throughput
Edge-to-Edge Throughput (S1→D1)
Edge-to-Edge Throughput (S1→D2)
Edge-to-Edge Throughput (S2→D1)
Edge-to-Edge Throughput (S2→D2)

Statistics for S2→D2 Flow

Throughput (bps) vs Simulation Time (sec)
Ex. 2: R1 $\rightarrow$ R2 Aggregated Throughput
Edge-to-Edge Throughput (S1→D1)
Edge-to-Edge Throughput (S1→D2)

![Graph showing statistics for S1→D2 flow with lines for min, max, and actual throughput over time.

Simulation Time (sec) on the x-axis, ranging from 0 to 20.

Throughput (bps) on the y-axis, ranging from 1e+06 to 6e+06.

Graph lines represent:
- Red line with markers for min
- Pink line with markers for max
- Blue line with markers for actual

The graph peaks and dips over time, showing variations in throughput.
Edge-to-Edge Throughput (S2 → D1)
Edge-to-Edge Throughput (S2→D2)
Outline

• Introduction
• DiffServ Framework and Its Limitations
• Monitoring Methods for Edge-to-Edge DiffServ Flows
• Applying Proposed Methods to Managing DiffServ Networks
• **Edge-to-Edge DiffServ Flow Monitoring System**
• Conclusions
Design Architecture

Management Console

Web Browser

DiffServ Manager
Configuration Management
Mgmt. Database

Metering & Monitoring
DiffServ MIB

Flow Management
SNMP Manager

SNMP Stack

Web Server

Web Integration

Domain Manager

HTTP

Network Element

Set of DiffServ Routers

DiffServ Router
SNMP Stack
SNMP Agent
SNMP MIB

System APIs
Routing Core

Set of TCBs
Packet
IETF DiffServ MIB

- For managing DiffServ router in SNMP framework
  - being standardized
  - IETF Internet draft : Ver1 (July 1999) ~ Ver14 (October 2001)

<table>
<thead>
<tr>
<th>Element</th>
<th>Table Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifier</td>
<td>Classifier</td>
<td>general classification parameters</td>
</tr>
<tr>
<td></td>
<td>SixTupleClfr</td>
<td>5-tuple information + DSCP value</td>
</tr>
<tr>
<td>Meter</td>
<td>Meter</td>
<td>general metering parameters</td>
</tr>
<tr>
<td></td>
<td>TBMeter</td>
<td>token bucket meter</td>
</tr>
<tr>
<td>Action</td>
<td>Action</td>
<td>general action parameters</td>
</tr>
<tr>
<td></td>
<td>MarkAct</td>
<td>marker action</td>
</tr>
<tr>
<td></td>
<td>CountAct</td>
<td>counter action</td>
</tr>
<tr>
<td></td>
<td>AbsoluteDrop</td>
<td>absolute drop action</td>
</tr>
<tr>
<td>Queue</td>
<td>AlgDrop</td>
<td>algorithmic dropper parameters</td>
</tr>
<tr>
<td></td>
<td>RandomDrop</td>
<td>random dropper parameters</td>
</tr>
<tr>
<td></td>
<td>Queue</td>
<td>queuing parameters</td>
</tr>
<tr>
<td></td>
<td>Scheduler</td>
<td>scheduling parameters</td>
</tr>
</tbody>
</table>
Detailed Architecture of DiffServ MIB

1. **SixTuple**
   - Filter

2. **Classifier**
   - Next

3. **Meter**
   - Success/fail
   - Meter specific

4. **Scheduler**
   - Next

5. **Queue**
   - Next

6. **Algorithmic Dropper**
   - Qmeasure
   - Alg-drop specific

7. **Action**
   - Success/fail
   - Next
   - Action specific

8. **TCB Pointer**

9. **Mark**

10. **Count**

11. **Absolute drop**

12. **Random drop**

---

60/74
Edge-to-Edge DiffServ Flow Construction

Input:
- Source IP
- Destination IP
- DSCP

MIB II:
- ifTable
- ipAddrTable
- ipRouteTable

Performance Analyzer:
- Classifier
- Meter
- Action
- Queue

E-to-E Monitoring Information:
- Meter: metering result
- Count: # of packets, # of octets
- Drop: drop rate, # of drops
- Throughput: min rate, max rate

QoS Concatenation Rules

Topology Information:
- Egress TCB
- Source Edge Router
- Core Routers
- Ingress TCB
- Destination Edge Router

Linked List of DiffServ Routers of A DiffServ Flow Path
Implementation Architecture

Management Console (Windows)
- GUI (Swing set)
  - performance viewer
  - topology viewer
  - event reporting
  - communication module

TCP socket

Central Manager (Linux)
- console manager
  - topology generator
  - resource manager
  - configuration manager
  - trap handler
  - polling engine
  - logging

SNMP stack

Network Element (Linux)
- SNMP agent
  - MIB II
  - DS MIB
  - NETLINK socket
  - DS-8 patch
  - ATM patch
  - Forwarding engine
  - packet in
  - packet out

UCD-SNMP library
C(gcc)
PostgreSQL

Java 1.3 + Kawa 3.5
SNMP Agent Architecture in Linux-based DiffServ Router

Linux DiffServ Router

User Space

UCD SNMP 4.1.2
SNMP Protocol Stack

MIB II
UCD-SNMP Built-in

DiffServ MIB
UCD-SNMP Extension

DiffServ Agent
TC-to-MIB Mapper
Netlink Parameter Retriever

NETLINK APIs

NETLINK Socket

DiffServ Extension (DS-8 patch)

Linux Traffic Controller

Kernel Space (Linux 2.3.40)

Packets
Implementation on a Linux-based Testbed

Linux-based DiffServ Testbed

QoS agent

QoS

QoS agent

QoS

QoS agent

QoS

QoS agent

QoS

SNMP-based Polling / Trap

DiffServ domain flow manager

console viewer
DiffServ Testbed Configuration

campus backbone

82.102
182.2

R1
(450MHz)

S1
(450MHz)

182.3

186.2
185.1

campus backbone

82.44
184.1

D1
(233MHz)

S2
(133MHz)

186.1

183.2

R2
(300MHz)

185.2

184.2

D2
(133MHz)

183.1

ATM

Ethernet
DiffServ Domain Flow Monitoring System
DiffServ Domain Flow Monitoring System
DiffServ Domain Flow Monitoring System
DiffServ Domain Flow Monitoring System

- Final Throughput (Measured) : 306
- Average Throughput (Calculated) : 426
- Drop Rate : 69.88188976377953 %
Outline

- Introduction
- DiffServ Framework and Its Limitations
- Monitoring Methods for Edge-to-Edge DiffServ Flows
- Applying Proposed Methods to Managing DiffServ Networks
- Edge-to-Edge DiffServ Flow Monitoring System
- Conclusions
Summary

- Need for Internet QoS
  - Internet explosion / QoS requirements
  - Internet QoS frameworks are suggested
- DiffServ and its limitations
  - DiffServ is a scalable backbone solutions in IP QoS frameworks
  - detailed management framework is needed
  - QoS monitoring is one of important requirements
- Edge-to-edge QoS monitoring of DiffServ flows
  - modeling edge-to-edge DiffServ flows
  - routing information + local QoS measurements → edge-to-edge QoS
  - two monitoring methods
  - applications of edge-to-edge QoS monitoring
- Developing an edge-to-edge DiffServ flow monitoring system
  - necessary when DiffServ is deployed in IP backbones
  - managing DiffServ routers in SNMP framework is suggested
  - working and testing on a Linux-based testbed
Contributions

- Various applications of edge-to-edge DiffServ flows
  - a useful building block for sophisticated management functions
  - configuration and provisioning
  - end-to-end traffic performance and QoS monitoring
  - traffic path controlling according to dynamic load changes
  - accounting and billing according to SLA’s

- Realization of an SNMP-based DiffServ flow monitoring system
  - guide for similar system development
  - SNMP-based DiffServ QoS monitoring agent with DSMIB

- Finer-grained DiffServ
  - enable different controls on different ingress/egress DiffServ flows
  - efficient and scalable monitoring methods for more flexible QoS control
Future Work

- Complete QoS management framework
  - integration with the policy framework (interworking with COPS/PIB)
  - feedback to reconfiguration

- Extend to various high-level management functions
  - service level agreements monitoring
  - accounting / billing
  - interdomain QoS negotiation

- Bidirectional edge-to-edge DiffServ flows
  - network service is provided with bidirectional communications
  - how to model and monitor bidirectional edge-to-edge DiffServ flows
Thank You.