A Method of Software-Hardware Integration for QoS Policy Combination in Gigabit Routers

Yasusi Kanada[*], Takeki Yazaki[†]

[*†]IP Network Research Center, Research & Development Group, Hitachi, Ltd.
[*]Totsuka-ku Yoshida-cho 292, Yokohama 244-0817, Japan
[*]kanada@crl.hitachi.co.jp, [†]yazaki@crl.hitachi.co.jp

Abstract: *In policy-based networks, two or more policies often have to cooperate because combined and customized network functions must be controlled using policies. Two types of policy transformation, policy fusion and policy division, are sometimes required to implement cooperating policy systems on high-performance hardware routers. Policy fusion transforms two or more policies into one, and policy division transforms a policy into two or more policies. These transformations causes a problem that the original policies must usually be strongly constrained to allow these transformations. This paper shows a method for resolving restrictions on the division of QoS policies by a software-hardware integration, i.e., by implementing virtual flow labels (flow IDs) in hardware and by dividing a policy and deploying the policies onto two filter blocks. We have developed a policy agent (PEP) and a gigabit router integrated by using this method. Both high-performance and flexibility are achieved by this integration.*

Keywords: *Policy-based networking, Diffserv (Differentiated services), Policy division, Policy fusion, Program transformation, Virtual flow label, Hardware router, Flow aggregation.*

1. Introduction

Network services will be demanded to be customizable by both customers and service providers. Both business customers and providers require customized services because the difference from competitors caused by the customization is the key to success in their business. Customization of networks is realized by using active-network and policy-based networking technologies. Novel functions are added to network devices by active network technologies, and they are controlled and managed by using policy-based networking technologies. Active network technologies have been widely studied and now portions of them are going to be in standardization process, e.g., proxy services in content delivery networks (CDN) [Tom 01]. Policy-based networking has also been widely studied and several new standards for policy-based networking are being developed in the IETF (Internet Engineering Task Force, e.g., the core and QoS policy information models [Moo 01, Sni 00], COPS (Common Open Policy Services) protocol [Dur 00], and COPS usage for COPS-PR [Cha 01].

In active networks, new network functions are organized into a component-based architecture. Components are selected, combined and added by a provider or customer. So the policies to control these functions should also be organized into a flexible component-based architectures [Kan 00a, Kan 00b]. A higher-level policy should be built by using lower-level policy components. For example, to build and to control a Diffserv (Differentiated Services) network [Ber 99], there must be at least two functions; classification and policing function that classifies and polices (i.e., limited to a certain bandwidth) packets come from service subscribers at an edge router, and queuing and scheduling function that queues and schedules the flow passes through. So a policy for classification and policing and a policy for queuing and scheduling should be composed into a higher-level Diffserv policy. If the service is typical Diffserv, the policy for classification specifies the class or the DSCP (Diffserv Code Points) [Nic 98] of the flow, and the policy for queuing/scheduling specifies the testing of the DSCPs to determine the algorithms and parameters for queuing and scheduling required by packets in that class. These policies can be regarded as components of a network-wide QoS policy. Although DSCPs can be used for implicit cooperation of policies in certain services, other services may require explicit specification of policy combination. A higher-level function or policy within a network is applied as a combination of lower-level functions or component policies.

As policy systems become more and more flexible, it will be increasingly difficult to translate policies at network nodes. The higher-level (device-independent) policies of the policy system must be automatically translated into lower-level (possibly device-dependent) policies that the network nodes can accept. A policy server, or a proxy that mediates between the policy server and the network devices, must mechanically translate the policies into configurations that suit devices from multiple vendors. Policy servers may thus have to translate multiple component policies that operate in cooperation into such configurations. Although this policy translation is comparable to program compilation, the process may, in fact, be much more complicated. A higher-level policy may have to be translated into two or more lower-level policies, while two or more higher-level policies may also have to be translated into one lower-level policy. For example, a higher-level policy may contain two (cooperating) functions, but there may be no lower-level policy that contains both functions. In this case, the policy must be divided in two; otherwise, its deployment to devices is not possible. The translation can thus be regarded as a type of program transformation [Par 83] and an author has discussed this [Kan 01b].

One reason that such non-straightforward correspondences exist is that the functions of devices, especially core routers, have become less flexible because they are now built with special-purpose hardware in order to meet performance requirements. In the 1970s, routers were general-purpose computers. Today, however, high-end routers include hardware-implemented routing engines and some of them include hardware-implemented QoS and MPLS (Multi-Protocol Label Switching) mechanisms. Programming of such routers is not as flexible as programming of general-purpose computers. Therefore, a complicated transformation algorithm may be required for such devices.

Sometimes, many restrictions have to be placed on the original policies to make transformations of the above type possible [Kan 01b]. In most practical cases, however, restrictions can be avoided without performance degradation by elaborate hardware design. In addition, even if it is not possible to avoid a complex transformation, requirements for the restrictions of high-level policies so that transformations can be eliminated by introducing virtual flow labels (VFLs) [Kan 01b]. The introduction of VFLs resolves indeterminacies of data dependence [Kan 01a] between policy rules. The VFL can be implemented in software in routers. However, a software implementation would not result sufficient performance.

This paper outlines a method of software-hardware integration for Diffserv policies; i.e., describes QoS architecture with hardware-implemented VFLs for gigabit routers and the policy transformation method. Section 2 outlines policy fusion and division. Section 3 explains the restriction of policy division and the method for resolving them by using VFLs, which was originally introduced in a previous paper [Kan 01b]. Section 4 explains an integration method of router hardware and policy agent software for Diffserv policies. This method has been implemented in a router. Section 5 shows performance evaluation results, and Section 6 concludes this paper.

2. Outline of Policy Fusion and Division

This section explains the fusion and division of policies, which were originally defined in an earlier work by an author [Kan 01b].

2.1 Definitions

A policy is a list of policy rules, which are if-then rules (condition-and-action rules). Transformations of two basic types are involved in translating higher-level policies to lower-level policies.

1. *Policy fusion*: If two or more higher-level policies are transformed into a lower-level policy, the transformation is called a policy fusion.
2. *Policy division*: If a higher-level policy is transformed into two or more lower-level policies, the transformation is called a policy division.

A transformation may be a combination of policy division and policy fusion; i.e., one set of higher-level policies may be transformed into a set of lower-level policies, while the functions of one of the higher-level policies are separated into two or more lower-level policies and the functions of two or more of the higher-level policies are merged into one lower-level policy.

2.2 An example of policy fusion

An example of policy fusion is given here. The original policies that police, mark, and queue packets in a flow are as follows:

```
E1 = { if (Source_IP is 192.168.1.1) {
           if (Information_Rate <= 1 Mbps) {
               DSCP = "EF";    -- Only 1 Mbps of the
                               -- traffic is marked as "EF".
           } else {    absolute_drop;
                       -- The overflow of traffic is discarded.
           }; } },
C1 = { if (DSCP is "EF") {
           Scheduling_Algorithm = "Priority";
                       -- Priority queuing is specified.
           Priority = "High"; } }.
```

Policies **E1** and **C1** each consist of a single rule for the sake of simplicity. The rule in **E1** monitors flow from IP address 192.168.1.1. When the information rate of the traffic exceeds 1 Mbps, the overflow is discarded. The DS fields of the packets that are passed are marked "EF" (expedited forwarding) [Jac 99]. The single rule in **C1** specifies priority queuing as the scheduling algorithm and sets the priority "High" for traffic marked "EF".

These policies are fused into policy **EC1**.

```
EC1 = { if (Source_IP is 192.168.1.1) {
           Scheduling_Algorithm = "Priority";
           Priority = "High";
           if (Information_Rate <= 1 Mbps) {
               DSCP = "EF";
           } else {    absolute_drop; }; }    }.
```

Policy **EC1** also consists of a single rule. The function of this policy is the same as that of **E1** and **C1** in combination. The major purpose of this policy fusion is performance improvement. If the two separate policies that express **E1** and **C1** are implemented in the router, and can be sequentially executed, this policy fusion is not necessary. However, the sequential execution of two policies may take more time than the execution of the fused policy, and the result will then be worsened the performance. Policy fusion then maximizes the performance.

2.3 Examples of policy division

Two examples of policy division are given. The first example is

```
EC2 = { if (Source_IP is 192.168.1.1) {
           DSCP = "EF"; Priority = "High"; },
        if (Source_IP is 192.168.1.3) {
           DSCP = "BE"; Priority = "Low"; } }.
```

The first rule in policy **EC2** marks "EF" on packets come from IP address 192.168.1.1 and queues them into a high priority queue. The second rule marks "BE" on packets come from IP address 192.168.1.3 and queues them into a low priority queue. This policy can be divided into the following two policies.

```
E2 = { if (Source_IP is 192.168.1.1) { DSCP = "EF"; },
       if (Source_IP is 192.168.1.3) { DSCP = "BE"; } }.
C2 = { if (Source_IP is 192.168.1.1) { Priority = "High"; },
       if (Source_IP is 192.168.1.3) { Priority = "Low"; } }.
```

Policy **E2** marks the packets and policy **C2** queues them into high or low priority queues. The list of conditions in a policy forms a classifier [Ber 99]. In **EC2**, the classifier is

{Source_IP is 192.168.1.1, Source_IP is 192.168.1.3}.

The classifier in the original policy **EC2** is copied exactly to the policies that result from the division, **E2** and **C2**. This is required to preserve the semantics of policies [Kan 01b].

The second example is shown. The original policy **E2** contains a single rule that including marking and metering actions:

```
E3 = { if (Source_IP is 192.168.1.1 ||
          Source_IP is 192.168.1.3) {
            -- Two flows from IP address 192.168.1.1 and
            -- 192.168.1.3 are aggregated.
            if (Information_Rate <= 1 Mbps) {
                DSCP = "EF";
            } else {    absolute_drop; }; }    }.
```

The single rule of this policy detects flows from IP addresses 192.168.1.1 and 192.168.1.3, meters the aggregated traffic, and marks or drops some of its packets.

A router, especially a hardware router, may require that two rules be used to detects the flows from the two IP addresses, i.e., it may not be possible to use a logical-OR in a condition of any single rule. This policy must then be divided into two policies as follows:

```
E31 = { if (Source_IP is 192.168.1.1) { DSCP = "EF"; },
        if (Source_IP is 192.168.1.3) { DSCP = "EF"; } },
E32 = { if (DSCP is "EF") {
          if (Information_Rate > 1 Mbps) { absolute_drop; };
        } }.
```

Traffic from both IP addresses, 192.168.1.1 and 192.168.1.3, is marked as "EF". The aggregated traffic is thus detected by the condition "DSCP is "EF"".[1] The classifier is not copied in this transformation. So this transformation can be used only when a certain condition holds; i.e., the resulted policies do not work correctly if there is a flow premarked "EF" (see Section 3.1).

A policy division is required here because the flow must be aggregated before the metering. If, in contrast, **E3** is translated into the following rule, the result of metering is different. The following single policy is an example of incorrect translation:

```
E31' = { if (Source_IP is 192.168.1.1) {
            if (Information_Rate <= 1 Mbps) {
                DSCP = "EF";
            } else {    absolute_drop; }; },
         if (Source_IP is 192.168.1.3) {
            if (Information_Rate <= 1 Mbps) {
                DSCP = "EF";
            } else {    absolute_drop; }; }    }.
```

The rules in this policy separately meters the two flows. The metered result is thus different from the result for the original policy.

3. Restrictions on Policy Division and the Use of VFLs to Resolve Them

This section describes restrictions on policy division, and a method of resolving restrictions by using VFLs. There are restrictions on policy fusion too [Kan 01b], but they are not covered in this paper because its focus is on the usage of VFLs (See Section 3.2).

3.1 Restrictions

The restrictions on policy division are explained here using three examples. The first example on a restriction on DSCP reference

[1] There is a restriction in this transformation explained in Section 3.1.

and marking follows. In policy division, restrictions may be created by marking. A policy **P** is assumed to be divided into **P1** and **P2**, with **P2** applied after **P1**. For example, **P**, **P1** and **P2** may be as follows:

```
P = { p1:  if (DSCP is 10) {  -- Testing premarked traffic.
                if (Information_Rate > 1 Mbps) {
                    absolute_drop; };  },
    p2:  if (Source_IP is 192.168.2.1) { DSCP = 10; } },
P1 = { p11: if (DSCP is 10) { },
    p12: if (Source_IP is 192.168.2.1) { DSCP = 10;  } },
P2 = { p21: if (DSCP is 10) {
                if (Information_Rate > 1 Mbps) {
                    absolute_drop; };  },
                -- This rule is wrongly applied to traffic
                -- that rule p12 has been applied.
    P22: if (Source_IP is 192.168.2.1) { }          }.
```

In policy P, *p1* detects packets with a DSCP of 10 and polices that flow. So rule *p2* tests whether the source IP address is 192.168.2.1, and then sets DSCP 10 to packets from that address. The condition part of rule *p1* of **P** tests a DSCP and the action part of rule *p2* marks the same DSCP. Rule *p21* of **P2**, which corresponds to rule *p1*, may wrongly passes traffic with DSCPs that were marked by rule *p12*. This problem is caused by copying the classifier. The classifier in **P**, i.e., {DSCP is 10, Source_IP is 192.168.2.1}, is exactly copied to **P1** and **P2**. However, this causes a conflict with an action, DSCP = 10, and causes a wrong result.

The second example on a restriction on DSCP reference and *re*-marking follows. Policy **Q** is assumed to be divided into **Q1** and **Q2**, with **Q2** applied after **Q1**.

```
Q  = { q: if (DSCP is 10) {
            DSCP = 14;
            if (Information_Rate > 1 Mbps) { absolute_drop; };
       } },
Q1 = { q1: if (DSCP is 10) { DSCP = 14; }  },
Q2 = { q2: if (DSCP is 10) {
            if (Information_Rate > 1 Mbps) { absolute_drop; };
       } }.
```

If rule *q* of **Q** tests one DSCP then marks another DSCP, rule *q2* of **Q2**, which corresponds to rule *q*, may wrongly *not* pass traffic with DSCP 14 by rule *q1*. The problem is also caused by copying of the classifier;[1] i.e., rules *q1* and *q2* have exactly the same conditions as *q*.

The third example on a flow aggregation restriction is explained using policy **E3** given in Section 2.3. The results of the policy division are **E31** and **E32**. These policies works correctly when there is no flow premarked "EF". However, if there is a flow that has DSCP "EF" and the source IP address is neither 192.168.1.1 nor 192.168.1.3, this flow is wrongly caught by the rule in **E32**. A flow that is not caught by any rule in a policy is called a *default flow* [Kan 01b]. Nonexistence of default flows is required in this transformation. An easy method that assures nonexistence of default flows is to add the following rule to the end of the policy:

```
if (true) { absolute_drop; };
```

The restrictions described here could be removed by designing the router hardware so that there would be no need of policy division. However, it will complicate the hardware and may reduce the performance.

3.2 Virtual flow labels and elimination of restrictions

The restrictions are caused because of copying of a classifiers. So they can be eliminated by developing a method of policy division that eliminates this copying. Introducing a VFL enables this.

A *virtual flow label* (VFL), or virtual label [Kan 00b], is a label attached to a packet or flow, and is similar to a DSCP, except that the DSCP is a *real label*. However, a VFL is external to the packet, and the number of different VFLs is not restricted. A VFL is not conveyed by packets themselves, so the policies to cooperate must exist in the same network node unless the VFL value is conveyed by some other means, such as wavelength, physical location, and so on.

All the example policies shown in Section 2.3 can be correctly and systematically divided using VFLs. In the first example, policy **P** can be divided as follows:

```
P1' = { p11': if (DSCP is 10) { VFL = "p11"; },
        p12': if (Source_IP is 192.168.2.1) {
                  DSCP = 10; VFL = "p12";  } },
P2' = { p21': if (VFL is "p11") {
                  if (Information_Rate > 1 Mbps) {
                      absolute_drop; };  },
        p22': if (VFL is "p12") { }       }.
```

Rules *p11'* and *p21'* are derived from rule *p1*, and rules *p12'* and *p22'* are derived from rule *p2*. Rule *p21'* catches the traffic passed through *p11'* and rule *p22'* catches the traffic passed through *p12'* because of VFLs "p11" and "p12".

In the second example, policy **Q** can be divided as follows:

```
Q1' = { q1': if (DSCP is 10) { DSCP = 14; VFL = "q11"; } },
Q2' = { q2': if (VFL is "q11") {
                  if (Information_Rate > 1 Mbps) {
                      absolute_drop; };  }          }.
```

Rules *q1'* and *q2'* are derived from rule *q*. In rules *p21'*, *p22'*, and *q2*' of these policies, the original conditions are replaced by the testing of the VFL. Rule *q2'* catches the traffic passed through *q1'* because of VFL "q11".

In the third example, policy **E3** can be divided as follows:

```
E31" = {if (Source_IP is 192.168.1.1) {
            DSCP = "EF"; VFL = "ef"; },
        if (Source_IP is 192.168.1.3) {
            DSCP = "EF"; VFL = "ef"; }  },
E32" = {if (VFL is "ef") {
          if (Information_Rate > 1 Mbps) { absolute_drop; };
        } }.
```

The only rule in policy **E32"** correctly catches the flows passed through one of the two rules in policy **E31"**. It never catches a default flow because no default flow has VFL "ef" (because "ef" is only marked in the rules in **E31"**).

The introduction of a VFL not only eliminates the restrictions but may also reduces the cost of classification. For example, a 32-bit IP-address is compared in the condition part of rule *p22*, but this operation is replaced by a comparison of a VFL "p12" (in rule *p22'*), which can be much shorter than 32 bit, and thus the performance can be improved by using high-speed indexing operation when there are many such rules. However, to use the above policy division method, VFLs must be implemented in the router either by software or hardware.

4. Method of Software-Hardware Integration for Policy-based QoS

This section explains the hardware-software integration method for applying Diffserv policies to high performance routers.

4.1 Diffserv policies and their deployments in PolicyXpert

The Diffserv policies in OpenView PolicyXpert and JP1/PolicyXpert [HP 00][2] and the method of deploying them to networks are reviewed here. PolicyXpert currently supports provi-

[1] One way of solving this problem is to rewrite the DSCP in the condition of *q2* from 10 to 14: if (DSCP is 14) { … }. This works because *q1* rewrites DSCP of packets with a DSCP from 10 to 14. However, this method does not always generate a correct result.

[2] PolicyXpert is a trademark of Hewlett-Packard Company. The second version of PolicyXpert was developed jointly by Hewlett-Packard Company and Hitachi, Ltd.

sioning and RSVP-based QoS policies, including Diffserv policies of three types: Traffic Classifier Policies (CL Policies), Traffic Conditioner Policies (TC Policies), and Queue Control Policies (QC Policies) [Kan 01c, HP 00]. Most Diffserv functions, i.e., flow classification, marking, policing, queuing, scheduling, shaping, and so on, and some other QoS function, e.g., Intserv function, can be represented by using TC and QC Policies. CL Policy adds flexibility in representation and usage (operation) [Kan 01d]. TC and QC Policies are natural representation of Diffserv functions as policies, i.e., collections of condition-action rules. TC Policies represents edge functions and QC Policies represents core functions. In a Diffserv network domain, a TC Policy is usually applied to the inbound traffic at an edge interface (i.e., an edge router interface to outside the netwrok), and a QC Policy is usually applied to the outbound traffic at core interfaces (i.e., interfaces between routers in the network). However, TC and QC Policies (and CL Policies too) can be combined at an interface (usually at a core interface of an edge/boundary router).

The policies are briefly explained below. The CL Policy matches a particular flow and places a VFL called a Classifier Identifier (CID) on the flow. An example of a CL Policy rule is:

```
if (Source_IP is 192.168.3.1) { CID = "User1"; }.
```

This rule attaches a CID "User1" to the flow. Because attaching a CID is the only function of a CL Policy, it is always used as a component of a larger policy.

The TC Policy matches and may police (i.e., limit traffic), a flow and marks the DSCPs on the packets, or absolutely (unconditionally; i.e., discards all packets in the flow) drops the traffic. An example of a TC Policy rule is

```
if (CID is "User1") {
    if (Information_Rate <= 1 Mbps) {
            DSCP = 10;    -- The first 1 Mbps of the traffic
                          -- is marked with DSCP 10.
    } else {  absolute_drop; };     -- The rest of the traffic is
}.                                  -- unconditionally dropped.
```

This rule marks the packets with either of two DSCP values or drops them with a CID value of "User1".

The QC Policy matches a flow and queues the packets in the flow. It may schedule and/or shape the traffic, and may algorithmically drop the packets (i.e., drop the packets according to a random or deterministic algorithm). The scheduling algorithm can be selected from among strict priority queuing (S-PQ), bounded priority queuing (B-PQ), aggregate bandwidth scheduling (A-BW), or per-flow bandwidth scheduling (P-BW). If the algorithm is A-BW or P-BW, minimum and/or maximum (shaping) rates can be specified for the traffic flow. An example of a QC Policy rule is:

```
if (DSCP is [10, 14, 18]) {  -- if DSCP is 10, 14, or 18
    Scheduling_Algorithm = "S-PQ";
    Max_Queue_Size = 200 packets;
    Discard_Algorithm = "Deterministic Discard";
    if (DSCP is 18) {
        Discard_Level = 100%;        -- 200 packets
    } elsif (DSCP is 14) {
        Discard_Level = 70%;         -- 140 packets
    } else {
        Discard_Level = 40%; };  }.  -- 80 packets
```

A QC Policy rule represents a scheduling queue. This rule specifies deterministic discard as its discard algorithm and specifies three levels of discard; packets with DSCP 18 are discarded only when the queue contains its full 200 packets, while packets with DSCP 14 are discarded when the queue contains 140 or more packets, and so on. This rule can thus be used to provide the AF service of Diffserv. Of course, a random method of discard can be specified instead of a deterministic method.

Virtual labels (called Traffic Identifiers or Queue Set Identifiers) are also used to connect pairs of TC Policy or QC Policy rules. Complex QoS policies can be expressed in terms of such policies. The component architecture of PolicyXpert policies has been explained in other papers [Kan 01c, Kan 01d].

The outline of PolicyXpert architecture is illustrated in **Figure 1**. Policies are defined by the administrator or operators using the console. They are inputted to the server (PDP) and stored into the policy database. They are sent to proxy or embedded policy agents (PEP) and transformed into network device configurations and deployed to the devices. The server also manages network interfaces of the devices by using the information sent by the agents.

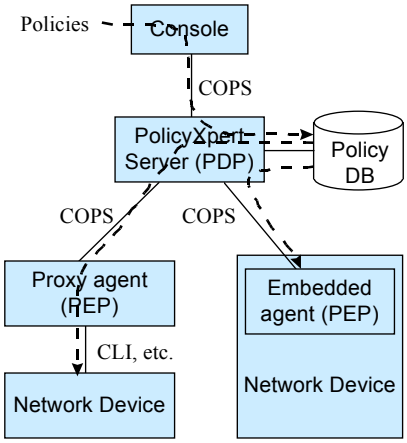

Figure 1: Architecture of PolicyXpert

4.2 A VFL function implemented in hardware

We have implemented VFLs in a hardware router. Policy division can be implemented with almost no restrictions by using this router.

The router has multiple logical filter blocks (**Figure 2**), in which packet flows can be classified and several packet filtering and QoS actions can be taken. Each filter block can be programmed by a list of condition-action rules that may be stored in a hardware table. A table entry is similar to a microcode instruction. The actions include packet dropping, marking a DSCP, policing, parameter determination for queuing, scheduling and shaping. The classifier in the first filter block can test the six tuples, i.e., the source and destination addresses, the IP protocol, the source and destination port (when the IP protocol is TCP or UDP), DSCP, and several other fields of the packet header. We have developed hardware for marking of flow IDs (i.e., VFLs) in the first filter block, and hardware for classifying packets by testing the flow ID in addition to the six tuples in the second filter block.

In typical usage, a flow ID is only assigned in one rule and is only referred to by one rule. The packets that pass through the assignment point are caught at the reference point. However, the same flow ID value can be assigned in and/or referred to by two or more rules.

Assume that the condition parts of rules in the filter blocks cannot be disjunctive ("OR"ed) conditions. Flows must be aggregated to implement disjunctive conditions. A single flow ID (VFL) is assigned to multiple flows to be aggregated. For example, the following rules can be used for detecting the condition "Source_IP is 192.168.2.10 || Source_IP is 192.168.2.20":

```
if (Source_IP is 192.168.2.10) { VFL = "Aggregated"; },
if (Source_IP is 192.168.2.20) { VFL = "Aggregated"; }.
```

4.3 Policy transformation

Policies are translated into router configurations by using policy division and fusion in the agent for the router. Two cases in which flow IDs are useful are explained. In the first case, flow aggregation and metering coexist. A policy division is inevitable if a TC or QC Policy is deployed to an interface, and this policy aggregates flows, i.e., the condition contains "OR"s, and an action part of the policy rules contains policing, the both filter blocks must be used. For example, if the TC Policy is **E2** shown in section 2.3, the rule in **E2** can be transformed into the following router configurations (expressed as low-level policy rules).

```
[For the first filter block]
if (Source_IP is 192.168.1.1) { Flow_ID = "EF_CID"; },
if (Source_IP is 192.168.1.3) { Flow_ID = "EF_CID"; }.

[For the second filter block]
if (Flow_ID is "EF_CID") {
    if (Information_Rate <= 1 Mbps) {
```

```
                DSCP = "EF";
        } else {   absolute_drop;  }; }.
```

The policy must be divided in this way because two rules are required for the classification but the flows thus classified must be aggregated before metering. The flow ID enables this transformation.

The second case is as follows. The rule shown below does not perform flow aggregation nor metering but the policy is divided because there is another rule that performs both flow aggregation and metering.

```
if (Source_IP is 192.168.1.1) { DSCP = "EF"; }.
```

This rule should be divided into the following two rules.

```
[For the first filter block]
if (Source_IP is 192.168.1.1) {
      Flow_ID = "EF_CID"; }.
[For the second filter block]
if (Flow_ID is "EF_CID") { DSCP = "EF"; }.
```

This rule division is required for not changing the semantics of the classifier. If no flow ID is available, the following rules can be used instead of the above rules.

```
[For the first filter block]
if (Source_IP is 192.168.1.1) { }.
[For the second filter block]
if (Source_IP is 192.168.1.1) { DSCP = "EF"; }.
```

The first rule is required even if the action part of this rule is empty in general because, if this rule is removed, a rule below this rule may catch a flow that must be caught by this rule and the semantics of this policy may be destroyed.

If the condition does not contain a DSCP, this division does not change the semantics. However, the evaluation of the condition may take more time than the condition that only contains a flow ID.

5. Evaluation

We measured the performance and confirmed that an introduction of VFLs does not degrade the performance. The measurement method was as follows. Two policies, **F** and **S**, were set to the hardware table of the router. Five flows that had different source IP address and that had the same bit rate (284 kpps, 64-byte packet)[1] are generated by Smartbit 6000B. The flows, whose total rate was 1.42 Mpps, were inputted to the router by a Gigabit Ethernet line, and the output was returned to the Smartbit by another Gigabit Ethernet line. The total traffic was slightly below the rate limit of Gigabit Ethernet (1 Gbps). Policy **F** had 100 rules, and the flows hit the 10th, 30th, 50th, 70th and 90th rules. Policy **S** had five rules, and five VFLs connect rules in **F** and **S**. The total input and output rates were both measured to be 1.42 Mpps, i.e., no performance degradation occurred.

6. Conclusion

Policies should cooperate for programmable and customizable policy-based networking, and policy fusion and division, which cause restrictions on the form of the policies, are sometimes required to implement the policies on high-performance hardware routers. We have developed a method of software-hardware integration of policies; i.e., a router architecture that supports VFLs (flow IDs) in hardware and developed a method of policy division for this architecture. The restrictions on the policies have been minimized by this method. We are developing a policy agent (PEP) and a gigabit router integrated by using this method to support the Diffserv policies of PolicyXpert. Preliminary evaluation results shows that both high-performance and flexibility are achieved by this integration.

[1] The size of each packet is 84 bytes including the overhead.

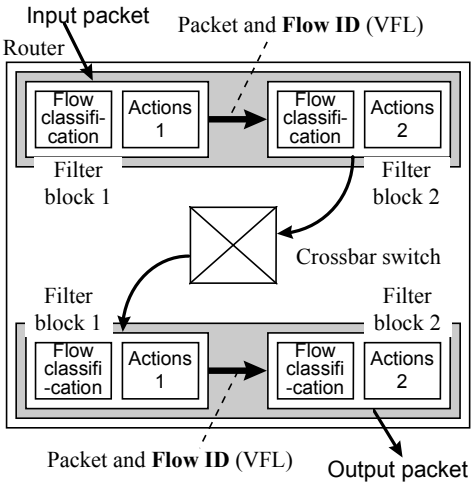

Figure 2: Filter modules in a router

Acknowledgment

We thank Toshio Shimojou, Yuuji Isezaki, and Hiromasa Okamoto of the Enterprise Server Division, Hitachi, Ltd., and Brian O'Keefe from Hewlett-Packard Company for discussing policy division and fusion and similar problems in PolicyXpert with us. We also thank Takao Nara, who measured the performance of the hardware.

References

[Ber 99] Bernet, Y., Binder, J., Blake, S., Carlson, M., Carpenter, B. E., Keshav, S., Ohlman, B, Verma, D., Wang, Z., and Weiss, W., "A Framework for Differentiated Services", draft-ietf-diffserv-framework-02.txt, *Internet Draft*, IETF, February 1999.

[Cha 01] Chan, K. H., Durham, D., Gai, S., Herzog, S., McCloghrie, K., Reichmeyer, F., Seligson, J., Smith, A., and Yavatkar, R., "COPS Usage for Policy Provisioning (COPS-PR)", RFC 3084, IETF, March 2001.

[Dur 00] Durham, D. (ed.), Boyle, J., Cohen, R., Herzog, S., Rajan, R., and Sastry, A., "The COPS (Common Open Policy Service) Protocol", RFC 2741, IETF, January 2000.

[HP 00] HP OpenView PolicyXpert 2.0 — Users Guide, Edition 1, Hewlett-Packard, October 2000.

[Jac 99] Jacobson, V., Nichols, K., and Poduri, K.: An Expedited Forwarding PHB, RFC 2598, IETF, June 1999.

[Kan 00a] Kanada, Y., "A Representation of Network Node QoS Control Policies Using Rule-based Building Blocks", *International Workshop on Quality of Service 2000* (*IWQoS 2000*), pp. 161–163, June 2000.

[Kan 00b] Kanada, Y., "Two Rule-based Building-block Architectures for Policy-based Network Control", *2nd International Working Conference on Active Networks* (*IWAN 2000*), pp. 195–210, October 2000.

[Kan 01a] Kanada, Y., "Taxonomy and Description of Policy Combination Methods", *Workshop on Policies for Distributed Systems and Networks* (*Policy 2001*), Lecture Notes in Computer Science, No. 1995, pp. 171–184, Springer, January 2001.

[Kan 01b] Kanada, Y., "Policy Division and Fusion: Examples and A Method – or, Multiple Classifiers Considered Harmful –", *7th IFIP/IEEE International Symposium on Integrated Network Management* (*IM 2001*), pp. 545–560, IEEE, May 2001.

[Kan 01c] Kanada, Y., and O'Keefe, B. J., "Diffserv Policies and Their Combination in OpenView/JP1 PolicyXpert", *5th Asia-Pacific Network Operations and Management Symposium* (*APNOMS 2001*), p. 501, September 2001.

[Kan 01d] Kanada, Y., "Diffserv Policies and Their Combinations in a Policy Server Called PolicyXpert", SIG Information Networks & SIG Network Systems, IEICE, March 2002.

[Moo 01] Moore, B., Ellesson, E., Strassner, J., and Westerinen, A., "Policy Framework Core Information Model — Version 1 Specification", RFC 3060, IETF, February 2001.

[Nic 98] Nichols, K., Blake, S., Baker, F., and Black, D., "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, IETF, December 1998.

[Par 83] Partsch, H., and Steinbruggen, R., "Program Transformation Systems", *Computing Surveys*, Vol. 15, No. 3, pp. 199–236, Association for Computing Machinery, 1983.

[Sni 00] Snir, Y., Ramberg, Y., Strassner, J., and Cohen, R., "Policy Framework QoS Information Model", draft-ietf-policy-qos-info-model-02.txt, *Internet Draft*, IETF, November 2000.

[Tom 01] Tomlinson, G., Orman, H., Condry, M., Kempf, J., and Farber, D., "Extensible Proxy Services Framework", draft-tomlinson-epsfw-00.txt, *Internet Draft*, IETF, July 13, 2000.