Policy Division and Fusion: Examples and A Method – Or, Multiple Classifiers Considered Harmful –

Yasusi Kanada
IP Network Research Center, Research & Development Group, Hitachi, Ltd.
Totsuka-ku Yoshida-cho 292, Yokohama 244-0817, Japan
kanada@crl.hitachi.co.jp

Abstract

Because higher- and lower-level policies do not necessarily correspond one-to-one, a higher-level network policy may have to be translated into two or more lower-level policies, and two or more cooperating higher-level policies may have to be translated into one lower-level policy. The former transformation is called a policy division, and the latter transformation is called a policy fusion. These transformations can be performed mechanically under certain restricted conditions, as described in this paper. In general, however, such transformations are very complicated, and the restrictions cannot be eliminated completely. This is mainly due to the existence of multiple packet classifiers in a set of policies. This paper thus concludes that multiple classifiers should not be introduced, if possible. Policy division and fusion can be avoided in certain cases, but are probably unavoidable in other cases. Given this, these problems should be solved by introducing virtual flow labels to remove harmful classifiers and conducting further studies on policy division and fusion.

Keywords

Policy-based networking, Policy division, Policy fusion, Program transformation, Packet classifier, Flow classifier, Virtual flow label.

1. Introduction

Policy-based heterogeneous networks are (will be) controlled and managed by using device-independent policies. The policies are managed by a policy server, and must be deployed mechanically to network devices such as routers. Policies deployed to devices must cooperate with each other to accomplish required functions. For example, in a Diffserv (Differentiated Services) [Ber 99] network, there may be policies that mark a DSCP (Diffserv Codepoint) [Nic 98] at the network edge and queuing/scheduling policies in the network core, and these policies must cooperate to maintain service-level agreements (SLA). These policies deployed to devices can thus be regarded as the building blocks of a higher-level policy [Kan 00a][Kan 00b].

A policy server, or a proxy that mediates between the policy server and the network devices, must translate the policies mechanically into multi-vendor device configurations. A policy consists of policy rules, which are if-then rules (condition-and-action rules). In network devices such as routers, functions such as QoS or access control are configured by using command-line interfaces (CLIs). Commands define conditions and actions; i.e., such device functions are also controlled by using policies that consist of device-level policy rules. For example, in a Cisco router, conditions are defined by an access control list (ACL) and referred to by an action command. Thus, the policy server or proxy must translate higher-level (device-independent) policies into lower-level (possibly device-dependent) policies. This translation can be compared to the compilation of programs written in languages such as C++.

Both higher-level policies [Moo 00][Sni 00] and lower-level policies [Str 00] are modeled by the Policy Framework Working Group of the IETF (Internet Engineering Task Force). A protocol for conveying lower-level policies, called the COPS (Common Open Policy Services) protocol [Dur 00], was standardized by the Resource Allocation Protocol (RAP) Working Group of the IETF. Its extension for provisioning policies, which is called COPS-PR [Cha 00], and policy information bases (PIBs) (e.g., the framework PIB [Fin 00a] and the Diffserv PIB [Fin 00b]) are both going to be standardized by the RAP and other working groups of the IETF.[1]

Although policy translation can be compared to program compilation, the process may be much more complicated. The functions of lower-level policies do not necessarily correspond to those of higher-level policies. A higher-level policy may have to be translated into two or more lower-level policies, and two or more higher-level policies may have to be translated into one lower-level policy. For example, a higher-level policy contains two (cooperating) functions, but no lower-level policy may contain both functions. In this case, the policy must be divided into two; otherwise, it cannot be deployed to devices. The translation process can thus be regarded as a type of program transformation [Par 83].

[1] Although the Policy Framework and RAP Working Groups of the IETF deal with higher- and lower-level policies, they do not deal with the translation process between them.

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 specific hardware in order to meet performance requirements. In the 1970s, routers were general-purpose computers. Today, however, high-end routers, such as the Cisco 12000 series, include hardware-implemented routing engines and some of them include hardware-implemented QoS or MPLS mechanisms. They thus cannot be programmed as flexibly as general-purpose computers. Therefore, an elaborate translation algorithm may be required for such devices.

In this paper, the outlines of policy division and fusion are defined, several typical examples are discussed, and a transformation method is described.

2. Basics of Policy Division and Fusion

This section defines policy division and fusion, and outlines their functions.

2.1 Definitions

There are two types of transformations in translating higher-level policies to lower-level policies.

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

A transformation may be a combination of policy divisions and policy fusions; i.e., a set of higher-level policies may be transformed into a set of lower-level policies, but the functions of one higher-level policy may be separated into two or more lower-level policies and the functions of two or more higher-level policies may be merged into one lower-level policy. These transformation types are illustrated in **Figure 1**.

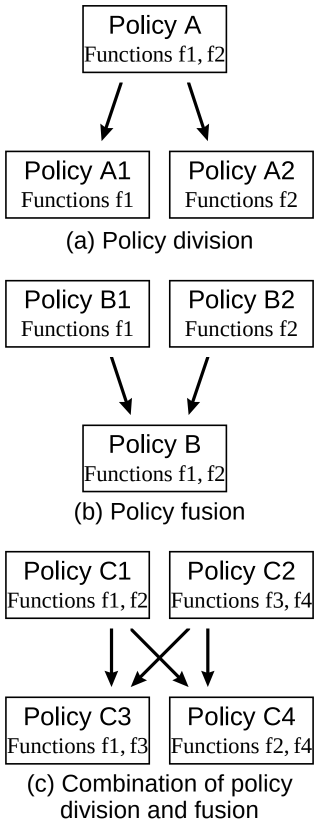


Figure 1: Policy division and fusion, and their combination

2.2 Outline of policy division

We assume a higher-level policy divided into multiple lower-level policies. Both the higher- and lower-level policies have rules with conditions. A rule in the higher-level policy must correspond to a rule in each of the lower-level policies. For example, a rule in a higher-level policy **P**,

P: if (Ci) {$Ai1$; …; Ain;}, (p)

where Ci is a condition and $Ai1$, …, Ain are sequences of actions, is assumed to be translated into the following rules in lower-level policies $\mathbf{P}_1$, …, $\mathbf{P}_n$:

$\mathbf{P}_1$: if (Ci) {$Ai1$;}, (p1)

…,

$\mathbf{P}_n$: if (Ci) {Ain;}. (pn)

The conditions of these rules are the same because actions $Ai1$, …, Ain must be executed under the same condition. There are several transformation conditions that must be satisfied to make the above transformation correct. These conditions are described in Sections 3 and 4 but are omitted here for simplicity.

A list of the conditions associated with a policy rule forms a classifier [Ber 99], which classifies the packet flows that must be handled by the policy, as shown in **Figure 2**. The conditions are evaluated sequentially and at most one rule (action) is selected for execution. Here, for simplicity, the classifier in the higher-level policy and those in the lower-level policies are assumed to be identical except for syntax.

2.3 Outline of policy fusion

We assume higher-level policies are merged into one lower-level policy. The process is easier if the classifiers of the higher-level policies are identical to that of the lower-level policy. If rules p1, …, pn (in policies $\mathbf{P}_1$, …, $\mathbf{P}_n$) in the previous section are to be merged, the result is rule p. However, the classifiers are usually different, and then the conditions must be distributed.

For example, assume we merge policies $\mathbf{Q}_1$ and $\mathbf{Q}_2$, defined below:

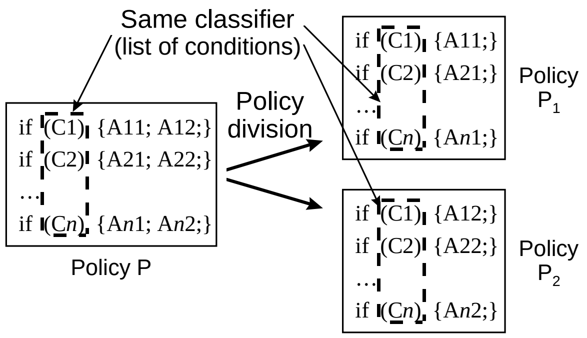


Figure 2: A classifier and its conservation in policy division

$\mathbf{Q}_1$ = { if ($C11$) {$A11$;}, …, if ($C1m$) {$A1m$;} },
$\mathbf{Q}_2$ = { if ($C21$) {$A21$;}, …, if ($C2n$) {$A2n$;} }.

If the actions in $\mathbf{Q}_1$ do not contain marking, the result must be as follows:

```
Q = {  if (C11 AND C21) {A11; A21;},
       …,
       if (C11 AND C2n) {A11; A2n;},
       if (C11) {A11;},                      (q1)
       …,
       if (C1m AND C21) {A1m; A21;},
       …,
       if (C1m AND C2n) {A1m; A2n;},
       if (C1m) {A1m;},                      (qm)
       if (C21) {A21;},                      (q01)
       …,
       if (C2n) {A2n;}                       (q0n)
     }.
```

Rules q01, …, q0n are required because there may be flows that do not match any of the conditions in $\mathbf{Q}_1$. Rules q1, …, qm are required because there may be flows that do not match any of the conditions in $\mathbf{Q}_2$. Such flows (i.e., flows for which no rule in a policy is applied) are called *default flows*. These default flows can be handled explicitly in $\mathbf{Q}_1$ and $\mathbf{Q}_2$ if the following rule is added at the end of the rule list:

if (true) {}.

The above transformation then becomes simpler. For example, the following rule will be generated from "if ($C11$) {$A11$;}" in $\mathbf{Q}_1$ and "if (true) {}" in the updated $\mathbf{Q}_2$:

if ($C11$ AND true) {$A11$; }.

Rule q1 is a simplified form of this rule.

If an action in $\mathbf{Q}_1$ contains marking and $\mathbf{Q}_2$ refers to the marked value, the above transformation cannot be applied, because the rule in $\mathbf{Q}_2$ refers to the newly marked value, while the corresponding rule in $\mathbf{Q}$ refers to the original value. Thus, a specific transformation method must be developed in such cases.

Three or more policies can be merged sequentially; i.e., the first two can be merged first, then the resulting policy and the third policy can be merged, and so on. If the numbers of rules in higher-level policies $\mathbf{Q}_1, \ldots, \mathbf{Q}_N$ are $n_1, \ldots, n_N$, then the number of rules in the fused policy is the product $(n_1+1)\ldots(n_N+1)$. This may be a very large number, leading to serious performance degradation. However, if the evaluation result of a condition in the merged rule is always false, the rule can be removed.

3. Examples

This section describes examples of Diffserv policy design and examples of policy division and fusion.

3.1 Outline of Diffserv policies

The outline of the Diffserv policies used in the transformation examples below is given here. Diffserv policies are used for the examples because they are common and the author has some experience in the division and fusion of Diffserv policies. Two types of higher-level policies designed for policy servers are defined here.

1. *Edge policy:* A type of policy that *classifies*, *meters*, and/or *marks* packets. This type of policy can be used in customer edge routers, provider edge routers, or any other type of edge router. The information rates of flows can be metered and compared with values specified in the rules. The packets may be marked with a DSCP unconditionally or according to the results of metering.
2. *Core policy:* A type of policy that *queues*, *schedules*, and *randomly drops* packets. This type of policy can be used in any type of core router. This policy can also *classify* packets. A scheduling algorithm, queue length, and WRED (weighted random early drop) can be specified.

As noted, both edge and core policies can classify packets. The classification function in edge policies is usually used for multi-field (MF) classifications [Car 98]; i.e., classifications by 6-tuples: the source and/or destination IP address, the IP protocol, the source and/or destination port, and/or the DSCP. The classification function in core policies is usually used for behavior aggregate (BA) classifications [Car 98]; i.e., classifications by DSCP.

An edge policy is usually used in edge routers, and a core policy is usually used in core routers. However, both of them may be specified in the network interface of an edge router. This cooperation of edge and core policies is preferable in some situations.[1] In such cases, the edge policy works first, followed by the core policy.

Examples of edge policy and core policy rules follow:

```
Edge:  if (Source_IP == 192.168.0.1) {
           if (Information_rate < 1M bps) {
               DSCP = 46; /* EF */
           } else {
               drop;
           };
       },
```

[1] This means that an edge router interface may have both edge and core roles. See Section 3.4, for example.

```
Core:  if (DSCP == 46) {
           Queue_number = 6;
           Scheduling_algorithm = "Priority_scheduling";
       }.
```

Next, two types of lower-level policies implemented in routers are defined here.

1. *Filtering policy:* A type of policy that *classifies*, *filters*, and/or *marks* a DSCP on packets. A filtering policy can thus be used for access control and marking.
2. *Metering and scheduling policy:* The information rates and burst sizes of flows can be *metered* by using a metering and scheduling policy. If the metering result is “out-of-profile”, the packets can be *remarked* or *dropped* absolutely according to the specifications in this policy. Packets can also be *queued*, *scheduled*, and *classified* by using this policy.

Both filtering policies and metering and scheduling policies can classify packets.

Examples of filtering policy and metering and scheduling policy rules follow:

```
Filtering:
    if (Source_IP == 192.168.0.1) {
        DSCP = 46; /* EF */
    },

Metering and scheduling:
    if (DSCP == 46) {
        if (Information_rate < 1M bps) {
            forward;
        } else {
            drop;
        };
    }.
```

Neither of these rules can have both metering and marking functions. An edge policy rule that contains both metering and marking must thus be divided into a filtering policy rule and a metering and scheduling policy rule.

A router is assumed to have only one instance of each policy; i.e., it can have neither two filtering policies nor two metering and scheduling policies. Thus, a metering function that is specified in an edge policy and a queuing and/or scheduling function that is specified in a core policy must be combined into a metering and scheduling policy. This means the two higher-level policies must be fused.

The conditions of policy rules in the filtering and metering and scheduling policies may not contain “OR”; i.e., subconditions cannot be ORed. If the condition “A OR B” is required, A and B must be tested by separate rules in the filtering policy, and the flows passing through these rules must be aggregated (Section 3.3).

These policies are closer to the device-level policies of the earlier versions of Hitachi’s gigabit router GR2000 [Aim 00], and the translation process described here is similar to that used in the GR2000 Proxy Agent for PolicyXpert,[1] but the constraints are tighter in the above policies.

3.2 Division with metering

If an edge policy contains a rule with classification, filtering, marking, and metering, this rule (and policy) must be translated into a filtering policy and a metering and scheduling policy. The edge policy is defined as:

```
E =  {  if (C1) {
            F1;
            if (MC1) {Min1;} else {Fout1; Mout1;};
        },
        …,
        if (Cn) {
            Fn;
            if (MCn) {Minn;} else {Foutn; Moutn;};
        }
     },
```

where C1, …, C*n* are conditions that test the 6-tuples of the packets. “OR” is assumed not to exist in these conditions. MC1, …, MC*n* are metering conditions that test the information rate of the flow [Ber 00]. F1, …, F*n*, Fout1, …, and Fout*n* are filtering actions that either drop all the packets or do nothing. Min1, …, Min*n*, Mout1, …, and Mout*n* are marking actions that mark a DSCP on the packets.

The resulting filtering policy and metering and scheduling policy are:

```
F  =  {  if (C1) {F1; Min1;},
         …,
         if (Cn) {Fn; Minn;}
      },

MS =  {  if (C1) {
             if (MC1) {} else {Fout1; Mout1;};
         },
         …,
         if (Cn) {
             if (MCn) {} else {Foutn; Moutn;};
         }
      }.
```

Conditions C1, …, C*n* in policy **MS** may refer to the DSCP. However, the DSCP may be rewritten by a rule in policy **F**, in which case the evaluation result of condition C*i* (*i* = 1, …, *n*) in **MS** will be different from that of the same condition in **F**. In this case, the transformation from policy **E** to **F** and **MS** is incorrect, so if a condition in **E**

[1] OpenView PolicyXpert (Version 2) is a policy server jointly developed by Hewlett-Packard Company and Hitachi, Ltd. OpenView and PolicyXpert are trademarks of Hewlett-Packard Company.

refers to a DSCP, the translator must check the corresponding actions. If an action in **E** marks the same DSCP, the transformation fails and the policy server reports that **E** cannot be deployed to the device. This problem occurs because the classifier in **E** is duplicated in **F** and **MS**; i.e., the multiple classifiers cause the problem.

To solve this problem, a virtual flow label (VFL) [Kan 00b][Kan 99] can be introduced. A VFL is a label attached to a packet or flow and is similar to a DSCP or an MPLS label. However, a VFL exists outside the packet (this is similar to an MPLS label that is outside an IP packet[1]), and the number of different VFLs is virtually unrestricted. If a VFL is supported by the device, **F** and **MS** can be rewritten as:

```
F' =    {  if (C1) {F1; Min1; VFL = "L1";},
              …,
           if (Cn) {Fn; Minn; VFL = "Ln";}
        },
MS' =   {  if (VFL == "L1") {
              if (MC1) {} else {Fout1; Mout1;};
           },
           …,
           if (VFL == "Ln") {
              if (MCn) {} else {Foutn; Moutn;};
           }
        }.
```

Because the actions in **F** never rewrite the VFL, **E** can be always transformed into **F**' and **MS**'. However, the device must support VFLs to be able to accept **F**' and **MS**'.

Each VFL value assigned in **F**' can be interpreted as a pointer to a specific rule in **MS**'. In this interpretation, the list of the conditions in **MS**' is no longer a real classifier, but each VFL value can be regarded as a control label (similar to a "goto" label), which is used in a very restricted way. The problem is solved because this transformation does not introduce multiple classifiers.

In the example above, all the rules in **E** have a metering action. However, a rule in **E** may have no metering action:

```
E:   if (Ci) {Fi;}.
```

There seems to be no need to add a rule derived from this rule to **MS**, because the whole function of this rule can be represented by a rule in **F**. However, to keep the classifiers in **E**, **F**, and **MS** identical, this rule must also be split into two:

```
F:   if (Ci) {Fi;},
MS:  if (Ci) {}.
```

[1] A VFL is more similar to a generalized MPLS label [Ash 00], which may be mapped to the layer-two header, TDM (Time-Division Multiplex) time slots, wavelength, or physical space.

In general, the rule in **MS** cannot be eliminated. However, for the purpose of optimization, empty rules *may* be combined with other rules in the same policy.

3.3 Division with aggregation and metering

If an edge policy is almost the same as **E** in the previous subsection, but the conditions C1, …, C*n* contain "OR", the result of the policy division will be different because "OR" is not allowed in a filtering policy. If the *i*th rule in edge policy **E**'' is

```
E'':  if (Ci1 OR … OR Cil) {
                               /* Ci is Ci1 OR … OR Cil. */
         Fi;
         if (MCi) {DSCP = di;} else {Fouti; Mouti;};
      },
```

then this rule must be split into *i* rules in the filtering policy:

```
F'':  if (Ci1) {Fi; DSCP = di;},
      …,
      if (Cil) {Fi; DSCP = di;}.
```

The original rule aggregates the flows before applying the actions, but the translated rules implicitly aggregate the flows after the actions are taken. They are aggregated because they have the same DSCP.

However, the corresponding metering and scheduling rule cannot be split into *l* rules because the metering result would be changed. The metering must be applied to the aggregated flow, so the rule in **MS**'' (updated **MS**) must be

```
MS'':   if (DSCP == di) {
           if (MCi) {} else {Fouti; Mouti;};
        }.
```

Because this new rule refers to a DSCP, **E**'' must satisfy the condition that the rules (that follow the above rule) in this policy must not refer to DSCP d*i*. If this condition does not hold, this transformation fails. To avoid this failure, a VFL can be used instead of DSCP:

```
F''':   if (Ci1) {Fi; DSCP = di; VFL = "La";},
        …,
        if (Cil) {Fi; DSCP = di; VFL = "La";},
MS''':  if (VFL == "La") {
           if (MCi) {} else {Fouti; Mouti;};
        }.
```

The value of the VFL must be initialized to a value other than "La" (e.g., ""). Again, the problem is caused by the multiplication of a classifier, and it is solved by not generating multiple real classifiers. Introducing VFLs is thus a method of solving this problem.

3.4 Fusion of BA classifications

If an edge policy and a core policy satisfy the following conditions, these policies must be merged into a metering and scheduling policy.

- The edge policy contains a rule with metering, the core policy contain a rule with queuing and scheduling, and these rules work on the same network interface and on the same (maybe aggregated) flow.
- Both policies have BA classifiers.

Such a situation can occur if the network interface in which the policies are deployed is the outbound interface of an edge router.

The edge policy is defined as:

```
E1 =  { if (DSCP == d1) {
            if (MC1) {} else {drop;};
        },
        …,
        if (DSCP == dn) {
            if (MCn) {} else {drop;};
        }
      }.
```

Such a policy can be used for a border router of a provider network, which is connected to another provider's network. Packets are not marked in this policy because they are premarked by following the SLA between the providers. The core policy is defined as follows:

```
C1 =  { if (DSCP == d1) {Q1; S1;},
        …,
        if (DSCP == dn) {Qn; Sn;}
      },
```

where Q1, …, and Q*n* are queuing actions, such as "Queue_number = 6", and S1, …, and S*n* are scheduling actions, such as "Scheduling_algorithm = "Priority_scheduling"". In this case, these policies can easily be merged into one:

```
MS1 = { if (DSCP == d1) {
            if (MC1) {} else {drop;};
            Q1;  S1;
        },
        …,
        if (DSCP == dn) {
            if (MCn) {} else {drop;};
            Qn;  Sn;
        }
      }.
```

Because the classifiers of policies **E1** and **C1** are BA classifiers, these policies can easily be merged even when the classifiers are not identical. The order of rules in these policies can be changed because conditions on DSCPs are exclusive unless there are duplicate conditions. In addition, if **E1** has a rule whose condition tests DSCP *d* but **C1** does not have such a rule, a rule that tests *d* and has no action can be added to **C1** without changing the meaning, and vice versa. Thus, **E1** and **C1** can easily be modified so that they have the same classifier, and they can be merged.

Since there is no rule that rewrites the DSCP in **E1** and **C1**, there is no restriction on this transformation. However, if there was a rule that rewrites the DSCP, the transformation would become more complicated, and the existence of a default flow or loophole (see the next section) would complicate it further. If MF classifiers are allowed in the core policy, it may become too complicated to be implemented.[1] All these problems are caused by the existence of multiple classifiers. Therefore, we can avoid these problems by using a VFL to eliminate classifiers from the core policy.

3.5 Combination of division and fusion

If an edge policy and a core policy satisfy the following two conditions, the edge policy must be split into a filtering policy and a metering and scheduling policy, and the core policy must be merged into the latter. This means that both policy fusion and policy division must be applied. The two conditions are as follows.

- The edge policy contains a rule with metering and marking, the core policy contains a rule with queuing and scheduling, and these rules work on the same network interface and on the same (possibly aggregated) flow.
- The edge policy has an MF classifier, and the core policy has a BA classifier.

The core policy is **C1**, and the edge policy is defined as:

```
E2 = {  if (Source_IP == i11 OR
            … OR
            Source_IP == i1m_1) {
            if (MC1) {DSCP = d1;} else {drop;};
        },
        …,
        if (Source_IP == in1 OR
            … OR
            Source_IP == inm_n) {
            if (MCn) {DSCP = dn;} else {drop;};
        },
```

[1] There is another complexity in policy fusion. Policies to be fused may be deployed or undeployed sequentially. In such cases, replacement of a lower-level policy is required. For example, if higher-level policies P1 and P2 are deployed at the same time and then P2 is undeployed, a fused policy is deployed first and it must then be replaced by a lower-level policy generated from P1. Due to page limitations, however, this problem will not be explained further in this paper.

```
        if (true) {drop;}
    },
```

where i11, …, i1m_1, …, in1, …, and inm_n are IP addresses. The last rule in policy **E2** is added to inhibit default flows. Default flows are inhibited because they make the following transformation impossible.[1]

In this case, the resulting filtering policy and metering and scheduling policy are:

```
F2 =  {  if (Source_IP == i11) {DSCP = d1;},
         …,
         if (Source_IP == i1m_1) {DSCP = d1;},

         …,

         if (Source_IP == in1) {DSCP = dn;},
         …,
         if (Source_IP == inm_n) {DSCP = dn;},

         if (true) {drop;}
      },

MS2 ={  if (DSCP == d1) {
            if (MC1) {} else {drop;};
            Q1;  S1;
        },
        …,
        if (DSCP == dn) {
            if (MCn) {} else {drop;};
             Qn;  Sn;
        }
       }.
```

Each rule in policy **F2** corresponds to a subcondition in **E2**; i.e., a source IP condition. Flows are aggregated by rules that mark the same DSCP, and each rule in policy **MS2** handles an aggregated flow.

If there is a default flow with DSCP d*i*, the flow may be caught incorrectly by the *i*th rule in **MS2**. This is the reason that the default flow is inhibited in this example. If a VFL is introduced (i.e., each rule in **F2** assigns an appropriate value to the VFL, and each rule in **MS2** tests the VFL instead of the DSCP) this restriction can be omitted.

In **E2**, if there is a rule that neither marks nor tests the DSCP, the policy fusion is not possible. Such a rule can be called a *loophole*. For example, if the following loophole is inserted before the last rule in **E2** (which handles default flows) it will catch flows that are premarked with any DSCP and that come from IP address i0:

```
E2': if (Source_IP == i0) {}.
```

This causes a problem similar to the one caused by default flows.

However, if there is a rule that does not mark but does test the DSCP and there is neither a loophole nor a default flow, the policies can be transformed correctly. For example, we assume the following rule replaces the second from the last rule in **E2**:

```
E2'':if (DSCP == dn) {}.
```

Then, this rule can be copied into **F2** as is, the last rule in **C1** can be copied into **MS2** as is, and the resulting policies work correctly.

If there is a rule with multiple subconditions but without metering, the subconditions can be distributed to rules in the metering and scheduling policy rule.[2] An example of an original rule and the resulting rules is shown here:

```
E2''':   if (Source_IP == ii1 OR
             … OR
             Source_IP == iim_i) {DSCP = di;},

MS2''': if (Source_IP == ii1) {Qi; Si;},
        …,
        if (Source_IP == iim_i) {Qi; Si;}.
```

When the classifiers must be divided or merged, the transformation is probably only possible when the forms of the edge and core policies are highly restricted. If multiple classifiers are eliminated by introducing VFLs into both the higher- and lower-level policies, most of these problems can be solved.

4. A Method of Policy Division and Fusion

In general, the process of policy division and fusion is very complicated. It is very difficult to describe a general algorithm. Instead, this section describes a method that translates an edge policy and a core policy into a filtering policy and a metering and scheduling policy. The forms of these policies are the same as these defined in Section 3.1. When necessary, both policy division and fusion are used.

4.1 Conditions and outline

To reduce the complexity of the algorithm while maintaining its significance, the following conditions are assumed.

1. The edge policy must be given, but the core policy may be absent.
2. If the edge policy contains both marking or filtering action and metering action (i.e., the transformation must contain a policy division):
   (a) No rule in the edge policy may test the DSCP marked by another rule in the policy.

[1] The absence of default flows can be guaranteed by methods other than including an explicit rule. However, if such a rule is not given, the policy translator must test all the conditions in the policy and prove that there are no default flows. This is a heavy task for the translator.

[2] However, this is possible only when the semantics of Q*i* and S*i* are not changed by distribution (copying).

(b) No rule in the edge policy, which can test a DSCP, may remark the DSCP.

3. If the edge policy contains metering action and the core policy exists (i.e., the transformation must contain a policy fusion):

   (a) The edge policy may not include either a default flow or a loophole (see Section 3.5).

   (b) The classifier of the core policy must be a BA classifier; i.e., the conditions must only have DSCP conditions.[1]

4. If the edge policy contains a rule with multiple subconditions and metering action (i.e., the transformation requires splitting a policy rule with ORed conditions), the edge policy may not include either a default flow or a loophole. The DSCP that the rule marks may not be marked by any other rule in the edge policy.

VFLs are assumed to be disallowed for network devices.

A table called a core policy table is used in this method. This table is used to store the analysis results of the core policy. Each entry in this table has the form (DSCP, A), where DSCP is the key for looking up this table and A is a list of actions.

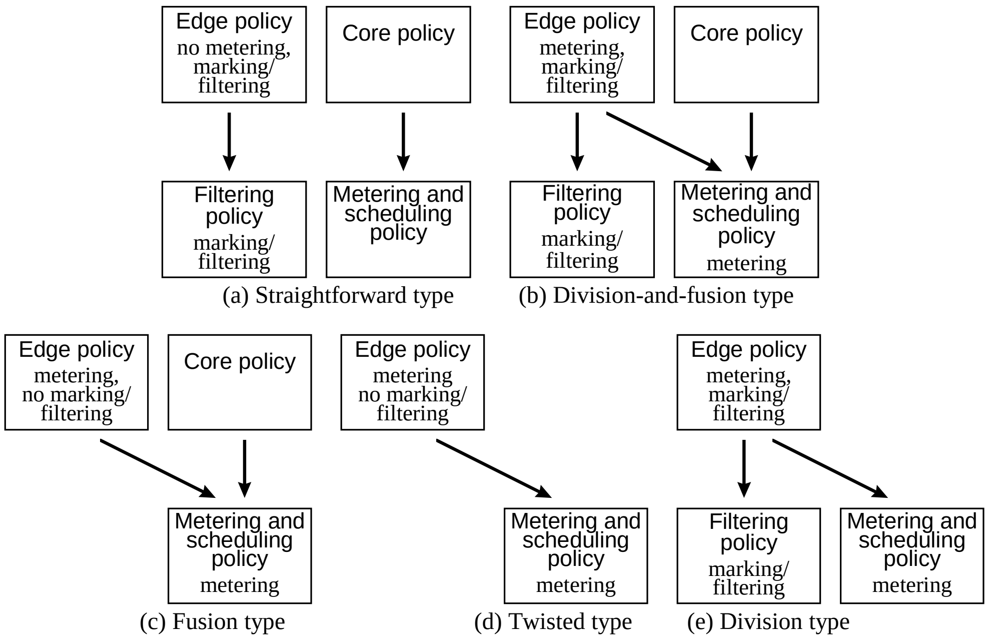
The outline of the process is as follows.

1. *Edge policy pass 1:* The edge policy is scanned and the transformation type (abbreviated as TT) is determined; i.e., division, fusion, division-and-fusion, straightforward, or twisted.

2. *Core policy pass:* The core policy is scanned and analyzed, if it exists. If the TT is the straightforward type, a metering and scheduling policy is generated from the core policy. Otherwise, the analysis results are written into the core policy table.

3. *Edge policy pass 2:* The edge policy is scanned again, and the core policy table is referred to when necessary. The policies are transformed into a filtering policy and a marking and scheduling policy according to the TT. However, only a filtering policy is generated when the TT is the straightforward type.

[1] This restriction is not necessary when the transformation type is a straightforward type (see Section 4.2). This restriction is introduced to reduce the complexity of policy fusion.



Edge policy
no metering, marking/ filtering
Core policy
Filtering policy
marking/ filtering
Metering and scheduling policy
(a) Straightforward type
Edge policy
metering, marking/ filtering
Core policy
Filtering policy
marking/ filtering
Metering and scheduling policy
metering
(b) Division-and-fusion type
Edge policy
metering, no marking/ filtering
Core policy
Metering and scheduling policy
metering
(c) Fusion type
Edge policy
metering no marking/ filtering
Metering and scheduling policy
metering
(d) Twisted type
Edge policy
metering, marking/ filtering
Filtering policy
marking/ filtering
Metering and scheduling policy
metering
(e) Division type

Figure 3: Four types of transformation

4.2 Edge policy pass 1

In this pass, the TT is determined by testing the following two conditions (**Figure 3**):

- whether there is a rule with metering, and
- whether there is a rule with marking or filtering action in the edge policy.

If the first condition does not hold, neither policy division nor fusion is required and the translation is straightforward; the filtering policy is derived from the edge policy, and the metering and scheduling policy is derived from the core policy. The core policy may be absent. This TT is called the *straightforward type* (Figure 3(a)).

If the first condition holds and there is a core policy, a policy fusion is required. In this case, if the second condition holds, a policy division is also required; i.e., the edge policy must be split and transformed into a filtering policy and a metering and scheduling policy. This TT is called the *division-and-fusion type* (Figure 3(b)).

If the first condition holds but the second condition does not hold, only a metering and scheduling policy is generated. If a core policy exists, a policy fusion is required. This TT is called the *fusion type* (Figure 3(c)). If no core policy exists, only the metering and scheduling policy is generated. This TT is called the *twisted type* (Figure 3(d)).

If the first and second conditions hold but there is no core policy (i.e., only the edge policy is given), a policy division is necessary but a policy fusion is not necessary. This TT is called the *division type* (Figure 3(e)).

In this pass, condition 3(b) in Section 4.1 is required. If there is an MF classifier in the core policy, the translation process reports an error and terminates.

4.3 Core policy pass

If the TT is the straightforward type, a metering and scheduling policy is generated from the core policy. This process is not explained any further here.

If the TT is other than the straightforward type, the following process is executed. For each rule in the core policy, the list of actions is entered into the core policy table, with the DSCP in the condition as the key. However, if two or more rules have conditions that test the same DSCP, only the first rule is entered into the table, because the policy rule to be applied is chosen according to the first match.

4.4 Edge policy pass 2

In this pass, the filtering policy and the metering and scheduling policy are generated. If the TT is the twisted type, the edge policy is copied to the metering and scheduling policy.

If the TT is the division or division-and-fusion type, then a rule is generated from each rule in the edge policy *both* for the filtering policy and for the metering and scheduling policy, even if there is no action for one of them. If the original rule marks or tests the DSCP, it is looked up in the core policy table,[1] and the actions found are fused into the rule for the metering and scheduling policy.

If the TT is the fusion type, no rules are generated for the filtering policy. If the TT is the straightforward type, rules are only generated for the filtering policy.

If a rule in the edge policy has multiple subconditions, a rule must be generated from each subcondition. This must be done even when the TT is the straightforward type. All the rules mark the same DSCP. If the original rule contains metering, a rule that tests the DSCP and meters must be generated for the metering and scheduling policy. To guarantee that this specific type of policy division is correct, condition 4 in Section 4.1 must be satisfied. This condition is required because if there is a default flow or a flow through a loophole, or if there is an action that marks the same DSCP, the flow will be wrongly aggregated to the expected flow.

This condition can be removed if a VFL is introduced into the transformation or if a separate rule that corresponds to each subcondition can be generated for the metering and scheduling policy. (See the last example in Section 3.5.)

In this pass, conditions 2(a) and (b) in Section 4.1 must be tested if the TT is the division or division-and-fusion type, and condition 3(a) must be tested if it is the fusion or division-and-fusion type. If a violation is found, the transformation process reports an error and terminates.

[1] If a rule tests a DSCP and marks another DSCP, the latter is used to look up in the table.

5. Conclusion

Although the process of translating higher-level policies into lower-level policies can be compared to the compilation process of programs in languages such as C++, this translation is much more complicated than compilation when policy division or fusion is required. If the forms of the policies are restricted, they can be transformed mechanically. However, if the restrictions are relaxed, the transformation may become too complicated to be implemented. In addition, restrictions, such as requirements for the nonexistence of default flows or loopholes, cannot be eliminated completely. Thus, these transformations should be avoided if possible. For example, in Diffserv they can be avoided if the policies for the devices or policy servers are properly defined. Functions used in other frequently used services can also be designed so that neither policy division nor fusion is required.

However, policy-based networking technology will be used for a wide variety of services. Policy-based packet processors, such as the Intel IXP1200, are pushing this trend. Thus, if the hardware function of the devices is restricted, policy division and fusion will probably be unavoidable. If they cannot be avoided, the resulting problems can be solved by introducing VFLs. Such problems are mainly caused by the existence of multiple classifiers in policies. By introducing VFLs, harmful classifiers can be eliminated from policies. Remaining problems may be solved by further studies on policy division and fusion. However, if such problems still cannot be solved completely, we may have to find a better method for controlling network devices than using policies in the current sense; i.e., a sequence of condition-and-action rules.

Acknowledgments

I thank Brian O'Keefe from Hewlett-Packard Company, and Toshio Shimojou, Yuuji Isezaki, and Hiromasa Okamoto of the Enterprise Server Division, Hitachi, Ltd., for discussing policy division and fusion problems with me.

References

[Aim 00] Aimoto, T., and Miyake, S., "Overview of DiffServ Technology: Its Mechanism and Implementation", *IEICE Transaction on Information and Systems*, Vol. E83-D, No. 5, pp. 957–964, http://search.ieice.or.jp/2000/pdf/e83-d_5_957.pdf, The Institute of Electronics, Information and Communication Engineers, 2000.

[Ash 00] Ashwood-Smith, P., et al., "Generalized MPLS – Signaling Functional Description", draft-ietf-mpls-generalized-signaling-00.txt, *Internet Draft*, IETF, October 2000.

[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.

[Ber 00] Bernet, Y., Blake, S., Grossman, D., and Smith, A., "An Informal Management Model for Diffserv Routers", draft-ietf-diffserv-model-04.txt, *Internet Draft*, IETF, July 2000.

[Car 98] Carlson, M., Weiss, W., Blake, S., Wang, Z., Black, D., and Davies, E., "An Architecture for Differentiated Services", RFC 2475, IETF, December 1998.

[Cha 00] 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)", draft-ietf-rap-pr-05.txt, *Internet Draft*, IETF, October 2000.

[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.

[Fin 00a] Fine, M., McCloghrie, K., Seligson, J., Chan, K., Hahn, S., Sahita, R., Smith, A., and Reichmeyer, F., "Framework Policy Information Base", draft-ietf-rap-frameworkpib-03.txt, *Internet Draft*, IETF, November 2000.

[Fin 00b] Fine, M., McCloghrie, K., Seligson, J., Chan, K., Hahn, S., Smith, A., and Reichmeyer, F., "Differentiated Services Quality of Service Policy Information Base", draft-ietf-diffserv-pib-02.txt, *Internet Draft*, IETF, November 2000.

[Kan 99] Kanada, Y., et al., "SNMP-based QoS Programming Interface MIB for Routers", draft-kanada-diffserv-qospifmib-00.txt, *Internet Draft*, October 1999, http://www.kanadas.com/activenet/draft-kanada-diffserv-qospifmib-00.txt.

[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.

[Moo 00] Moore, B., Ellesson, E., Strassner, J., and Westerinen, A., "Policy Framework Core Information Model — Version 1 Specification", draft-ietf-policy-core-info-model-08.txt, *Internet Draft*, IETF, October 2000.

[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.

[Str 00] Strassner, J., Westerinen, A., Moore, B., Durham, D., and Weiss, W., "Information Model for Describing Network Device QoS Mechanisms for Differentiated Services", draft-ietf-policy-qos-device-info-model-02.txt, *Internet Draft*, IETF, November 2000.