Taxonomy and Description of Policy Combination Methods

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

Abstract. To control complicated and decomposable networking functions, such as Diffserv, two or more policies must cooperate. Combining two or more mutually dependent policies for a specific purpose is called policy combination. Methods of passing information between combined policies can be classified into real tags and virtual tags, or labels and attributes. Policy combinations can be classified into concatenation, parallel application, selection, and repetition. Explicitly specifying policy combinations makes policy systems semantically clearer and better suited to general use, extends the range of functionality, and improves the possibility of optimization. If policy combinations can be specified in a policy system, two types of policy organizations can be distinguished: homogeneous and heterogeneous. Heterogeneous organization is more service-oriented and seems to meet service-management requirements, but homogeneous organization is more device-oriented and may provide better performance.

1. Introduction

In policy-controlled networks, networking functions such as access, QoS, and security function are specified in a device-independent method and customized for individual customer or group of customers. Network service is managed based on service-level agreements (SLA). SLA can be regarded as a policy specifications or high-level policies. High-level policies are broken down into low-level policies and stored into policy servers.

If the functions controlled by policies are simple, the policies can be independent of each other. In contrast, to control functions that are complicated and decomposable, such as Diffserv (Differentiated Services) [Ber 99], the relationships between policies must be taken into account. For example, marking and queuing policies in Diffserv must work together. If related policies are deployed to different targets (e.g., network devices or device interfaces), the application order may be naturally constrained. However, if policies are deployed to the same target, the order is not explicitly determined or determined in a device-specific way. This makes the semantics of policies vague, and makes the policy system less useful. Thus, the relationships between policies should be clearly defined, and these relationships should be controllable for network operators and administrators. For example, if marking and queuing policies are deployed to a network interface in a router, the marking policy must be applied before the queuing policy. This means that mechanisms to define the relationships between policies and to control the relationships in the policy description language must be supplied.

Kanada [Kan 00b][Kan 00a] previously proposed two architectures for combining building-block policies: *label-connection architecture* and *pipe-connection architecture*. Both architectures are based on declarative programming languages used in artificial intelligence and knowledge engineering. Thus, in these architectures, policies are regarded as programs. This is because they control the behavior of network nodes or a network. Label-connection architecture is a direct extension of policies that consists of if-then (condition-and-action) rules used in commercial policy servers. Pipe-connection architecture offers significant advantages in terms of semantics and parallelism. However, it is still in an early stage of research. Label-connection architecture offers its own advantages and can be implemented by using currently available technology [Kan 00b]. This paper focuses on label-connected architecture. Most of the results discussed in this paper can, however, be applied to pipe-connected architecture too.

In this paper, the term *policy combination* refers to combining two or more mutually dependent policies for a specific purpose. Our focus is on lower-level (implementation-level) policies. Means of passing information between two policy rules are classified in Section 2. Methods for combining policies are classified in Section 3. Methods for organizing combined policies are classified and explained in Section 4 using Diffserv examples. Problems and possible solutions associated with these forms of policy combination are discussed in Section 5.

2. Methods for Passing Information

A policy consists of if-then rules. To make two or more policy rules that belong to different policies cooperate, the information must be transferred between them. A single piece of information transferred between rules is called a *tag*. Tags can be *real*, i.e., in the packet, or *virtual*, i.e., outside of the packet (see

Fig. 1).[1] A tag causes a *data dependence* [Kuc 81] between policy rules.

Tags can be categorized into the following two.

1. *Labels*: A tag may be used for selecting a rule from a policy. This type of tag is called a label. A label connects one rule to another (i.e., the label assigned by one rule specifies the next rule to be applied). DSCPs (Diffserv code points) [Nic 98] are used as real labels. Virtual labels, called VFLs (virtual flow labels), were introduced by Kanada [Kan 99]. Examples of labels are described in Section 3.

2. *Attributes*: A tag might not only be used in the action part of the rule, but not in rule selection. This type of tag is called an attribute [Kan 00b]. Virtual attributes are useful for hierarchical scheduling, shaping, and policing. An example of hierarchical shaping is described in Section 3.4.

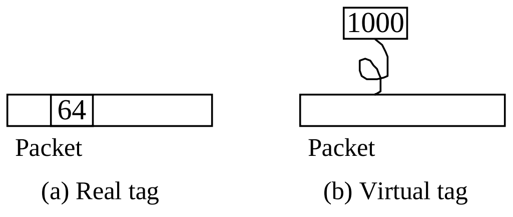


Fig. 1. Real and virtual tags

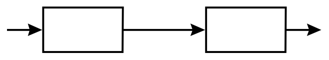
3. Types of Policy Combination

Policies change the behavior of a network in a consistent way. Thus, a set of policies that work together may be regarded as a distributed program. Each policy consists of if-then rules, so a set of policies is a rule-based program similar to expert system programs written in a language such as OPS5 [For 81].

In procedural programs, such as those written in C or C++, there are four relationships (control structures) between statements: concatenation (sequential application), parallel application, selection, and repetition. Despite the fact that a set of policies is a rule-based program, the application order and the *control dependence* [All 83] of policies must be specified. Thus, the relationships between policies can be classified into these four types.[2]

3.1 Concatenation

If two policies are sequentially applied, the relationship between these two policies can be called a concatenation. A concatenation can be represented by the following diagram.



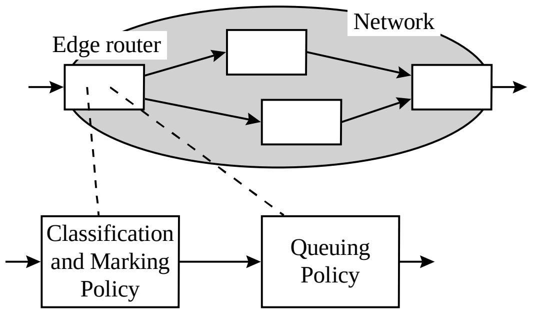


Fig. 2. Policy concatenation in a Diffserv network

For example, in a Diffserv network, a policy for MF (multi-field) classification and marking, and a policy for queuing may be concatenated and deployed to an edge router (see **Fig. 2**).[3] There are two possible policy rules in the classification and marking policy. They are as follows.

```
# Rule C1:
if (Source_IP is x.x.x.x) {
   DSCP = “EF”;
}
# Rule C2:
else if (Source_IP is y.y.y.y) {
   DSCP = “DF”;                  # “DF” == 0
}
```

In these rules, “EF” refers to expedited forwarding [Jac 99], and “DF” refers to default forwarding or best-effort forwarding. Note that the two policies are ordered by the concatenation, and the two rules in these policies are connected by DSCP. The concatenation specifies a control dependence between the policies, and the DSCP specifies a data dependence (a flow dependence [Kuc 81]) between them.

Rules in the queuing policy in the concatenation may be as follows.

```
# Rule Q1:
if (DSCP is “EF”) {
   Scheduling_Priority = 6;
                    # Set an attribute for the scheduler.
   Enqueue;         # Put the packet into a queue
                    # until the scheduler pulls it off.
}
```

[1] A virtual tag may be transmitted by additional data lines in the network device hardware, by a value in a register of the hardware or a variable in the software, or by the location in the hardware circuit. So a virtual tag cannot be transmitted between network devices, but may be transmitted within a network device.

[2] Kanada [Kan 00b] described a policy language in which the application order can be specified by using an unstructured method. However, the control should be structured as described in the present paper.

[3] A set of primitive building blocks (policies) for Diffserv was discussed by Kanada [Kan 00a][Kan 00b].

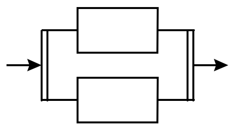
```
# Rule Q2:
else {
    Scheduling_Priority = 1;
    Enqueue;
}
```

DSCP is used as a real label in Rule Q1. DSCP "EF" connects Rules C1 and Q1, and DSCP "DF" connects Rules C2 and Q2.

Rules Q1 and Q2 assume priority scheduling. This means a priority scheduling rule is connected to the above rules. The EF traffic receives higher priority, and the DF traffic receives lower.

3.2 Parallel application

Two policies may be applied in parallel if they do not conflict. Parallel application can be represented by the following diagram.



In a Diffserv network, a marking policy and a queuing policy may be applied in parallel. For example, the marking policy in the parallel application may contain the following rules.

```
# Rule M1:
if (VFL is "Policed-EF") {
    DSCP = "EF";
}
# Rule M2:
else {
    DSCP = "DF";
}
```

Here, the marking policy is separated from the classification policy. Rule M1 is applied to higher-priority packets, and Rule M2 is applied to other (lower priority) packets. The classification policy does not mark a DSCP, it just puts a VFL on the packets.

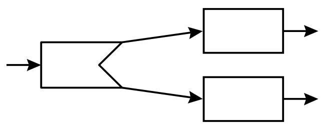
The queuing policy in the parallel application may contain the following rules.

```
# Rule Q1':
if (VFL is "Policed-EF") {
    Scheduling_Priority = 6;
    Enqueue;
}
# Rule Q2':
else {
    Scheduling_Priority = 1;
    Enqueue;
}
```

Rules M1 and Q1' refer to the same VFL value. VFL "Policed-EF" connects a rule in the classification policy to Rules M1 and Q1'.

3.3 Selection

If a policy outputs multiple types of application results depending on the conditions of the rules, and there are multiple policies each of which inputs each of the results, the relationship between these three policies can be called a selection. A two-way selection can be represented by the following diagram.



N-way selection, where *N* is larger than 2, is also possible.

For example, in a Diffserv network, a policy for policing may be combined with a policy for marking and a policy for absolute dropping, and these policies may be deployed to an edge router (see **Fig. 3**). Policy rules in the policing policy may be as follows.

```
# Rule P1:
if (DSCP is "EF" && Information_Rate <= 2 Mbps) {
    VFL = "Policed-EF";
}
# Rule P2:
else if (DSCP is "EF") {
    VFL = "Drop";
}
# Rule P3:
else {
    VFL = "Policed-DF";
}
```

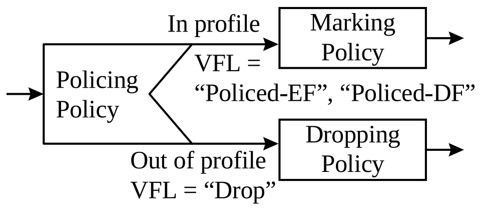


Fig. 3. Policy selection in a Diffserv network

Rules P1 and P2 are applied to packets whose DSCP is "EF". This means they are higher-priority packets. The packets are passed through an information-rate meter. Rule P1 is applied to in-profile packets, and Rule P2 is applied to out-of-profile packets. Rule P3 is applied to lower-priority packets whose DSCP is not "EF". Rules P1 and P3 attach a VFL that indicates that the packet must be forwarded with higher or lower priority, and Rule P2 attaches a VFL that indicates the packets must be dropped.

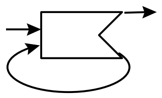
Rules in the marking policy may be the same as Rules M1 and M2 in Section 3.2. A rule in the dropping policy may be as follows.

```
# Rule D1:
if (VFL is “Drop”) {
    Absolute_Drop;
}
```

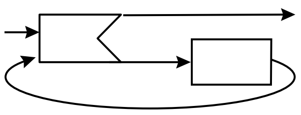
This rule drops all the packets that have “Drop” as their VFL value.

3.4 Repetition

If a policy is repeatedly applied until a condition is met, the relationship of the policy to itself can be called a repetition. A repetition with only one policy can be represented by the following diagram.



A repetition may contain two or more policies as shown below.



In a Diffserv network, for example, a hierarchical shaping function can be represented by a repetition. If the following three rules are included in a queuing policy, this policy represents a hierarchical shaping function, as shown in **Fig. 4**.

```
# Rule S1:
if (VFL is “Policed” && DSCP is “EF”) {
    Scheduling_Priority = 6;
    Maximum_Rate = 700 kbps;# shaping rate
    VFL = “Shape2”;
    Enqueue;
}
# Rule S2:
else if (VFL is “Policed”) {        # except EF
    Scheduling_Priority = 5;
    Maximum _Rate = 500 kbps;  # shaping rate
    VFL = “Shape2”;
    Enqueue;
}
# Rule S3:
else if (VFL is “Shape2”) {
    Scheduing_Algorithm = Priority_Queuing;
    Maximum_Rate = 1 Mbps;   # shaping rate
        # 200 kbps (700 k + 500 k – 1 M) or
        # less traffic may be dropped here.
    VFL = “Outgoing”;
    Enqueue;
}
```

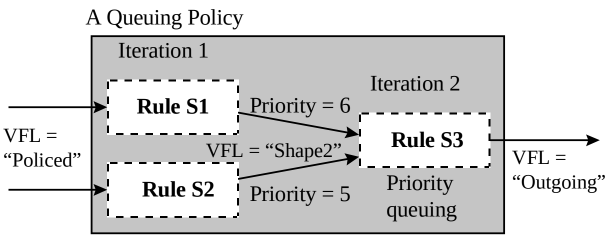


Fig. 4. Policy repetition for a Diffserv network

In this example, Rules S1 and S2 assign a virtual attribute called Scheduling_Priority to the packets and then the rules queue them. Each rule in the queuing policy has its own queue. Rule 3, which schedules packets by using a priority queuing algorithm, pulls off packets from the queues according to the Scheduling_Priority. VFLs are used for loop control. A different VFL value is used for each iteration. In this example, “Policed” is used for the first iteration, and “Shape2” is used for the second. When “Outgoing” is set as a VFL, there is no rule whose condition matches this VFL in this policy, so the loop terminates.

In this example, the number of repetitions is only two, but it may be an arbitrary value. This number, however, is usually assumed to be constant, and the rule to be applied in each iteration is usually different. Otherwise, mapping the combined policies to the network device functions is difficult, and infinite repetition might occur.

4. Methods of Policy Organization

In this section, two methods are defined for organizing combined policies. They are explained using examples, and compared.

4.1 Definitions

A set of combined policies can, in a wider sense, be regarded as a single “policy”. Thus, a *compound policy* is defined as a set of combined policies collected for a specific organizational purpose.

There are two methods for organizing combined policies.

1. *Homogeneous organization*: If no compound policies are used, and the policies are organized such that all rules in a policy have the same type of conditions and the same type of actions, the policies are said to be organized homogeneously.
2. *Heterogeneous organization*: If the policies are not organized homogeneously, the policies are said to be organized heterogeneously.

In a typical homogeneously-organized policy, all the rules, except the default rule (the rule applied when the conditions of all other rule are not satisfied), test the same tag, and the actions of all the rules are the same type.

In heterogeneous organization, a compound policy

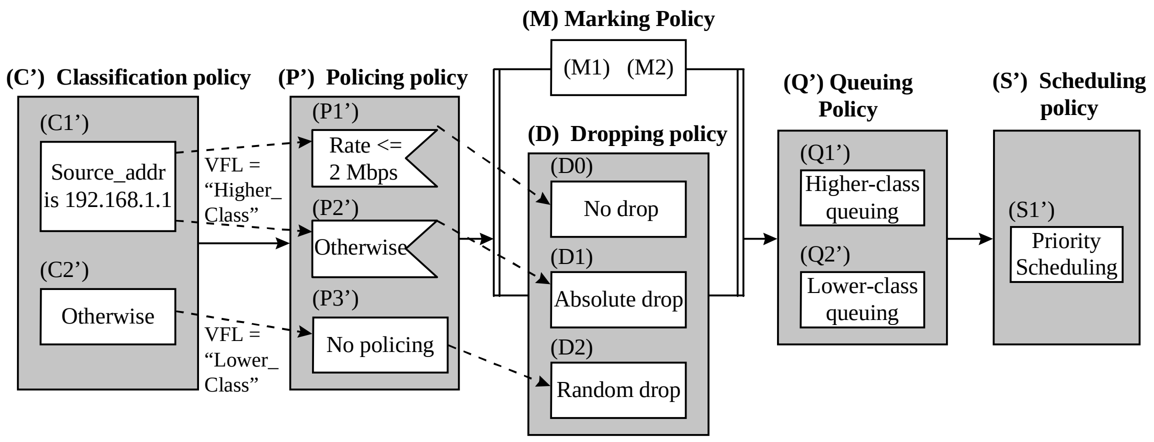


Fig. 5. A homogeneous policy set for Diffserv

may contain other compound policies and non-compound policies. Thus, compound policies are organized in a recursive way, and they may contain homogeneously organized policies.

In homogeneous organization, the same type of rules (instead of instructions) are applied to multiple data. In heterogeneous organization, different types of rules (instead of instructions) are applied to multiple data. Homogeneous organization is, therefore, analogous to single-instruction-stream multiple-data-stream (SIMD) [Fly 66] and heterogeneous organization is analogous to multiple-instruction-stream multiple-data-stream (MIMD) in parallel processing [Fly 66].

4.2 Examples

All the policies described in Section 3 are homogeneous. An example of homogeneously-organized set of policies for a Diffserv is shown in **Fig. 5**. An example of heterogeneously-organized set of policies for a Diffserv is shown in **Fig. 6**. The functions of both sets of policies are identical. These policies are only deployed to the outbound interface of an ingress edge router (**Fig. 7**). They classify and mark the packets, police the flow, and then queue and schedule the packets. There are two classes of service: premier and default. A customer whose IP address is 192.168.1.1 is a premier service customer here.

In Fig. 5, classification policy C' classifies the packets and attaches VFLs. Policing policy P' polices the classified traffics. P2' drops the excess higher-priority traffic, but P3' allows all the traffic to pass through (because the bandwidth for the lower-priority traffic do not need to be limited). P' contains rules similar to P1, P2, and P3 (described in Section 3.3), but the rules in P' test VFLs instead of DSCPs.

If a flow from the default service customer is detected (C2'), default service policy De is applied. A random dropping is applied to this flow (D2'), and the packets are marked 0 ("DF") on the DS field (M2') and queued. Finally, a Scheduling_Priority attribute is attached (Q2'').

The premier and default traffics are merged into priority scheduler S'. The premier traffic has a higher priority.

Marking policy M and dropping policy D are applied in parallel. M contains rules identical to M1 and M2 in Section 3.2. D drops all or part of the traffic if necessary. In the case of in-profile higher-priority traffic, the packets need not be dropped. So Rule D0, which explicitly passes through the traffic, is applied. Rule D1 is an absolute dropping rule, and is the same as that described in Section 3.3. Rule D1 drops the out-of-profile packets detected by Rule P2'. Rule D2 is a random dropping (WRED) rule applied to lower-priority packets.

Next, queuing policy Q' is applied and then scheduling policy S' is applied. Q' contains rules identical to Q1' and Q2' in Section 3.2. S' is different from the scheduling policy shown in Section 3.4. It is a simple priority scheduler.

In Fig. 6, if the premier service customer's flow is detected by Rule C1' in classification policy C', premier service policy Pr is applied. C' is identical to C'' in Fig. 5. Pr is a compound policy. The bandwidth is limited to 2 Mbps by the only rule in Policy P''. Out-of-profile packets are dropped by the only rule in Policy D1'. In-profile packets are marked "EF" on the DS field [Nic 98] and queued with the Scheduling_Priority attribute by rules in Policies M1' and Q1''. Policies M1' and Q1'' can be applied in parallel; i.e., packet marking can be done at anytime before the packet leaves the queue.

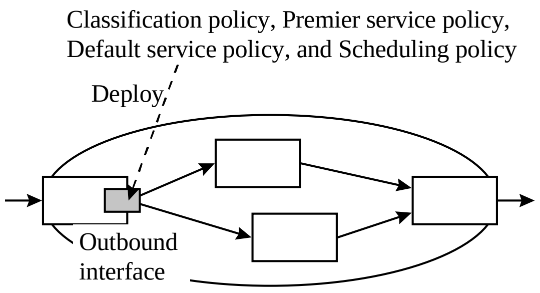

Classification policy, Premier service policy,
Default service policy, and Scheduling policy
Deploy
Outbound interface

Fig. 7. Policy deployment

4.3 Comparisons

Homogeneous organization is more device-oriented because rules of the same type are collected and a policy type usually represents specific sets of device functions, such as those in Fig. 5. This structure may be directly mapped to pipelined or SIMD packet-processing hardware, such as the routing and QoS processors in Hitachi's GR2000 router [Aim 00] and network processors. Homogeneous organization, thus, seems better suited to device and performance management purposes, and to high-speed network. This is because pipeline processing offers better performance and translating a heterogeneous set of policies to a homogeneous set is not an easy task.

Heterogeneous organization is more service-oriented because compound policies usually represent abstract functions. In Fig. 6, Pr and De represent, respectively, services for premier and normal customer policies. We can see that heterogeneous organization seems better suited to service management.

Heterogeneous organizations are usually assumed to contain less rules than homogeneous ones, because two or more policies in a heterogeneous organization correspond to a single policy in a homogeneous one. Policy, such as P', D1', M1', and Q1'', often contain only one rule. Because this may increase the complexity of the set of policies, the policy editing interface must reduce the complexity.

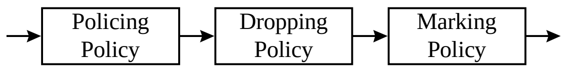
5. Discussions

The advantages of explicitly specifying policy-combination types are discussed below.

5.1 Clarification of the semantics

The semantics of a policy system can be clarified by explicitly specifying policy combination types. In commercial network policy systems, the relationships between policies are not explicitly specified. If the application order is inappropriate, policy applications may cause erroneous results. For example, the rule in the MF classification and marking policy described in Section 3.1 marks "EF" in the DS field. If a rule in a queuing policy contains the condition "DSCP is EF", and this rule is applied before the marking, the condition evaluation will cause a wrong action as a result. This problem can be avoided by specifying these policies as a concatenation. Introducing a parallel application causes a similar effect.

Introducing a selection also allows us to clarify the relationship between alternative policies. For example, an alternative structure may be represented by using concatenations. The selection in Fig. 3 may be simulated by the following diagram.



This works correctly because the dropping policy works exactly on the packets to be dropped. However, if the order of the dropping and marking policies are altered and the marking policy rewrites the VFL, these policies generate a wrong result. The following rule is assumed to be used instead of M2 in marking policy M.

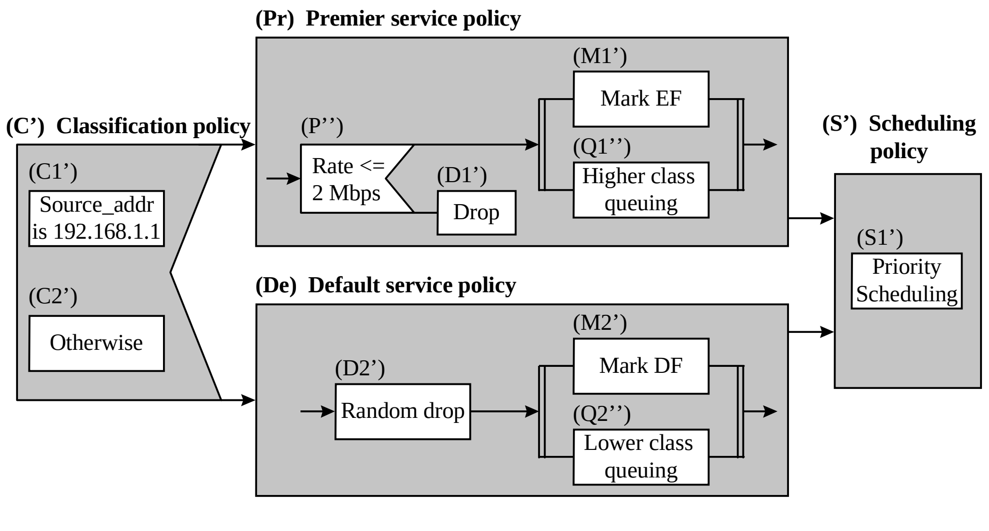


Fig. 6. Heterogeneous set of policies for Diffserv

```
# Rule M2*:
else {
   DSCP = “DF”;
   VFL = “Policed-DF”;
}
```

Then, Rule M2 will be applied to the out-of-profile packet, the VFL will be rewritten, and the packet will not be dropped. This problem can be avoided if a selection is used instead of the concatenations.

5.2 Developing policy systems suited to general use

If the policy combination type is explicitly specified for each policy combination, the policy system becomes more suited to general use. If the relationships between policies are not explicitly specified by the user, the order of the applications must, at least be predefined in the policy system or inferred from the context. If the order is predefined, the usage of the policy system is restricted, and the system is less suited to general use. Inferring relationships from generic knowledge on policies is not easy. If the relationship is explicitly specified, the policies are context-free and, thus, more suited to general use.

In addition, introduction of a repetition makes repeated application of a policy possible. It cannot be specified in conventional policy systems.

5.3 Adaptation to a variety of devices

If the policy combination types are explicitly specified, the repertory of executable policies can be extended. If the application order is strictly sequential (i.e., each pair of contiguous policies are specified as a concatenation), the policies may be unmappable to device functions because the functions are limited and the application order is constrained. For example, in some router, packets must be marked before random dropping. If random dropping strictly precedes marking in the policy description, the policy translator, which resides in the device or in a proxy, may fail to convert the policy to an executable commands.

If parallel applications and selections are specified, however, the policies may be translated. For example, if random dropping and marking are specified as parallel application as shown in Fig. 5, the policies can be translated into router commands without a possibly complicated semantic analysis.

5.4 Optimization

Policies specified by an operator may be inefficient and may have to be optimized. If the types of policy combination are explicitly specified, the possibility of optimizing policies is improved by parallel application or reordering policies. If the function of the target router is flexible enough for a number of parallel applications to be specified, and the parallel applications of policies are translated into parallel commands, the performance will be better than that associated with sequential commands. However, if the parallel applications are specified and the performance is better for a specific command-execution order, performance can be optimized.

6. Summary and Conclusion

When two or more policies work together, four types of policy combination can be distinguished: concatenation, parallel application, selection, and repetition. If the policy-combination types are explicitly specified in a policy-based system:

1. the system will be semantically clearer,
2. the system will be better suited to general use,
3. the range of functionality will be wider because the range of translatable policies widens, and
4. the possibility of policy optimization will be improved.

We can add policy combination specification, which includes type specification, to existing policy-based systems. However, if a new policy language and system is designed, we can obtain a well-defined and more concise policy-based system.

If policy combinations are specified in a policy system, two types of policy organizations can be distinguished: homogeneous and heterogeneous organizations. Heterogeneous organization is more service-oriented, so it seems to be better suited to service management. Homogeneous organization, however, is likely better for device and performance management purposes. It is advantageous in high-speed networks.

Future work includes designing a new policy system in which policy combinations and organization can be explicitly specified. Future work also includes developing methods for translating a set of combined policies into a set of policies that can be implemented by network devices. The translation methods to be developed include policy fusion, which merges two or more policies into one, and policy division, which splits a policy into two or more policies.

Acknowledgments

The author would like to thank Brian O’Keefe from Hewlett-Packard for his contributions to the development of policy combination and organization concepts, especially on the repetition.

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.

[All 83] Allen, J. R., Kennedy, K., Porterfield, C., and Warren, J., “Conversion of Control Dependence to Data Dependence”, *10th ACM Symposium on Principles of Programming Languages,* (*POPL 83*), pp. 177–189, 1983.

[Ber 99] Bernet, Y., et al, “A Framework for Differentiated Services”, draft-ietf-diffserv-framework-02.txt, *Internet Draft*, February 1999.

[Fly 66] Flynn, M. J., “Very High-speed Computing Systems”, *Proc. IEEE*, 54:12, pp. 1901–1909, 1965.

[For 81] Forgy, C. L., “OPS5 User’s Manual”, *Technical Report* CMU-CS-81-135, Carnegie Mellon University, Dept. of Computer Science, 1981.

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

[Kan 99] Kanada, Y., Ikezawa, M., Miyake, S., and Atarashi, Y., “SNMP-based QoS Programming Interface MIB for Routers”, draft-kanada-diffserv-qospifmib-00.txt, *Internet Draft*, http://www.kanadas.com/activenet/draft-kanada-diffserv-qospifmib-00.txt, October 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*), Lecture Notes in Computer Science, No. 1942, pp. 195-210, Springer, October 2000.

[Kuc 81] Kuck, D. J., Kuhn, R. H., Padua, D. H., Leasure, B., and Wolfe, M., “Dependence Graphs and Compiler Optimizations”, *8th ACM Symposium on Principles of Programming Languages* (*POPL 81*), pp. 207–218, 1981.

[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, December 1998.