Two Rule-based Building-block Architectures for Policy-based Network Control

Yasusi Kanada
Hitachi Ltd., IP Network Research Center

Relation between Active Networks and PBN

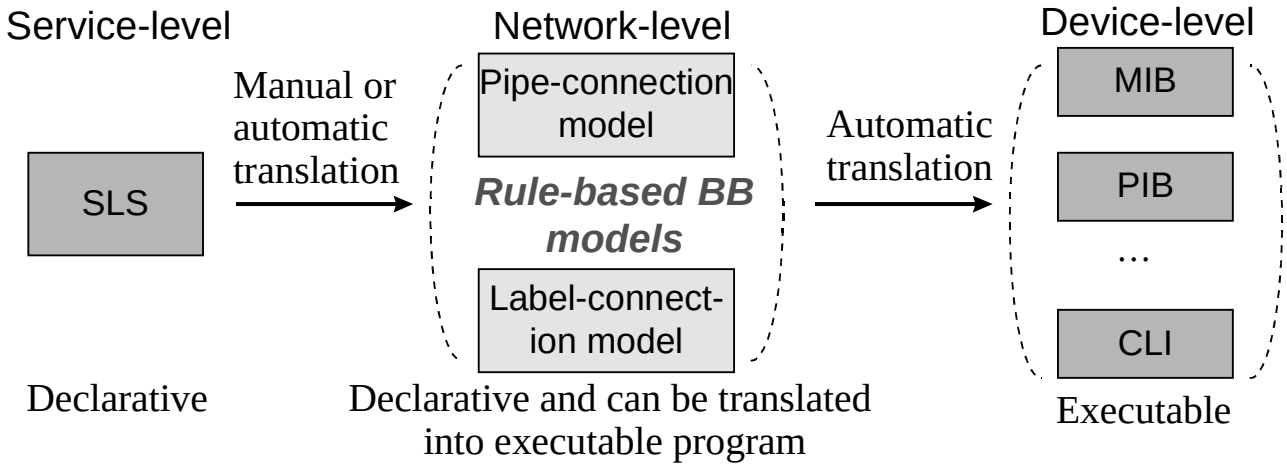
- **PBN (policy-based networking) is an apropriate first step toward active networks.**
  - PBNs are active networks (with limited functions), because
    - Active networks are
      - customizable networks, and
      - networks that programs can be injected into.
    - PBNs are also
      - customable using policies, and
      - networks that programs (called policies) are injected into.
  - Function of policies will be extended.
    - PBN is currently limited to access control, QoS, security, etc., but
    - PBN will become more general-purpose — real active networks.

Technical Requirements regarding Policy-based Networking

- **Policies must be declarative.**
  - An SLS is declarative and should be translated into policies easily.
  - Policies should not depend on specific procedure.
- **Policies must be executable.**
  - Policies change the network behavior, so they are programs.
- **Policies must be modular.**
  - Network never stops, so partial replacement of policies must be possible.
- **Policies must be structured — component-based.**
  - Translating an SLS into a policy must be possible even if the SLS is complicated.
- **An optimized policy should be expressible by using the same language.**
  - Otherwise, it is difficult to prove the original and optimized policies are equivalent.

A Method That Meets the Requirements

- **To represent policies using a *rule-based building-block* architecture.**
  - A rule-based architecture
    - is declarative and executable (such as Prolog programs), and
    - has modular rules — can be modified dynamically
  - A building-block (BB) architecture
    - is component-based, and
    - is able to be transformed into an optimized form



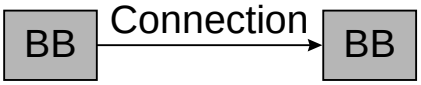
Two Building-block Architectures

■ Two architectures are proposed.

- ◆ Pipe-connection architecture
  - ▮ a refined version of the architecture [Kanada@IWQoS 2000]
- ◆ Label-connection architecture
  - ▮ a refined version of the architecture [Kanada@IETF 47]

■ Common structure

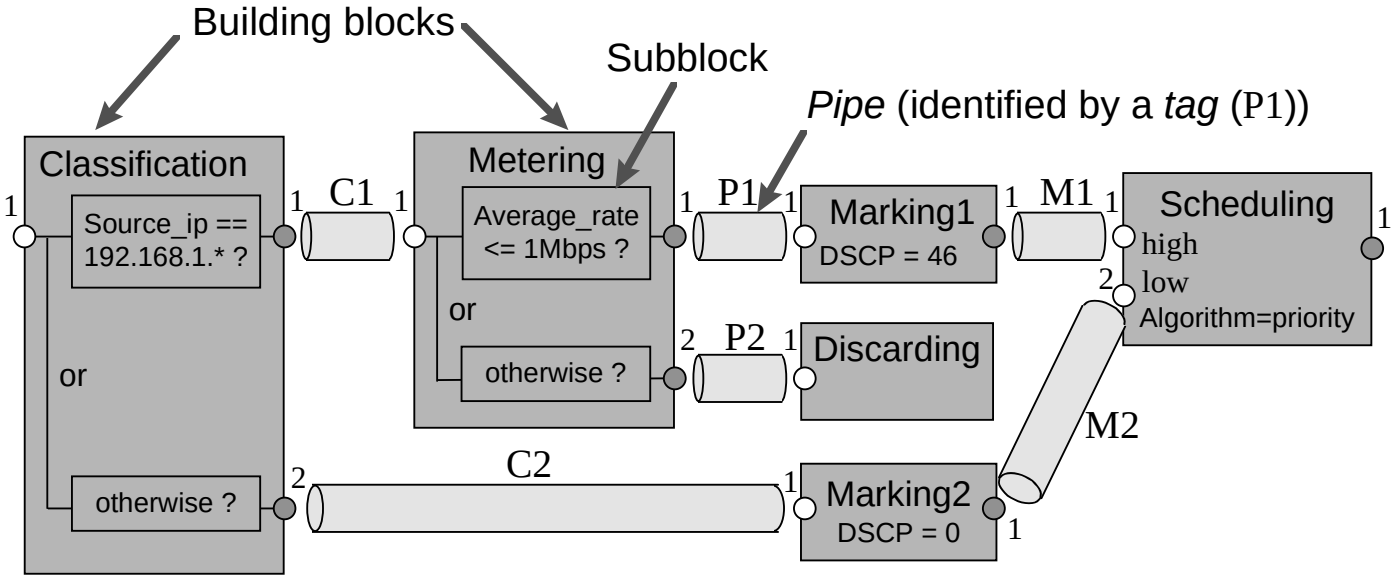
- ◆ Policies or policy rules consist of
  - ▮ building blocks: rules or sets of rules, and
  - ▮ connections between BBs



- ◆ Behavior of a BB
  - ▮ a rule is selected if the input packet matches the condition,
  - ▮ the action is executed, and then
  - ▮ an output packet is generated.
- ◆ If no condition matches the input packet,
  - ▮ no action is taken, and
  - ▮ no packet is outputted.
- ◆ A BB inputs a stream of packets and outputs a stream of packets.

Pipe-connection Architecture

■ An example: Diffserv-based router control policy



■ Each BB has one or more (but fixed number of) I/O ports.

Pipe-connection Architecture (cont'd)

- **This architecture can be expressed using a "parallel logic language".**
  - e.g. GHC, Concurrent Prolog, or Parlog.
    - These languages were developed in the "5th Generation Computer Projects".
  - This type of language is good for stream processing.
    - Good for packet flow processing — a packet flow is a type of data stream.
- **BBs are represented by predicates (rules).**
- **Pipes are represented by logical varialbles.**

Pipe-connection Architecture (cont'd)

- **The Diffserv example using a language called SNAP***

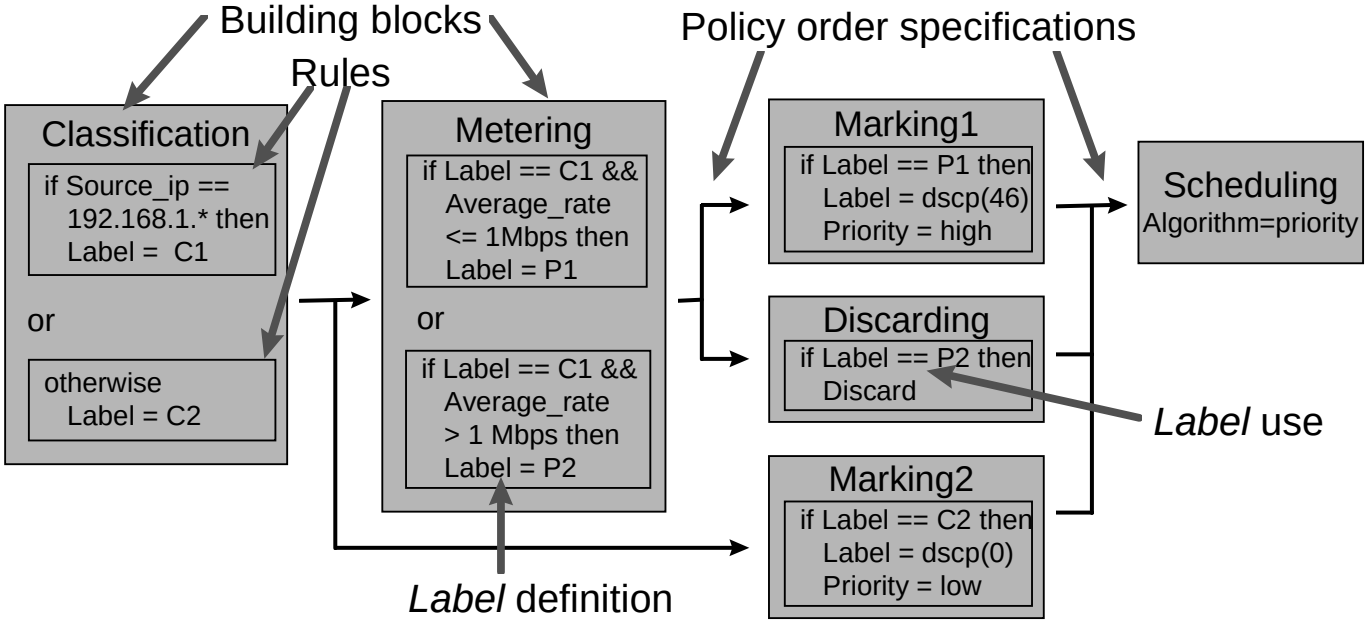
* Structured Network programming by And-Parallel language

```
ef_ingress(Si, So) :–        // Building block ef_ingress inputs stream Si and outputs stream So.
   or( filter[Source_IP = 192.168.1.*](Si, C1) |
            // Packets (in Si) whose source IP subnet is 192.168.1.* are outputted to C1.
         or( meter[Average_rate_max = 1Mbps](C1, P1) |
                  // Packets (in C1) within the bandwidth limit are outputted to P1.
               mark[DSCP = 46](P1, M1)
                                    // Packets in P1 are marked and outputted to M1.
          ;   otherwise(C1, P2) | // Packets (in C1) that do not meet other (only one here)
                                  // conditions in the case structure are outputted to P2.
               discard(P2)        // All the packets in P2 are discarded.
         )
    ;  otherwise(Si, C2) | // Packets (in Si) that do not meet other conditions
                           // in the case structure are outputted to C2.
         mark[DSCP = 0](C2, M2)      // Packets in C2 are marked and outputted to M2.
   ),
   schedule[Algorithm = priority](M1, M2, So).
                  // Streams M1 and M2 are merged into So.  A queue is assigned to M1,
                  // and another queue is assigned to M2. They are scheduled by a priority
                  // scheduler.  The priority of M1, which is the first argument, is higher.
```

Predicate
Logical variable

Label-connection Architecture

- **An example: Diffserv-based router control policy**



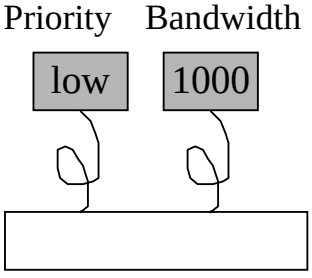
- **Each BB has only one input port and one output port.**
- **Execution order is determined using both *labels* and *policy order specifications*.**

Label-connection Architecture (cont'd)

- **Two types of labels: real and virtual labels**

(a) DSCP (a real label) (b) VFL 1000 (virtual (flow) label)

- **A packet/flow may have two or more tags (attributes)**



* Attributes are tags except labels.

Label-connection Architecture (cont'd)

- **This architecture can be expressed using a language for describing production systems (or expert systems).**
  - E.g. OPS5.
  - A program consists of if-then (condition-action) rules in this type of languages.
- **However, OPS5-like language does not have a method of structuring rule sets.**
  - A new language should be developed.

Label-connection Architecture (cont'd)

- **An example using a new language**

```
MODULE ef_ingress IS
    RULE SET Classification, Metering, Marking1, Discarding, Marking2,
            Scheduling;
    RULE SET ORDER
        Classification –> Metering, Marking2;
        Metering –> Marking1, Discarding;
        Marking1, Discarding, Marking2 –> Scheduling;
    RULE SET Classification IS
        IF Source_ip == 192.168.1.* THEN
            Label = C1;
        OTHERWISE
            Label = C2;
        …
    RULE SET Scheduling IS
        IF true THEN    // This scheduler is always used.
            Algorithm = priority;
END ef_ingress;
```

Comparisons

■ Major differences between the two architectures:

	Label-connection architecture	Pipe-connection architecture
Rule structures	● If-then rule	● Generalized structure*
Control-flow specification	● Policy order must be specified.	● Not necessary. (Dataflow decides control flow.)
Multiple I/O ports and modularity	● Always one input and one output ports. ● Number of ports does not change by rule addition.	● Multiple I/O ports are sometimes required. ● Ports must be added when rules are added.
Tag usage	● Same tag can be used multiple times. ● Multiple tags for a packet. ● BBs can be more fine-grained.	● Each pipe must have a unique label. ● Only one tag for a pipe. ● BBs can be less fine-grained.
Parallelism	● BB execution semantics is sequential.	● BB execution semantics is parallel.

* A rule consists of guard (~ condition) and body (~ action).

Conclusion

■ Two rule-based building-block architectures for modeling networking policies has been developed.

- ◆ Pipe-connection architecture
- ◆ Label-connection architecture

■ Currently, only label-connection architecture is feasible.

■ Pipe-connection architecture

- ◆ is better in parallelism, which is very important for networking, and
- ◆ has clearer semantics.

■ We should continue studying pipe-connection architecture.

- ◆ It will be a better solution.