Introduction

- **Policy-based QoS-control**
  - Policy rules:
    Administrator or Operator $\rightarrow$ Policy Server $\rightarrow$ Network Nodes
  - Interfaces between Policy Servers and Network Nodes
    - Protocols: SNMP, COPS, …
    - APIs: CORBA, …

[Diagram showing the flow of policy rules from an Administrator or Operator to Policy Server, and then to Network Nodes, with interfaces using protocols like SNMP, COPS, and CORBA.]
A problem in the policy-control interfaces

- The grammar is limited.
  - Grammar = Syntax + Semantics
  - The syntax is limited.
    - E.g., Only one value can be sent at a time by SNMP.
    - E.g., Only function calls can be described by an API.
  - The semantics is limited.
    - Control structures cannot be expressed.
    - Constraints cannot be expressed.

A Solution

- To use a *rule-based programming language* for the interface.

- The reasons
  - A *language* is defined by a grammar.
    - Our problem is limitation of grammar.
  - Policy-based control is a matter of *programming*.
    - A set of policies is a program, because policies describe the behavior of network nodes.
  - Policies are *rule-based*.
    - A policy rule is (should be) an if-then rule.
A Building-Block Architecture

- Each building block has
  - One or more input ports.
  - One or more output ports.
  - An input and output ports are connected by a named pipe.

- An example: a Diffserv router setting

```
SNAP: A Parallel Logic Language

- An example
  - An informal description of a rule:
    ```
    if (Source_IP == 192.168.0.1 && Source_port == 80)
    DSCP = 46;
    ```
  - The rule in SNAP:
    ```
    filter_mark(Si, So) :-
      filter[Source_IP = 192.168.0.1, Source_port = 80](Si, S1) |
      mark[DSCP = 46](S1, So).
    ```

- Building-block syntax
  - Basic syntax: `bb_name(Si, So)`.
  - General case:
    `class_name[parameters](Si1, Si2, ..., Sin, So1, So2, ..., Som)`. 
SNAP: A Parallel Logic Language (cont’d)

- SNAP is a descendant of parallel logic languages
  - Parallel logic languages: Parlog, Concurrent Prolog, GHC, …
    - Developed in 5th Generation Computer Project or similar projects.
  - The reasons why a parallel logic language is used:
    - Rule-based.
    - Well-defined semantics suited for pipelined processing.

- Conditional control using a case structure:
  \[ \text{or}( c_1 \mid a_1 ; c_2 \mid a_2 ; \ldots ; c_n \mid a_n ) \]
  - Control structure is necessary because rules are structured:
    - Not like this, but like this

Conclusion

- A rule-based language SNAP has been introduced.
  - SNAP is for the interface between a policy server and network nodes.

- SNAP enables
  - Definition of building blocks by users
    - Using primitive building blocks and case structures.
  - Interoperable policy-based QoS control.
  - Expressing wide variety of QoS functions.

- Future work
  - Detailed specification of SNAP.
  - Implementation of SNAP on a policy server and routers.
SNAP: A Parallel Logic Language (cont’d)

- An informal description of a rule
  - if (SrcIP == 192.168.1.*) {
    if (average_rate <= 1Mbps) {
      dscp = 46; queue_priority = high;
    } else {
      discard;
    }
  } else {
    dscp = 0; queue_priority = low;
  }

- The rule in SNAP
  - ef_ingress(Si, So) :-
    or(filter[Srclp = 192.168.1.*](Si, C1) |
      or(meter[Average_rate_max = 1Mbps](C1, P1) |
        mark[DSCP = 46](P1, M1)
      ; otherwise(C1, P2) | discard(P2) )
    ; otherwise(Si, Sother) |
      mark[DSCP = 0](Sother, Sother1) ,
      schedule[Algorithm = priority](M1, Sother1, So).

Building Blocks for Diffserv

- Six types of primitive building blocks
  - Filtering, Metering, Marking, Discarding, Scheduling, and Merging (MUX) rules.

- A control structure and constraints
  - Order of building blocks: