Eliot

A Tool for Generating Code from Declarative Fault Models

Author: Samuel Štefánik Supervisor: Ing. Michal Rozsíval

Excel@FIT 2026

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Declaratively describe **network behavior** → Automatically generate fault **injection scenarios**

Processing pipeline

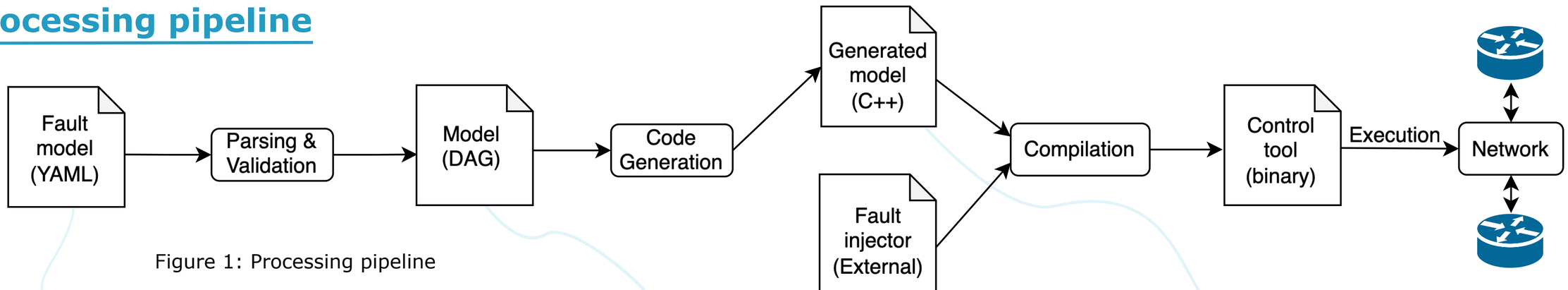

Figure 1: Processing pipeline

YAML

```yaml
root:
  condition:
    type: Payload
    pattern: DROP
  if_true:
    action:
      type: SetState
      target: S
      state: DELAY
    next:
      action: { type: Drop }
  if_false:
    id: S
    initial: ACTIVE
    transitions:
      ACTIVE:
        action: { type: Finish }
      DELAY:
        action: { type: Delay, n: 200 }
        next:
          action: { type: Finish }
```

DAG example

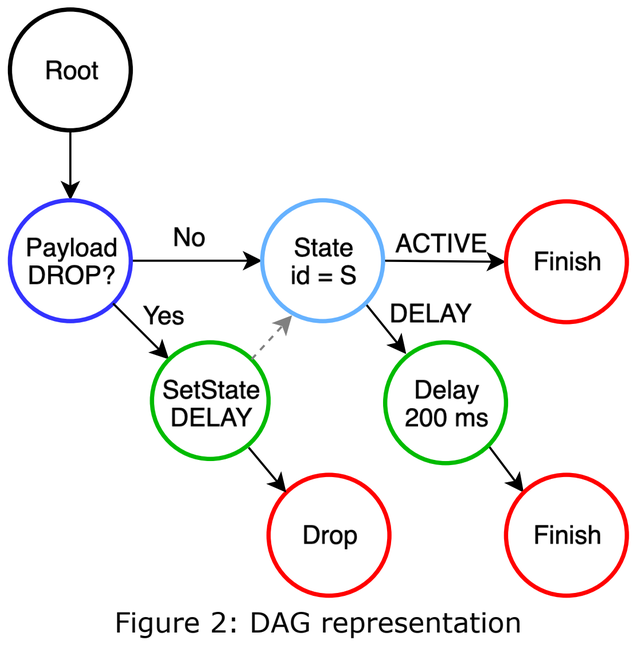

Figure 2: DAG representation

Generated code
(simplified pseudocode)

```
state = ROOT
while (processing)
    switch state:
        case ROOT:
            if condition(Payload, Packet):
                state = SET
            else:
                state = STATE
        case SET:
            execute_action(SetState, Packet)
            state = DROP
        case STATE:
            state = resolve_state(S)
        case ACTIVE:
            execute_action(Finish, Packet)
        case DELAY:
            execute_action(Delay, Packet)
            state = FINISH
        case DROP:
            execute_action(Drop, Packet)
```

Functionality

Conditions:
- Packet filtering rules
- Stateless & stateful conditions
- Time and propability based

Actions:
- Packet manipulation
- Drop, delay, reorder
- Replicate, throttle, modify

Value Generators:
- Dynamic parameters
- Random distributions
- Time & sequence based

State Nodes:
- Behavior depends on internal state
- Enables complex behavior
- Internal state updated via actions

Processing loop

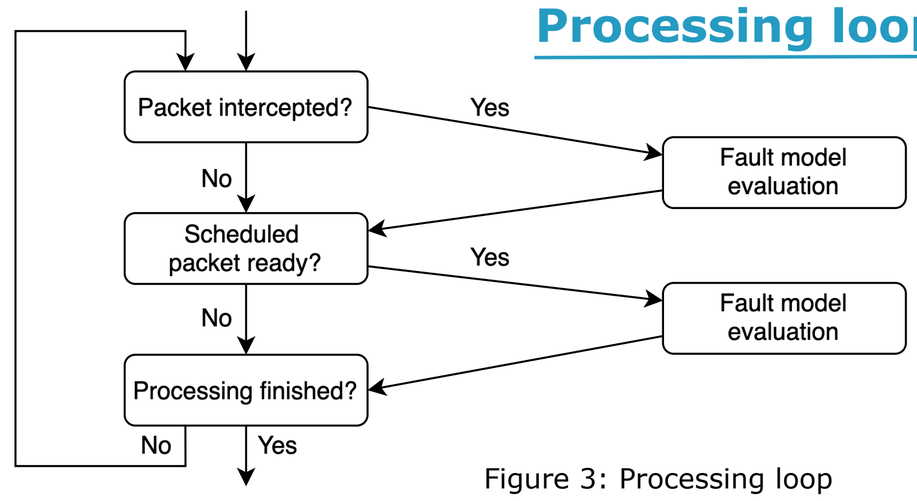

Figure 3: Processing loop

Packet Scheduling

Concept

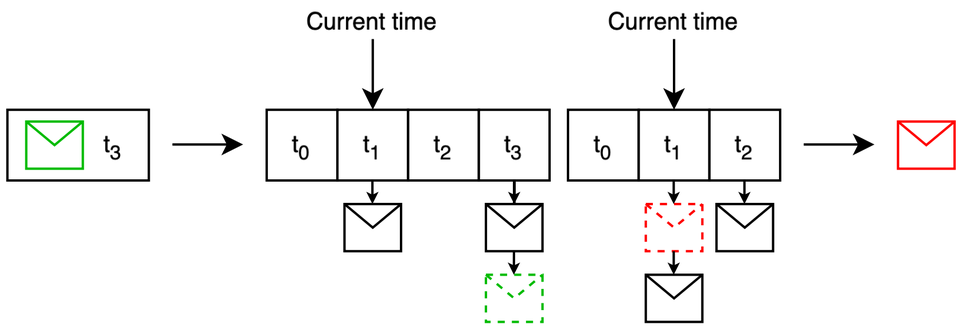

Figure 4: Packet scheduling concept

Implementation

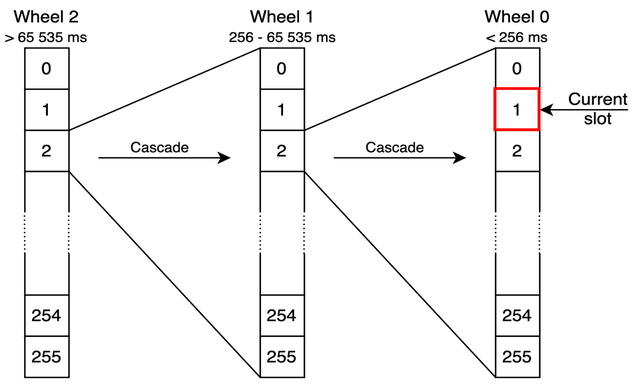

Figure 5: Hierarchical timer wheel structure

Technologies

Performance evaluation

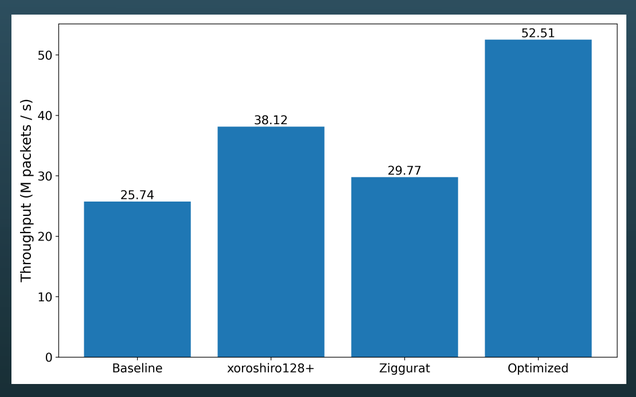

Figure 6: Impact of number generation strategies

Results:
- No runtime interpretation overhead
- Deterministic and reproducible behavior
- Up to 2.6x higher throughput

Implementation	Throughput (M packets / s)	Speedup
Baseline	2.23	1.0×
Optimized	5.97	2.68×

Table 1: Overall performance comparison

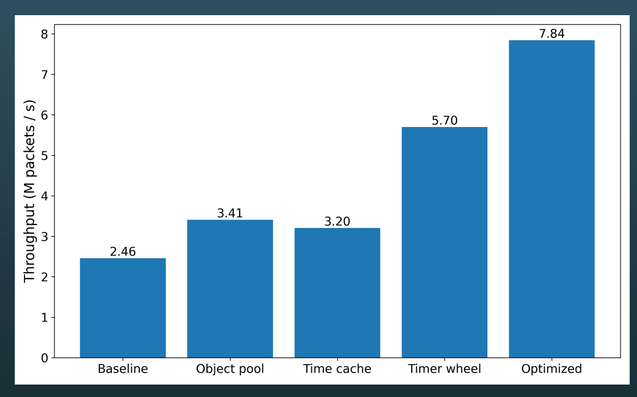

Figure 7: Impact of delay optimizations