What P4 Can Learn From Linux Traffic Control Architecture

Jamal Hadi Salim
Mojatatu Networks
Intro To Linux TC
Intro To Linux TC

We define **Network Service** as: *The treatment of selected network packets, as defined by a user policy, so as to achieve a defined goal on the selected packets.*

The TC architecture is a **Network Service Infrastructure**

- Has been around since late 90s

Functional Block Types are abstracted to allow composition of *policy graph(s)* to achieve a **Network Service**

4 Functional Block Types

1. **Qdiscs** provide templating for queue algorithms (enqueuing and dequeuing packets)
2. **Classifiers** provide templates that define filtering algorithms (to discriminate/select packets)
3. **Actions** provide templating for arbitrary packet processing
4. **Classes** provide templating for encapsulating qdisc FBTs to allow service topology branching
Intro To Linux TC: Functional Block Types

Some Qdisc Kinds:
- Pfifo which implements a basic packet counting FIFO queueing algorithm.
- RED which implements the Random Early Detection (RED) algorithm.
- DRR which implements the Deficit Round Robin (DRR) algorithm.

Some Action Kinds:
- gact which implements amongst other things dropping and accepting of packets.
- mirred which implements redirecting or mirroring packets.
- skbedit which implements metadata editing on a packet.
- pedit which implements arbitrary packet editing

Some Classifier Kinds:
- u32 which implements a 32-bit key/mask (ternary, lpm and exact) matching algorithm.
- flower which implements a multi-tuple matching algorithm.
- fw which implements a (skbmark) metadata based matching algorithm.
- Others implementing string matching, ebpf etc etc

Class Kinds
- Classes provide templating for encapsulating qdisc FBTs to allow service topology branching. On their own classes do not implement algorithms, so there is only one kind.
Intro To Linux TC: Policy Graphs

All FBT instances have:
- A 32 bit node id used as graph vertex id
  - In a tree graph a parent id as well
- A control interface
  - Each node is configured individually

A Service graph anchored at a location
- FBT node instances are composed to form a service using node IDs
Intro To Linux TC: Policy Graph Anchors

To build a TC policy topology we need a root/start node ID (associated with a port/netdev)

- An ID of 0xFFFFFFFF is reserved for use as a handle for the anchor point of the EGRESS topology.
- An ID of 0xFFFFFFFF3 is reserved for use as a handle on the egress anchor point for the EGRESSCLSACT topology.
- An ID of 0xFFFFFFFF1 is reserved for use as a handle for the anchor point of the INGRESS topology.
- An ID of 0xFFFFFFFF2 is reserved for use as a handle for the INGRESSCLSACT topology.
- More could be added at different stack points
Intro To Linux TC Qdisc Subsystem
EGRESS Service Topology

Policy graph nodes composed of:
- Classifiers
- Actions
- Queueing algorithms
- Scheduling algorithms

Policy scripting BNF grammar via the tc utility
- It is possible to describe more than match-action
- Policy not part of datapath program \(\text{apply(\_)}\)
  - Graph composition of different nodes done in the control plane
Sample EGRESS Service Topology
Intro To Linux TC Classifier Action Subsystem
Basic Classifier Action Chain/Pipeline

Multi Classifier types in a chain
- Multi tuple (flower)
- Raw OLV matcher (u32)
- String matches, etc

- Pipeline in priority order
- Dynamic runtime control (as opposed to static compile time)
  - Add, remove and reroute CA blocks
  - Add, remove and reroute Actions
- Action Block Result opcodes dictate exec path
More Complex Classifier Action Pipeline

Each classifier match keyed by 
\{protocol, priority, header\}
- **Lowest priority is default**
  - No need for special Default matches

**TC CA Blocks shareable**
- **Across ingress, egress +port**
  - P4 MA can only exist within a control block
Peeking into a Classifier Action Block

Multiple Actions per match rule

**OPCODES** are
1. programmed into the actions
2. generated by the actions based on runtime conditions

Each action can act on the whole packet
- Consider an action that does packet compression for example
  - P4 deals with headers only?
  - Means activity where the whole packet is processed requires redirection to an external device?
Actions Runtime Implementation vs Abstraction

More OPCODES: **REPEAT, PIPE, JUMPX**
Allows programming control abstraction
- if/else/elseif/while/goto
Peeking Into Actions Implementation

Actions are abstracted as **indexed tables**
- Each action has one table per instance

Control instantiates action table rows with desired attributes
- When specifying the actions with matches (**by value as in P4 semantics**)
- Independently then binding to matches (**by reference**)
Peeking Into A Classifier Action Block

Matches point to an ordered list of actions
- From a table perspective actions are referred to using a foreign key
- From a s/w implementation perspective they are pointers to the action info structures
Action Sharing

Because actions are referenced by their \{type id, index\} they can be shared by multiple matches
How TC Can Help P4
Suggestions: Modularity And Policy Control

Allow for decomposable construction of match-action
- Runtime binding
- Independent upgrades and maintenance
  - Add a new action without recompiling the P4 program

Q: How difficult would it be to have *hardware* implement dispatchers for Classifier-Action?

Move `apply()` out to control plane
- New policy language? tc cli has a BNF grammar that would be a good start
- Graph policy definition of the different constructs
  - Independent policy updates
Suggestions: Traffic Management

Schedulers and enqueue algorithms
  • Is PIFO sufficient?

Hierarchical construction
  • Possible if TC graph abstraction is adopted
Suggestions: Multiple Actions Per Match

Doable with an action dispatch loop
Suggestions: Sharing Of Tables And Actions

TC supports Match-Action blocks to be shared on different controls
- Achievable on P4 hardware?

TC supports sharing of actions across controls
- P4 already supports it for meters and counters
  - Just need to make it generic for all actions
Suggestions: Event Modelling

Not sure how well to define eventing to controller
- TC kernel allows to notify subscribers of datapath and control activities (table changes etc)
Back Slides: Sample Service Topologies
EGRESS Classless Service Topology

Very simple service topology
- No matches or actions
  - Implicit metadata classification
- Anchored at Egress of a port/netdev
EGRESS Classful Service Topology
EGRESS Complex Classful Service
EGRESS Clsact Service Topology
INGRESS Service Topology

Diagram showing the flow from Port Ingress to Ingress qdisc, then to Classifiers, and finally to Network Stack. The Classifiers block has three Action Actions connected to it.
INGRESS To Egress Service Topology