Implementing Stream Operations in P4

The writeup is devided into three sections: overview, implementation, and application.

Overview

Stream operations are query operations tailored towards streams of data. There are multiple representations of streams of data, however to simplify implementation I use schema-key-value tuples.

I used *The CQL Continuous Query Language: Semantic Foundations and Query Execution* by Arvind Arasu, et al. as a guide to chose stream relation operators. I chose the following building block stream operators. I provide example queries that make use of the stream operators – the current system does not have the full functionality necessary to compute these queries, however, the analgous operator logic has been implemented.

- **Map** - `Select speed/2 From PosSpeedStr`
- **Filter** - `Select Istream(*) From PosSpeedStr [Range Unbounded] Where speed > 65`
- **Zip** - `Select pos1 + pos2 From PosSpeedStr1 [Now], PosSpeedStr2[Now]`
- **Join** - `Select Distinct L.vehicleId, L.segNo, L.dir, L.hwy From SegSpeedStr [Range 30 Seconds] as A, SegSpeedStr [Partition by vehicleId Rows 1] as L Where A.vehicleId = L.vehicleId`
- **Aggregate** - `Select segNo, dir, hwy From SegSpeedStr [Range 5 Minutes] Group By segNo, dir, hwy`

With these operations, generic stream queries should be able to be computed on the router.

Queries are defined per schema by a developer. The queries should then be translated to stream operators (the above 5 operators) and then sent to routers.

Implementation

The stream query packets are a stack of tuples (schema, key, value). Initially, each packet would be a single event in the stream, however, difficulties with the P4 compiler necissitated that I use a stack.

Currently queries are computed on a single router, though, further implementation could distributed computations across routers. Currently there is no distributed computing protocol, so, for example aggregate operators are done on a per router basis.

Operators

All stream operators have been implemented in P4. Operators that need to buffer data use a register to store historical data.

Map

Map is implemented as a single operation that changes the value or key of the key-value pair.

Filter

Drops a packet if a the key or value matches some value.

Zip

For a stream `(k_1, v_1), (k_2, v_2), ..., (k_n, v_n)` of schema `s` and a stream `(k'_1, v'_1), (k'_2, v'_2), ..., (k'_n, v'_n)` for schema `s'` a zip operation can be defined with function `f`, `g` to ouput a zipped stream `(f(k_1, k'_1), g(v_1, v'_1)), (f(k_2, k'_2), g(v_2, v'_2)), ..., (f(k_n, k'_n), g(v_n, v'_n))`. Implemented using a buffer of key-value pairs for a schema.

Join

Computes a join for a schema where some join condition, `c`. The join is computed across buffered historical tuples for a stream and a join function is applied to joined tuples to generate the new value.

Aggregate

Computes an aggregate using some aggregate function over the buffered historical values for a schema-key pair.

Operator Compiler

There is no compiler for a standard or continuous QL to the P4 query operators. I have added Python representation of the operators which can then be written to the action table for the switches. A QL compiler would simply to to output the queries in this python representation to install the queries on the routers.

Applications

Arvind Arasu, et al. introduce dynamic tolling for traffic systems as an application of streaming queries. This application also happens to be a good application of a network based query operations.

Autonomous cars have the potential to drive more efficiently, in part, because they can share information between cars. Some of this information will be specific to the location of the car (eg known road hazards, congestion statistics). Rather than using a central database to serve requests about this data, some of these queries can be directly processed within the network.