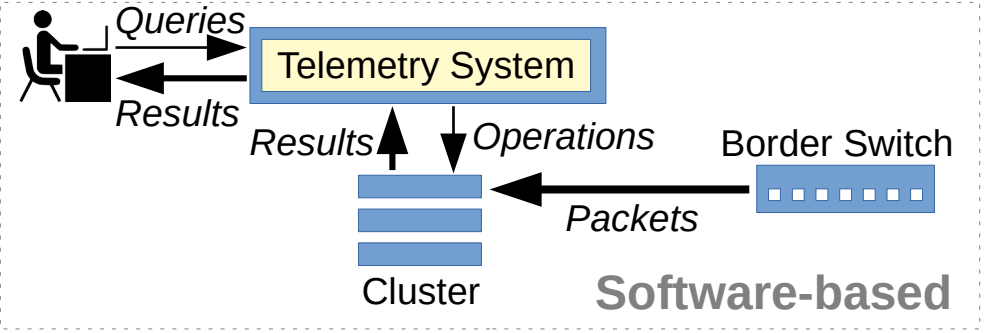
Dynamic Scheduling of Approximate Telemetry Queries with DynATOS

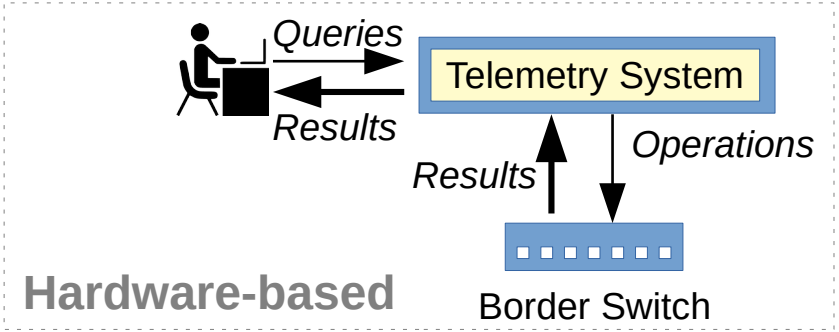
Chris Misa
Walt O' Connor
Ramakrishnan Durairajan
Reza Rejaie
Department of Computer Science, University of Oregon

Walter Willinger — NIKSUN, Inc.

Telemetry for Traffic Monitoring

- Telemetry systems give operators visibility into network traffic.
  - Operators submit queries, telemetry system returns results.

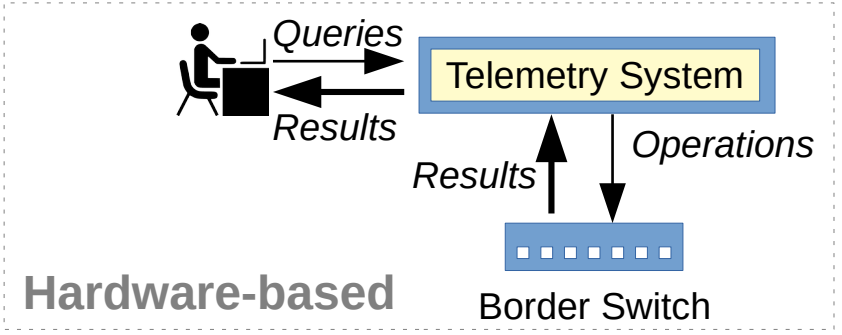




➔ **How to evaluate query operations over massive volumes of traffic data in realtime?**

Telemetry for Traffic Monitoring

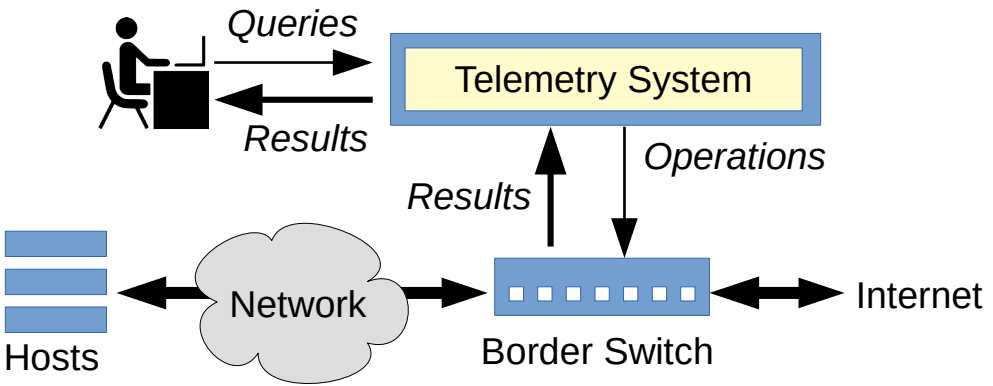
- Switch hardware reduces costs by orders of magnitude.
  - Queries happen in-network reducing export volume.



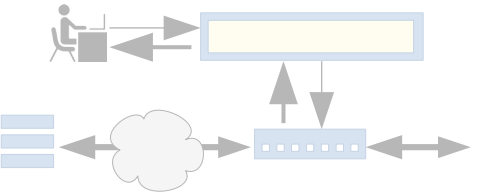
Sonata (SIGCOMM 2018), *Jaqen* (USENIX Sec. 2021)

Illustrative Example

- Suppose we need to monitor traffic on an enterprise or campus network.
  - Don’t have admin access to end hosts.
  - Go for a single deployment at the border:



Illustrative Example



- Suppose we want to run two queries in parallel:

Query

```
pkts.distinct
      (ipv4.src,ipv4.dst)
    .keyby (ipv4.dst)
    .reduce (count)
    .filter (>= T1)
```

Detect DDoS Victims

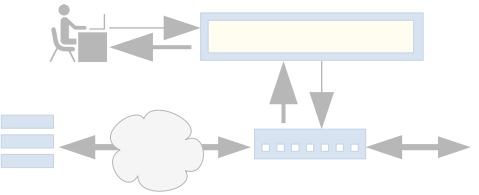
```
pkts.distinct
      (ipv4.src,tcp.dport)
    .keyby (ipv4.src)
    .reduce (count)
    .filter (>= T2)
```

Detect Port-scanning Hosts

Operations

Queries from *Sonata* (SIGCOMM ’18).

Illustrative Example



- Suppose we want to run two queries in parallel:

```
pkts.distinct
      (ipv4.src,ipv4.dst)
    .keyby (ipv4.dst)
    .reduce (count)
    .filter (>= T1)
```

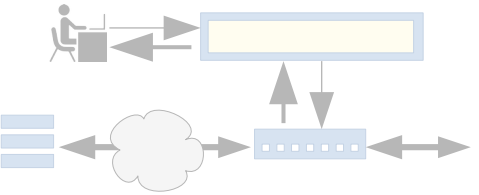
Detect DDoS Victims

```
pkts.distinct
      (ipv4.src,tcp.dport)
    .keyby (ipv4.src)
    .reduce (count)
    .filter (>= T2)
```

Detect Port-scanning Hosts

- Queries emit list of **results** (addrs.) each **epoch** (e.g., 1 s).

Illustrative Example



- Suppose we want to run two queries in parallel:

```
pkts.distinct
      (ipv4.src,ipv4.dst)
    .keyby (ipv4.dst)
    .reduce (count)
    .filter (>= T1)
```

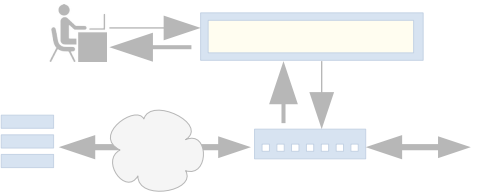
Detect DDoS Victims

```
pkts.distinct
      (ipv4.src,tcp.dport)
    .keyby (ipv4.src)
    .reduce (count)
    .filter (>= T2)
```

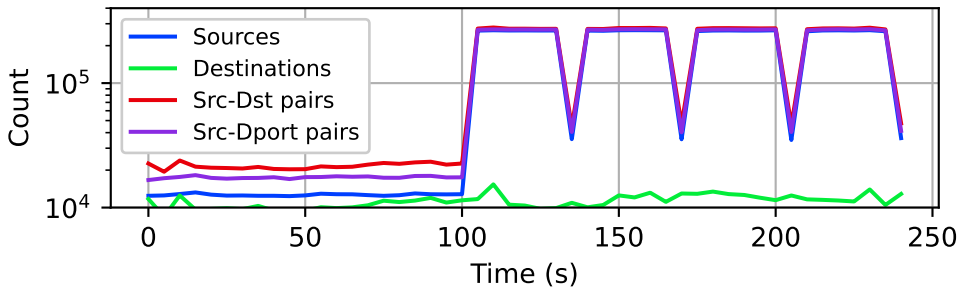
Detect Port-scanning Hosts

- Queries emit list of **results** (addrs.) each **epoch** (e.g., 1 s).
- Hardware resources required for both queries depends on number of distinct sources…

Illustrative Example



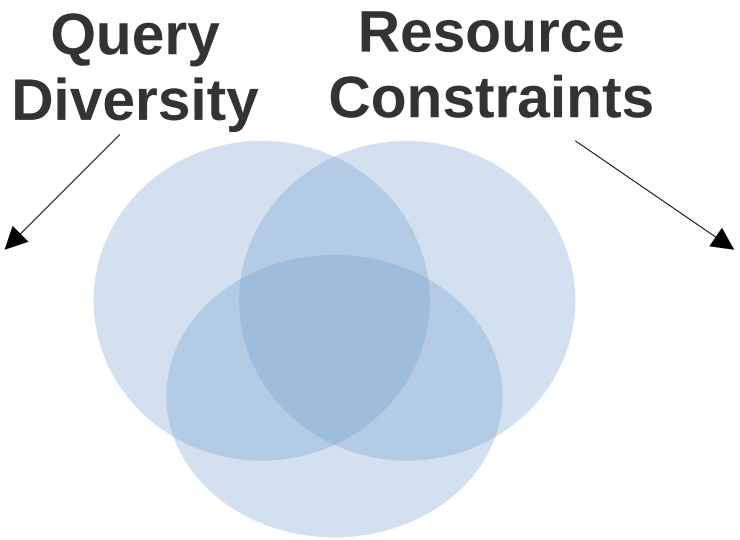
- # of distinct elements is stable for a while, then changes.



Example traffic from MAWILab on 2015-11-14 starting at 2pm.

- The resources required for both queries also change.
- Must update allocations or face degraded accuracy.

Key Requirements



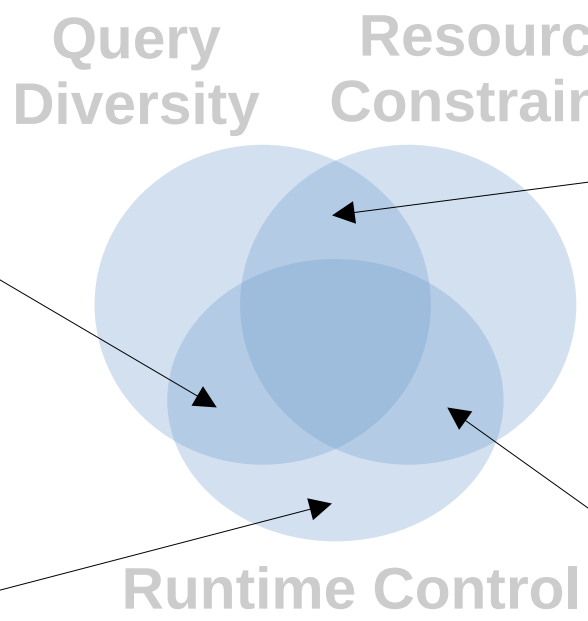
- Use parameterized dataflow operators to specify user-defined **filters**, **aggregates**, and **summaries**.

- Reduce volume of results from switch.
- Fit aggregations in limited switch memory.

- Change queries and resource allocations at runtime.

Related Works

Newton (CoNEXT ‘20): Change queries without recompiling.



SketchLib (NSDI ‘22): Implement diverse queries on hardware.

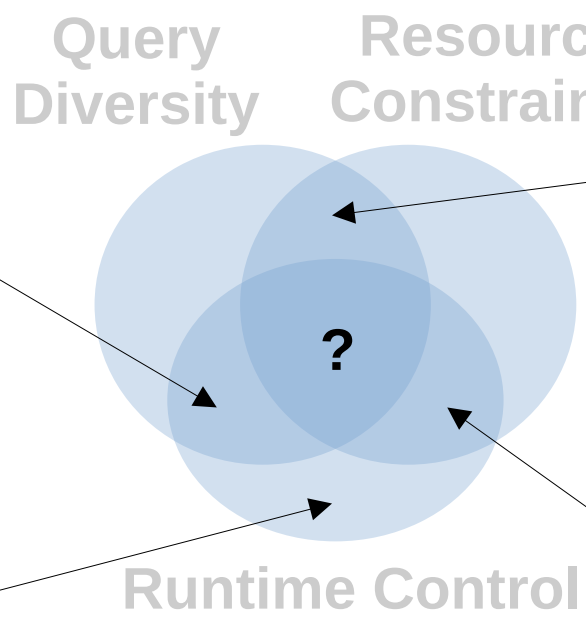
FlexCore, IPSA (NSDI ‘22)

ElasticSketch (SIGCOMM ‘18): Adapt single query based on load.

Related Works

Newton (CoNEXT ‘20): Change queries without recompiling.

SketchLib (NSDI ‘22): Implement diverse queries on hardware.

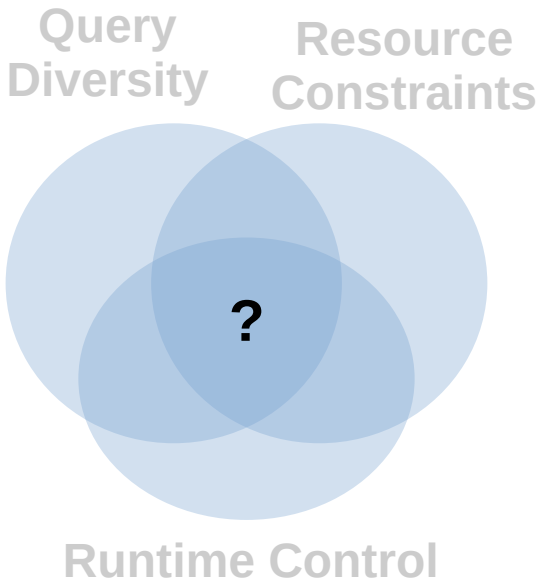


FlexCore, IPSA (NSDI ‘22)

ElasticSketch (SIGCOMM ‘18): Adapt single query based on load.

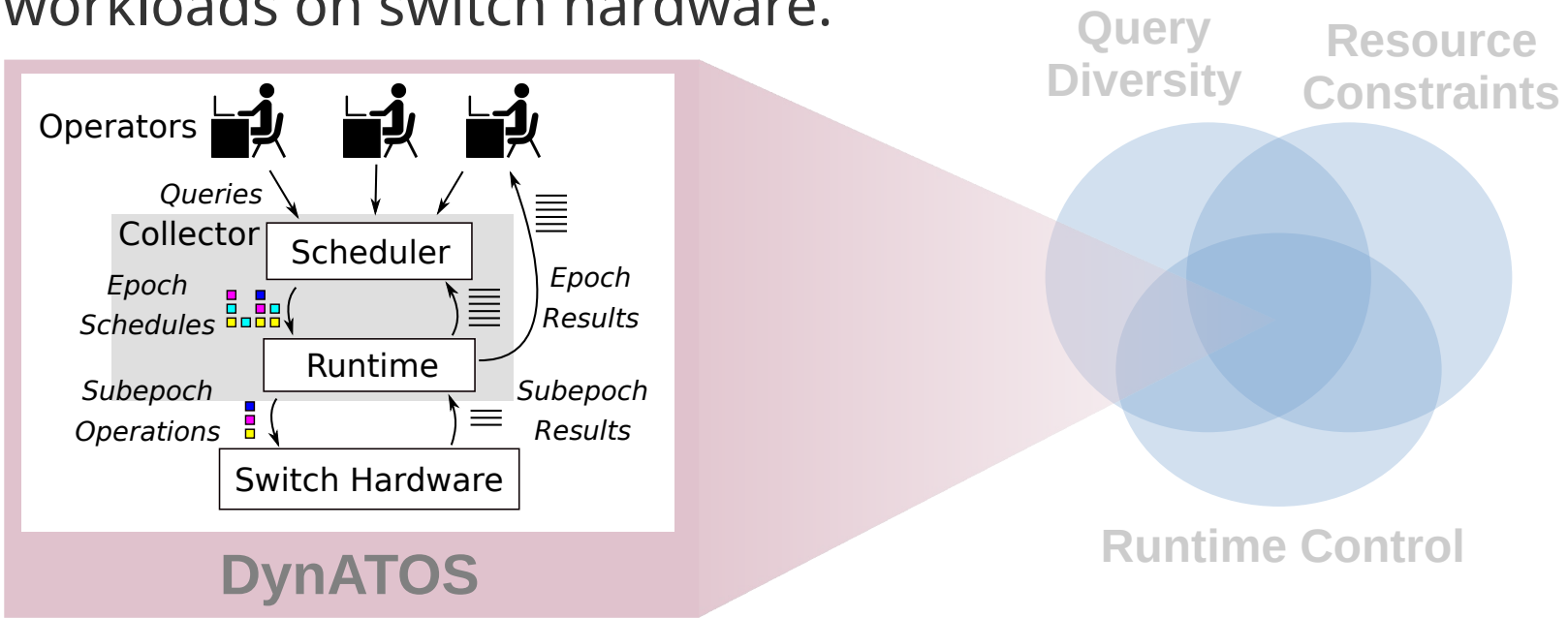
Design Challenges

- **D1:** Need to approximate generic query operations;
- **D2:** Need dynamic accuracy estimation *without assumptions about traffic*; and
- **D3:** Need to schedule finite hardware resources among dynamic query workload.



Our Approach - DynATOS

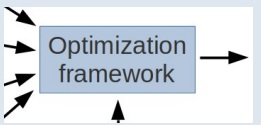
- Build a closed-loop system to schedule dynamic query workloads on switch hardware.



Key Insights

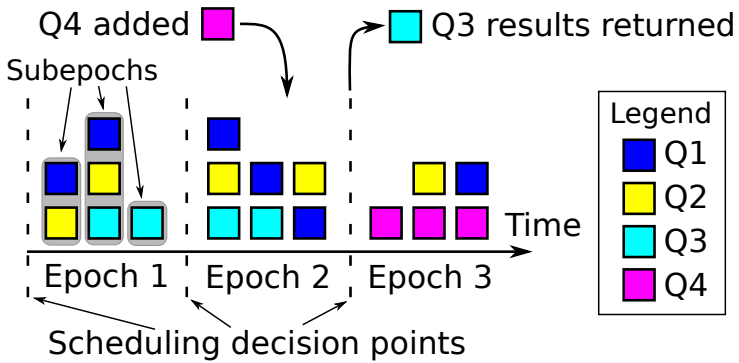
- *(1)* ***Time-division approximation:***
  - Enables tradeoffs to reduce resource requirements (**D1**);
  - Can estimate error based on observations alone (**D2**).

- *(2)* ***Scheduling formulation:***
  - Realizes tradeoffs enabled by time-division approximation (**D3**);
  - Efficiently decides how to execute query operations.



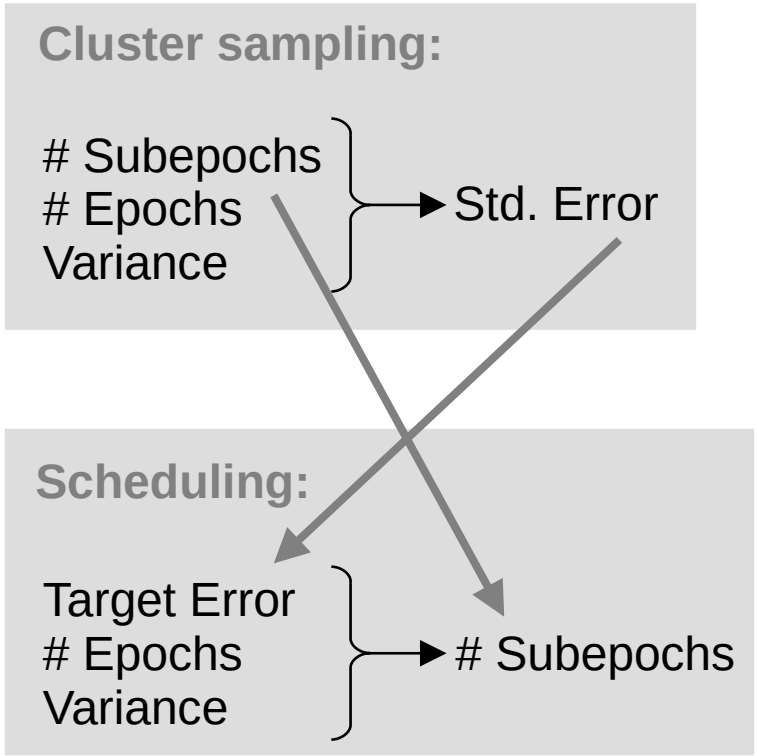
(1) Time-Division Approximation

- Observation: query operations don't need to run all the time.
  - Divide the epoch into sub-epochs and run operations in a subset of sub-epochs.



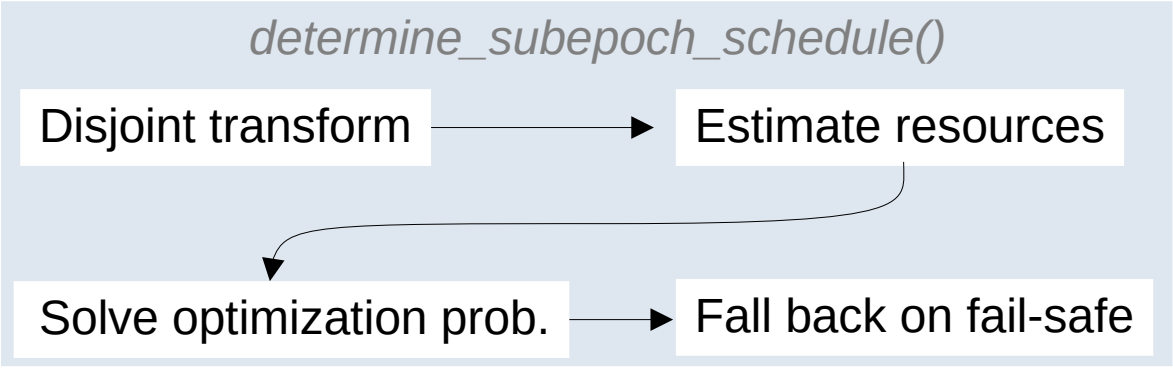
(1) Time-Division Approximation

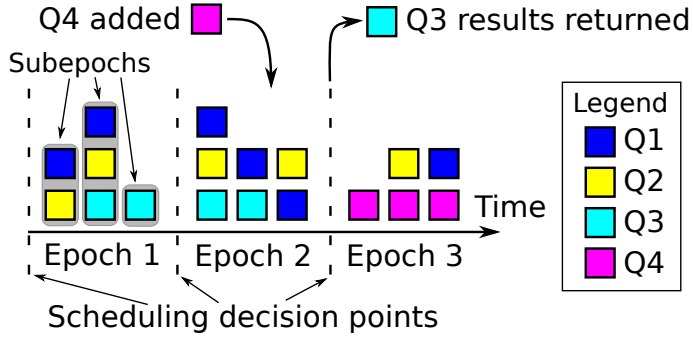
- Cluster sampling gives error bounds.
  - Most inputs are known or estimatable (unlike sketches).
- Enables accuracy tradeof:
  - Predicts how many subepochs are required for a target error bound.
- Slight modification also enables latency tradeoff (see paper).



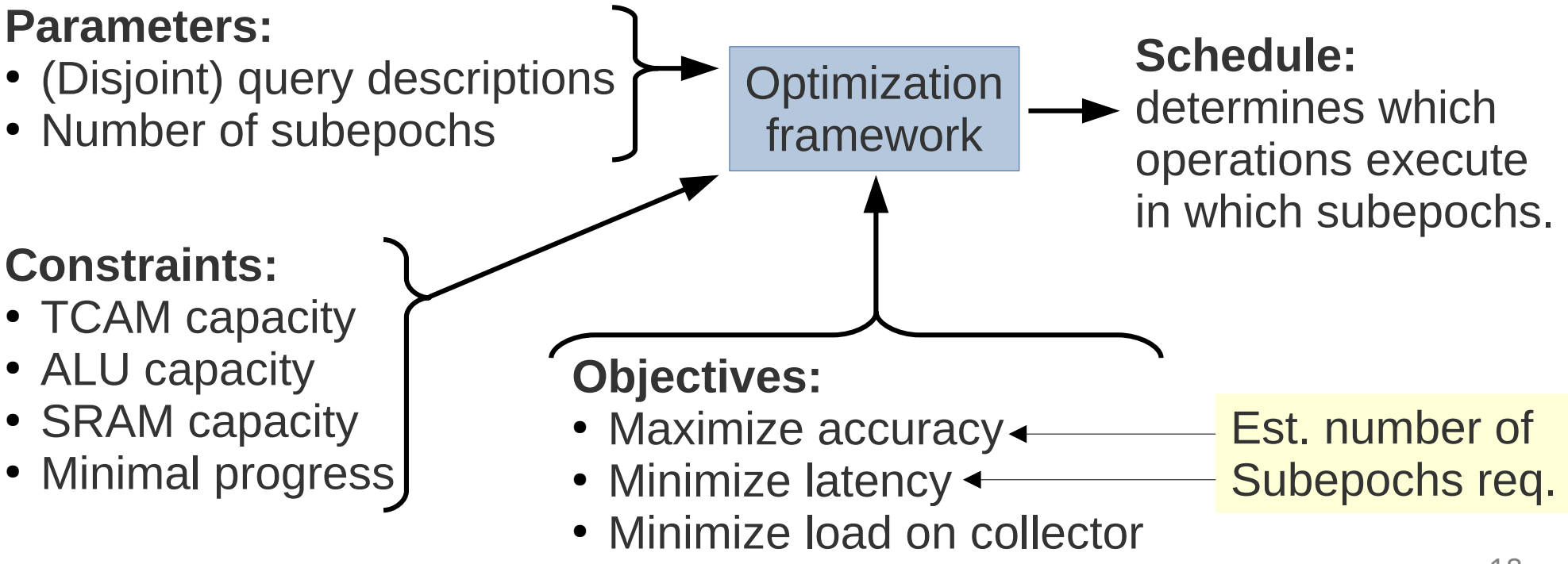
(2) Scheduling Formulation

- Observation: before each epoch, make a fixed scheduling decision for all subepochs.
  - Both deterministic and adaptive.



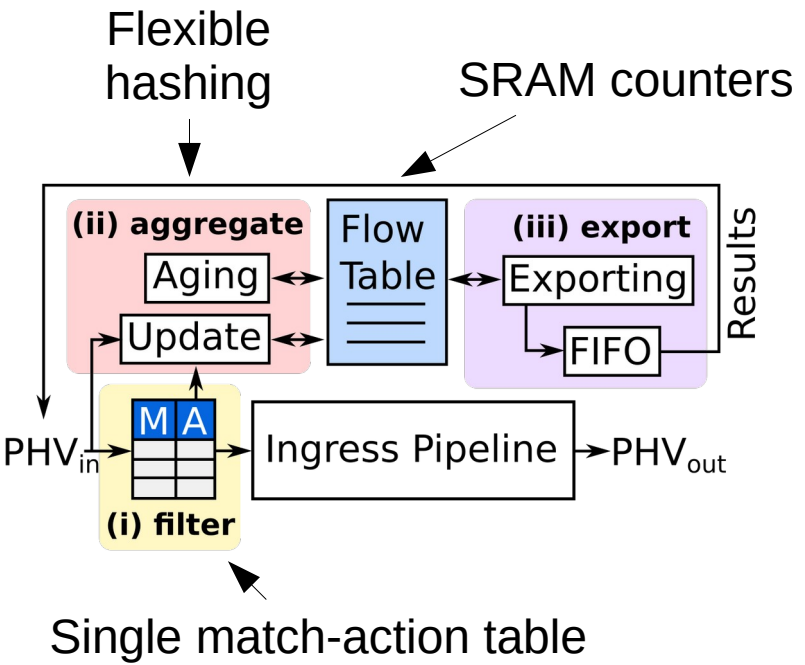


(2) Scheduling Formulation



Implementation on Switch Hardware

- We implement a hardware-based prototype of DynATOS.
- Built around a runtime programmable switch hardware telemetry module.
  - Currently uses Broadcom's BroadScan module.
  - Similar to a single stage in Newton.



All elements are *runtime programmable*!

Evaluation

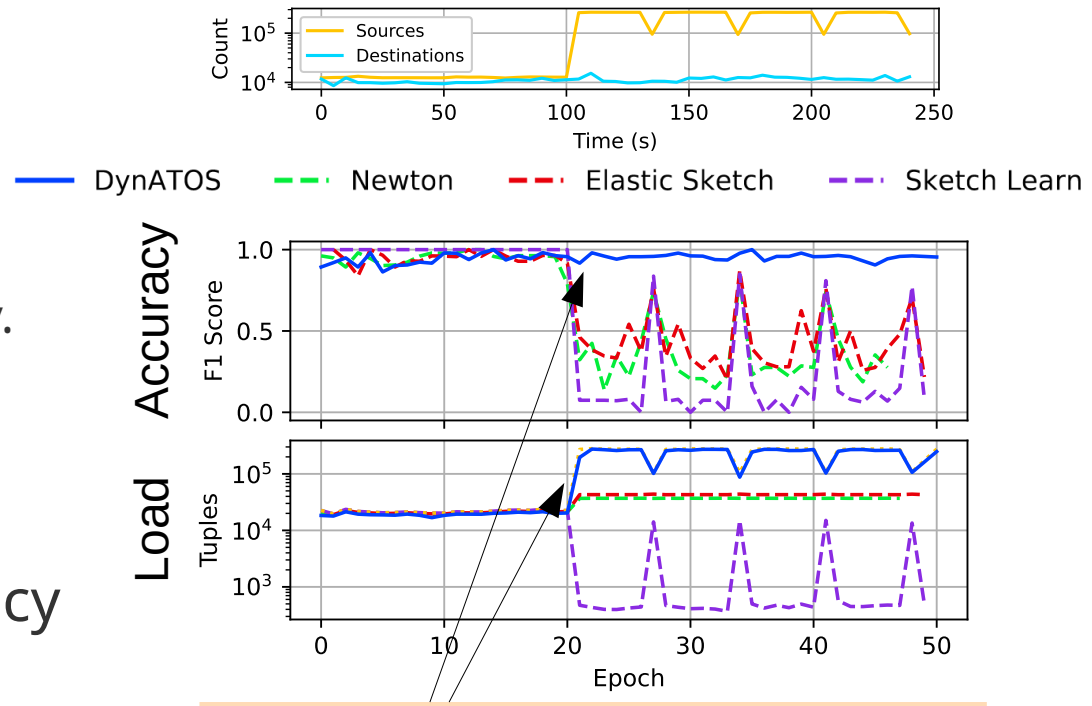
- Performance of time-division approximation w.r.t. sketches:
  - More robust to traffic dynamics;
  - Offers similar accuracy-load tradeoffs.
- Performance of scheduler:
  - Handles dynamic workloads with high satisfaction;
  - Accuracy and latency tradeoffs reduce resource reqs.
- Runtime overheads are minimal.

Evaluation

- Performance of time-division approximation w.r.t. sketches:
  - More robust to traffic dynamics;
  - Offers similar accuracy-load tradeoffs.
- Performance of scheduler:
  - Handles dynamic workloads with high satisfaction;
  - Accuracy and latency tradeoffs reduce resource reqs.
- Runtime overheads are minimal.

Robustness of Time-Division Approx.

- Case study on traffic dynamics.[1]
  - We run Sonata's DDoS query.
- Sketch-based approaches calibrated before increase.
- Measure per-epoch accuracy and load on collector.

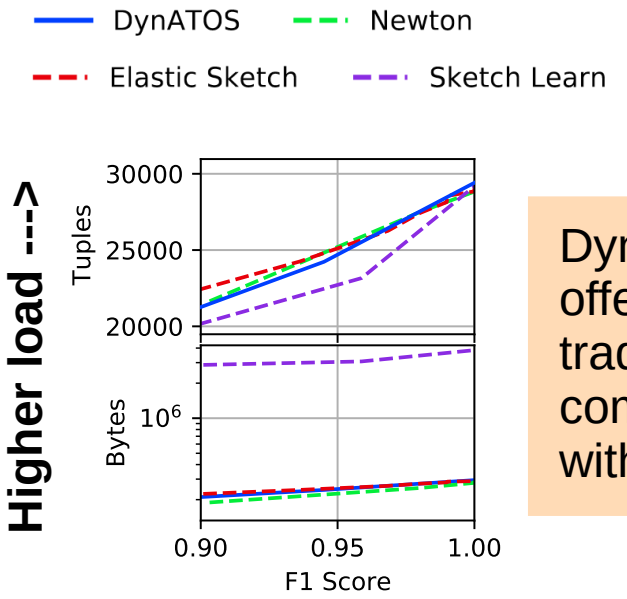


DynATOS maintains high accuracy by adjusting load on collector.

[1] Same packet-level trace from slide 8.

Tradeoffs Compared with Sketches

- Quantify accuracy-load tradeoff for different parameter settings:
  - Sketches adjust # counters;
  - DynATOS adjusts target accuracy (σ).
- Observe accuracy and load on collector for each setting.

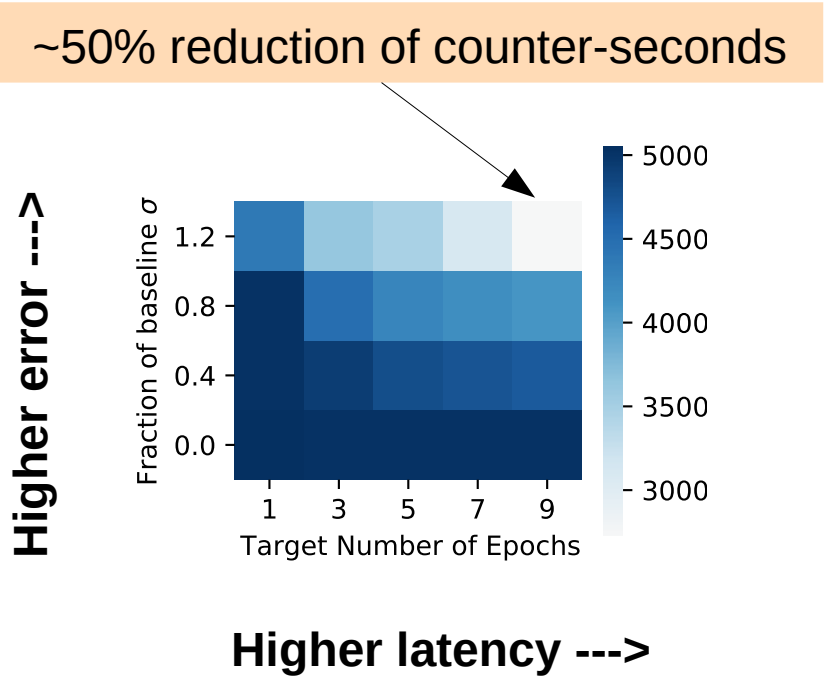


DynATOS offers similar tradeoffs compared with sketches.

Example based on *port scanning* query from Sonata.

Tradeoffs Reduce Resource Requirements

- Quantify effectiveness of accuracy and latency tradeoffs using resource-seconds metrics.
- DynATOS effectively leverages accuracy and latency tradeoffs to reduce resource requirements.



Color intensity shows counter-seconds required for TCP new connections example query.

Summary

- Current approaches don’t handle simultaneous traffic and query dynamics.
- We develop DynATOS to handle traffic and query dynamics with time-division approximation and an optimization-based scheduler.
- Our evaluation shows DynATOS can deal with both traffic and query dynamics while offering similar tradeoffs as state-of-the-art and minimal overheads.

Future Work

- Runtime programmablility motivates new monitoring algorithms:
  - Developing iterative, adaptive, and dynamic queries.
- DynATOS must ultimately be extended to multiple stages and distributed context.
  - Requires combined topology-aware placement and scheduling.

Thanks!

- Special thanks to Shahram Davari and Broadcom, Inc. for providing hardware and technical support for our testbed.
- This work is supported by:

- See onrg.gitlab.io/projects/dynatos/ for more details.

Questions?

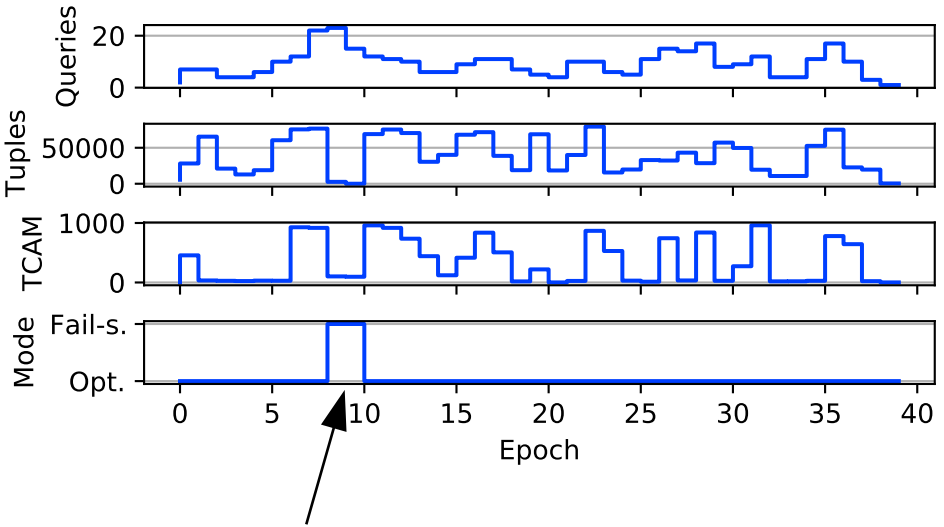
Backup Slides

Evaluation

- Performance of time-division approximation w.r.t. sketches:
  - More robust to traffic dynamics;
  - Offers similar accuracy-load tradeoffs.
- Performance of scheduler:
  - Handles dynamic workloads with high satisfaction;
  - Accuracy and latency tradeoffs reduce resource reqs.
- Runtime overheads are minimal.

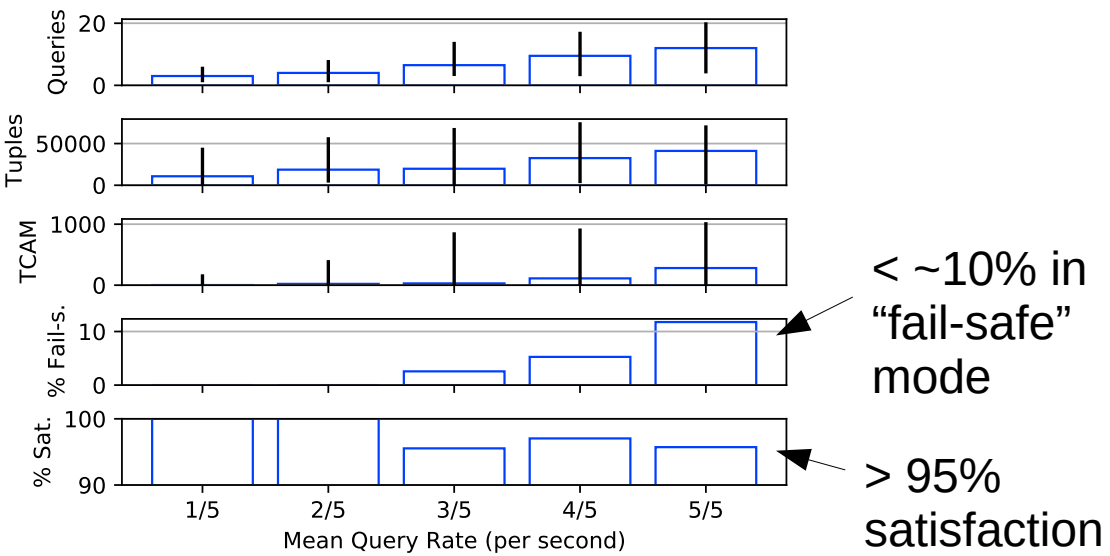
Example Dynamic Query Workload

- Generate query workloads using Poisson query arrivals plus random followup queries.
  - Random overlapping filters and aggregations.
- DynATOS dynamically adjusts resource usage.



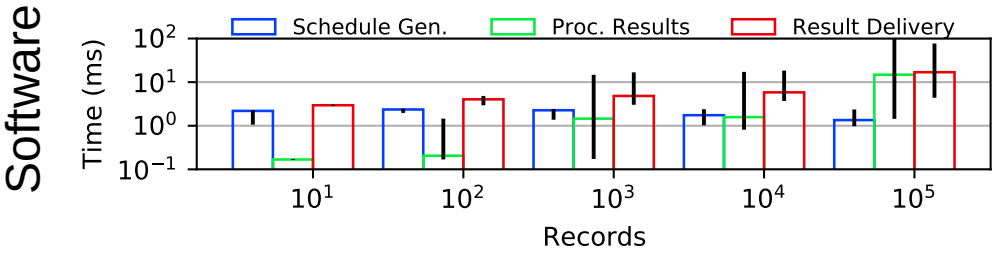
Handling Dynamic Workloads

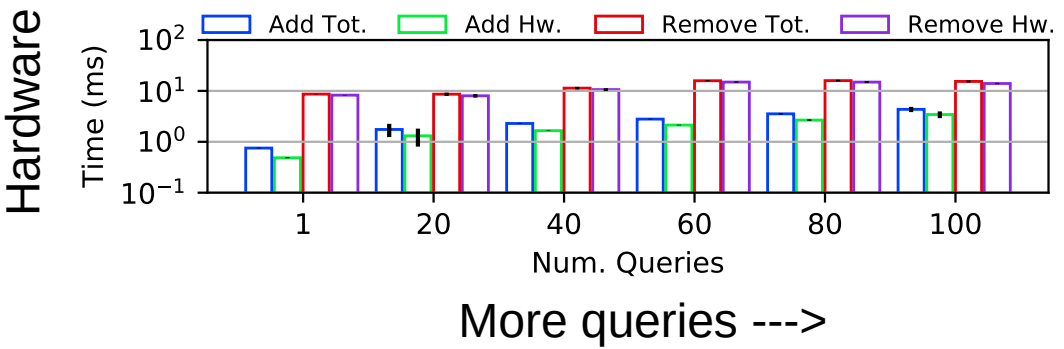
- Use workloads with different arrival rates to evaluate scalability.
- Measure satisfaction as % of queries where SE <= σ.



Minimal Runtime Overheads

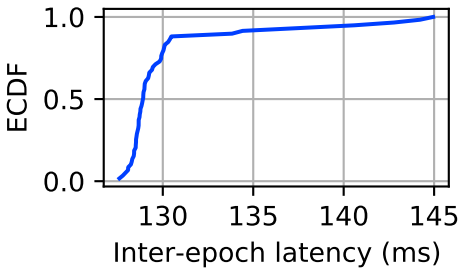
- Measure execution time of selected procedures.
- Software overheads are dominated by result processing.
- Hardware overheads are dominated by resetting tables.



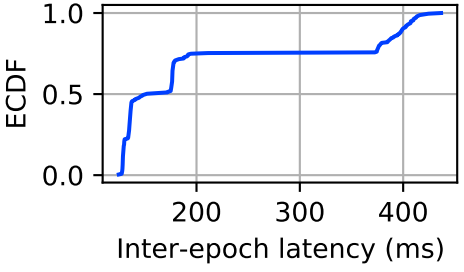


Turnaround Time Distribution

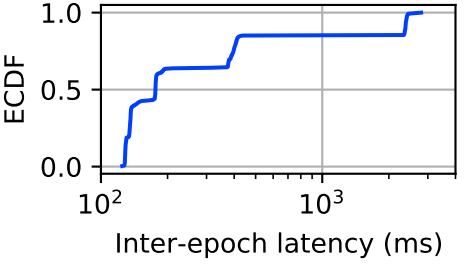
- Inter-epoch latency is around 100 ms on average, but the tail grows when more result records are returned.



10 records



10^4 records



10^5 records

More records --->

Accuracy Tradeoff Formulation

- Cluster sampling provides sound error bounds.
  - Most inputs are known or can be estimated ahead of time (unlike sketches).

$$SE(\hat{t}_E) = \frac{N}{E}\sqrt{\sum_{j=1}^{E}\left(1-\frac{n_j}{N}\right)\frac{s_{t_j}^2}{n_j}}$$

Cluster sampling standard error.

Input: Target Error **Estimate: Variance**

$$n^{acc} = \frac{s_{t_E}^2 N^2}{E^2\sigma^2 - \left(\sum_{j=1}^{E-1}\mathrm{Var}(\hat{t}_j)\right) + N s_{t_E}^2}$$

subepochs required for target error.

Accuracy Tradeoff Formulation

- Use n^{acc} to drive scheduling decisions that trade off accuracy for reduced number of subepochs.

Drives Scheduling

$$SE(\hat{t}_E) = \frac{N}{E}\sqrt{\sum_{j=1}^{E}\left(1-\frac{n_j}{N}\right)\frac{s_{t_j}^2}{n_j}}$$

Cluster sampling standard error.

rearrange

$$n^{acc} = \frac{s_{t_E}^2 N^2}{E^2\sigma^2 - \left(\sum_{j=1}^{E-1}\mathrm{Var}(\hat{t}_j)\right) + N s_{t_E}^2}$$

subepochs required for target error.

Latency Tradeoff Formulation

- We formulate a latency tradeoff by assuming some allocations will happen in future epochs.

$$n^{lat} = \frac{s_{t_E}^2 N^2}{\tilde{E}^2\sigma^2 - N^2\left(past + future\right) + N s_{t_E}^2}$$

$$past = \sum_{j=1}^{e-1}\left(1 - \frac{n_j}{N}\right)\frac{s_{t_j}^2}{n_j}$$

$$future = \sum_{j=e+1}^{E}\left(1 - \frac{n_j}{N}\right)\frac{s_{t_j}^2}{n_j}.$$

cmisa@cs.uoregon.edu

Distinct Count Queries

- We correct distinct-count type queries (e.g., DDoS from Sonata) using the Chao estimator without replacement:

$$\hat{S}_{Chao1,wor} = S_{obs} + \frac{f_1^2}{\frac{n}{n-1}2f_2 + \frac{q}{1-q}f_1}$$

where S_{obs} is the number of elements observed in the sample, f_1 is the number of elements observed once, f_2 is the number of elements observed twice, and q is the sampling rate.

Scheduling Optimization Problem

Variable	Description
Q	index set of queries ready for execution
SE	index set of subepochs
K	index set of all disjoint traffic slices
U_k	index set of all update operations in slice k
t_k	number of TCAM entries required by slice k
u_i	index of update operation required by query i
$s_{i,k}$	indicator that query i requires slice k
m_i	memory required in each subepoch by query i
n_i^{acc}	number of subepochs required for accuracy goal for query i (§ 4)
n_i^{lat}	number of subepochs required for latency goal for query i (§ A.2)
T	total available TCAM entries
A	total number of available switch ALUs
M	total available SRAM counters
$d_{i,j}$	indicator that query i executes in subepoch j

C1: $\forall j \in SE,\ \sum_{k \in K} t_k \mathbf{I}\left[\bigvee_{i \in Q} d_{i,j} s_{i,k} = 1\right] \leq T$

C2: $\forall j \in SE,\ k \in K,\ \sum_{u \in U_k} \mathbf{I}\left[\bigvee_{i \in Q} d_{i,j} s_{i,k} \mathbf{I}[u_i = u] = 1\right] \leq A$

C3: $\forall j \in SE,\ \sum_{i \in Q} d_{i,j} m_i \leq M$

C4: $\forall i \in Q,\ \sum_{j \in SE} d_{i,j} \geq 2$

O1: minimize $\sum_{i \in Q} \left| \left(\sum_{j \in SE} d_{i,j} \right) - n_i^{acc} \right|$

O2: minimize $\sum_{i \in Q} \left| \left(\sum_{j \in SE} d_{i,j} \right) - n_i^{lat} \right|$

O3: minimize $\sum_{i \in Q, j \in SE} d_{i,j} m_i$