Traffic monitoring in software dataplane: a generic accuracy-aware adaptive solution

Daphné Tuncer

Department of Computing
Imperial College London (UK)
dtuncer@ic.ac.uk

18/02/2020

Collaborative work with

Gioacchino Tangari (leading efforts – Macquarie University)

Marinos Charalambides (UCL)

George Pavlou (UCL)

Managing networks is hard

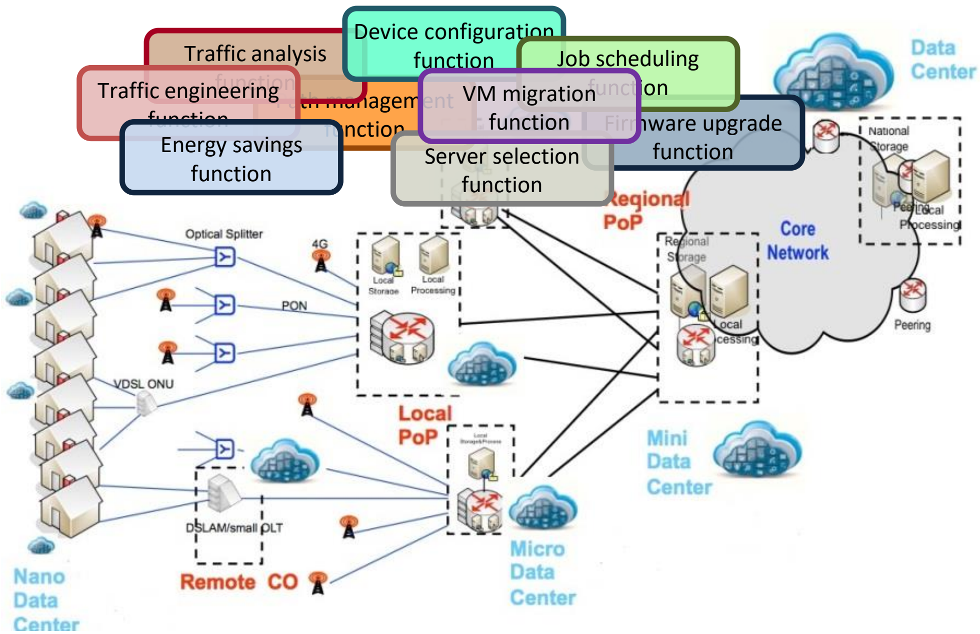
Online Social Media

Rich Multimedia Distribution

Computation and Storage Services

Smart Cities

Online Gaming



Complex systems **challenging** to **manage**

Demand **agility**, **flexibility**, **adaptability** and **reactivity**

Advances in networking have brought promises

Enable reactive behaviour in response to emerging requirements

New opportunities for effective network resource management

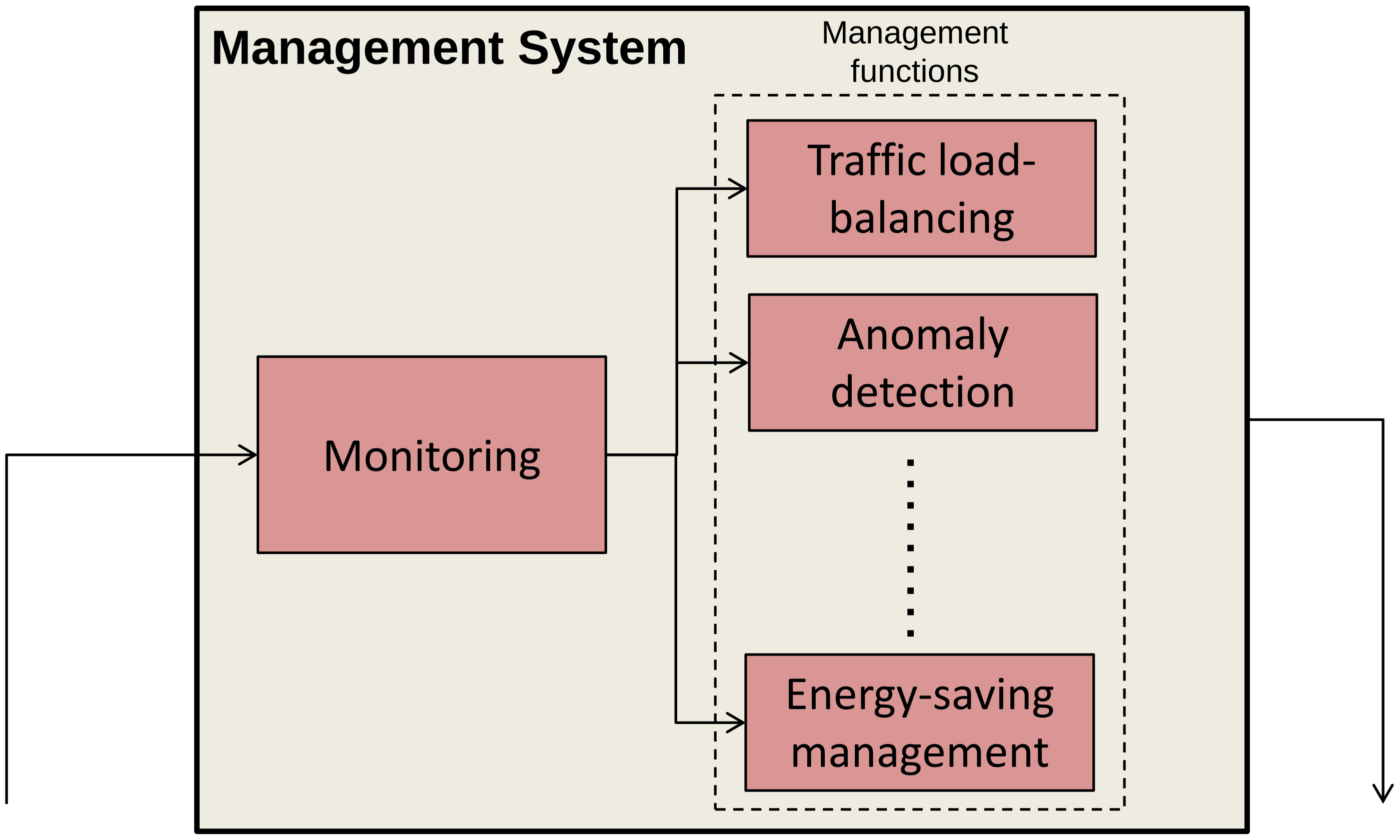
Quickly adapt and react to network and traffic dynamics

Configuration **flexibility**

Express **high-level** operators' **policies**

A key functionality: monitoring

Goal: provide efficiency



Collect statistics from the network resources

Apply new configurations in the network

What is efficient monitoring?

Detailed information

Identify congestion, DDoS attacks, unresponsive servers, …

Timely reports

Detect short-lived episodes, support fast resource reconfigurations, …

Well, easy?...

Reality: hard to produce *detailed* and *timely* reports

- Hardware & resource constraints
- Large scale settings
- Massive and dynamic network traffic

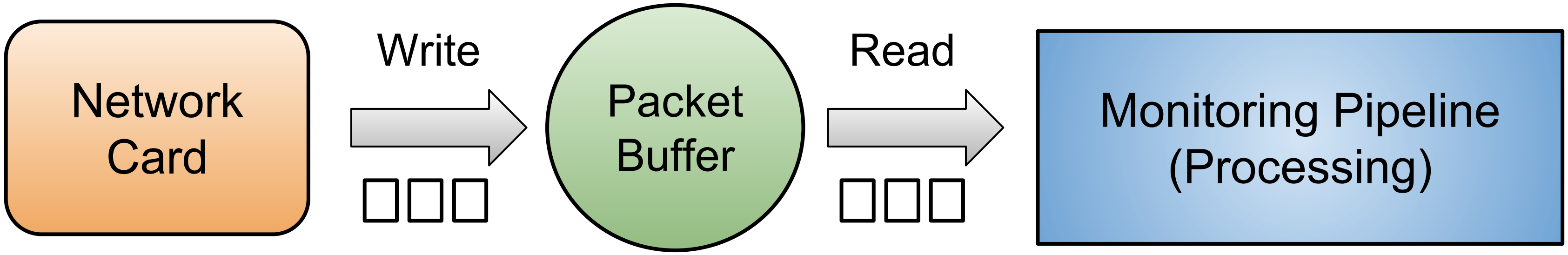
On top of that constraints on the monitoring system

- Scalability requirements
- Good accuracy/resource usage trade-offs

Firestone *et al. "a physical core sells for $0.10-0.11/hr, [...] a maximum potential revenue of around $900/yr"* [FirestoneNSDI18]

Designing efficient monitoring for software-based networks

Traffic monitoring in software dataplane



<u>Monitoring States</u>

State1 : Count

State 2: Count + Heavy Hitter

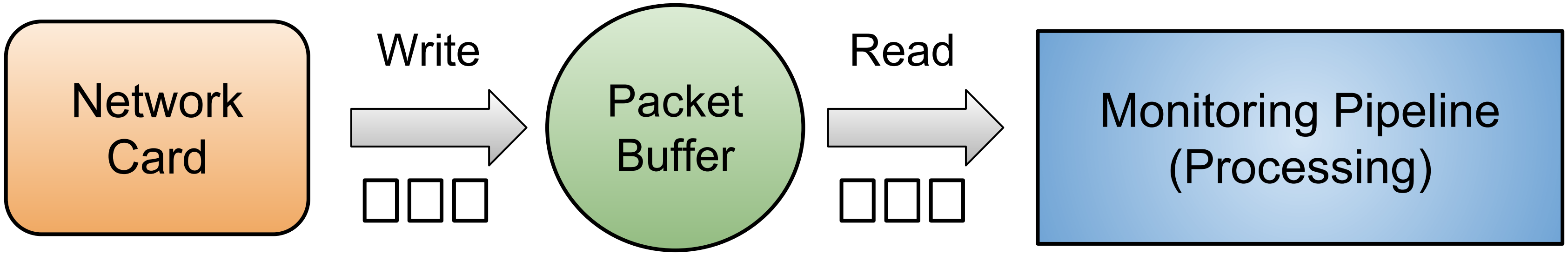
Measurement Task: involves a set of operations (*e.g.,* sum bytes)

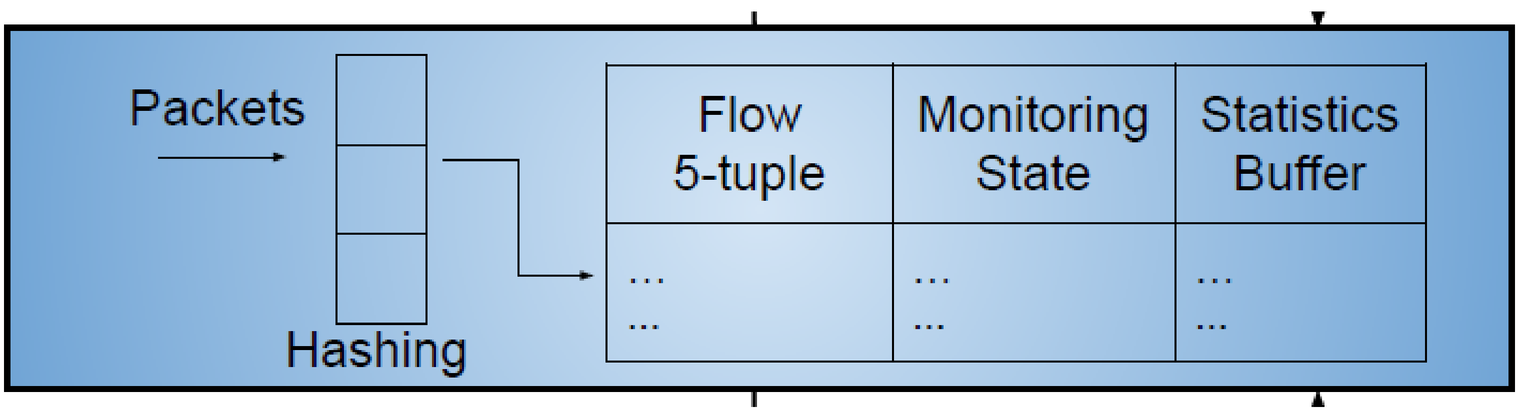
State 3: Count + Heavy Hitter + Retransmission

State 4: Count + Heavy Hitter + Retransmission + Bursty Flow

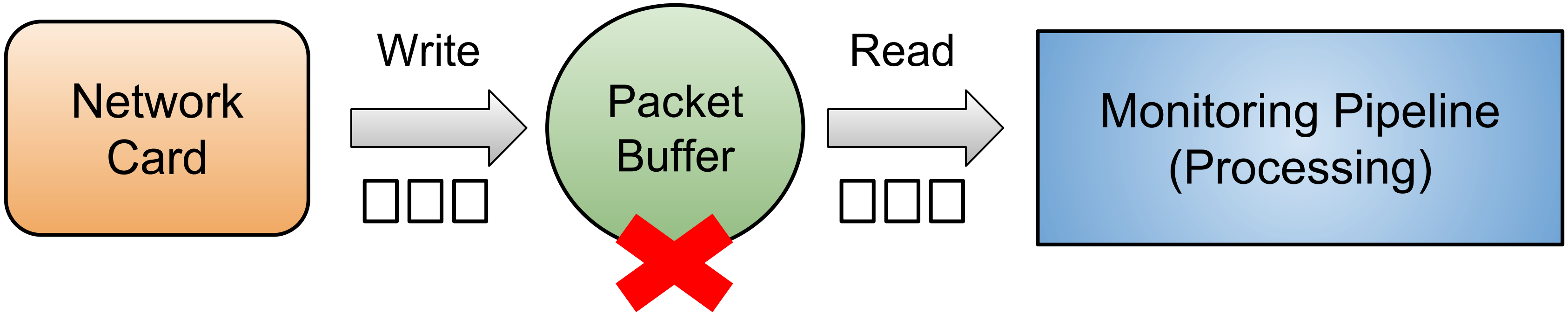
State 5: Count + Heavy Hitter + Retransmission + Bursty Flow + Latency Change

Traffic monitoring in software dataplane





Traffic monitoring in software dataplane



Limited total time budget available

=> Potential packet loss at the buffers in case of bottlenecks

Monitoring States

S1 26 ns for processing
S2 87 ns for processing
S3 96 ns for processing
S4 122 ns for processing
S5 163 ns for processing

Traffic monitoring in software dataplane

How to reconfigure monitoring operations at run-time to cope with emerging conditions?

Monitoring States

S1 26 ns for processing
S2 87 ns for processing
S3 96 ns for processing
S4 122 ns for processing
S5 163 ns for processing

Solution: adapting monitoring at run-time

Detect changes in operating conditions in a timely manner

Dynamically reconfigure (per flow) measurement operations

<u>Monitoring States</u>

S1
S2
S3
S4
S5

Adjust packet processing time

<u>Monitoring States</u>

S1
S2
S3
S4
S5

No need of overprovisioning

Yes, but how do you…

1. Preserve monitoring report accuracy?

2. Avoid packet starvation (*i.e.,* all packets processed *in time*)?

3. Guarantee low computation overhead (no more than 1% CPU-time)?

Key idea: estimating per-packet processing time

Total expected time of a packet in the monitoring pipeline

Packet rate

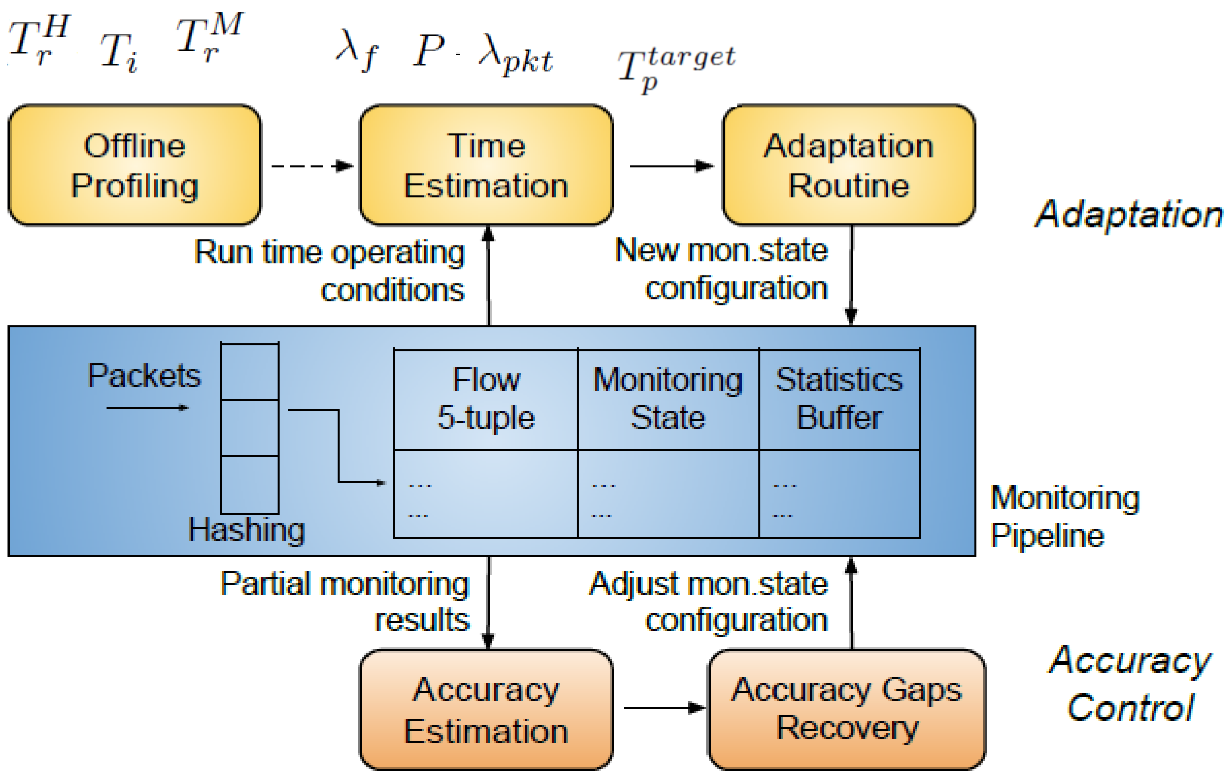
$$(1-\lambda_f)[T_r^H P + T_r^M (1-P) + T_p^{target}] + \lambda_f T_i = 1/\lambda_{pkt}$$

Retrieval of existing flow entry information based on whether this is in cache (**H**it) or memory (**M**iss)

Insertion of a new flow entry

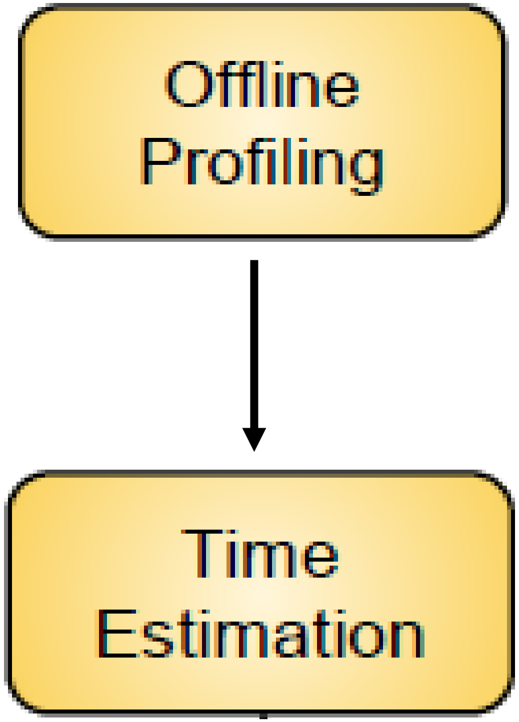
Targeted per packet time

MONA



T_r^H T_i T_r^M
λ_f $P \cdot \lambda_{pkt}$
T_p^{target}
Offline Profiling
Time Estimation
Adaptation Routine
Adaptation
Run time operating conditions
New mon.state configuration
Packets
Hashing
Flow 5-tuple
Monitoring State
Statistics Buffer
...
Monitoring Pipeline
Partial monitoring results
Adjust mon.state configuration
Accuracy Estimation
Accuracy Gaps Recovery
Accuracy Control

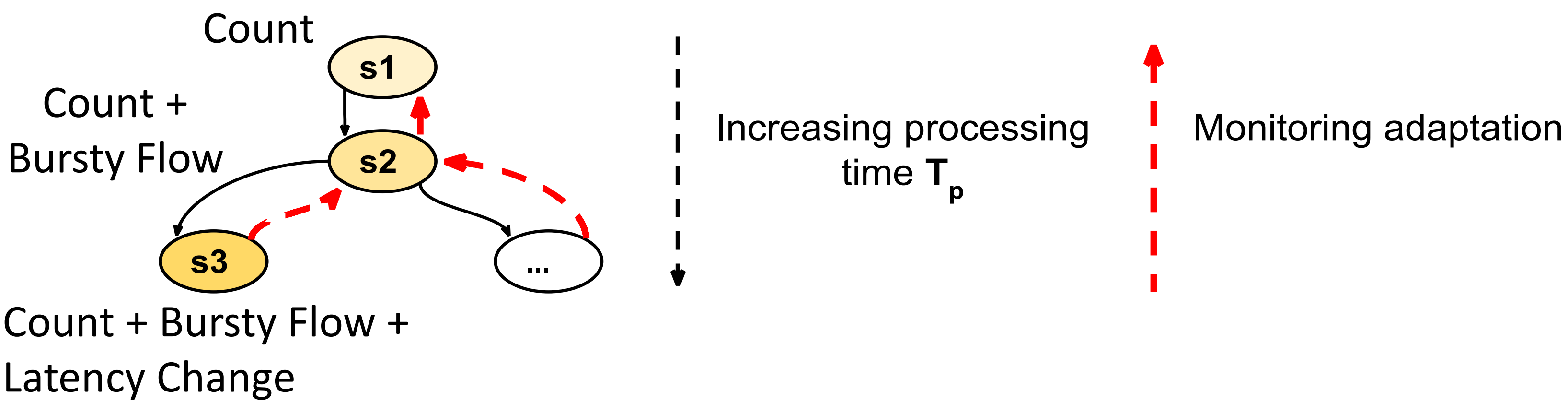
Time profiling and estimation



Benchmarking of retrieval and insertion times

Based on sampling

Adaptation routine



Two strategies

Greedy: adaptation applied to random sets of flow-entries

Low-States-First (LSF): downgrade in priority flows mapped to less advanced monitoring states (except s1)

Monitoring accuracy control

Objective:

To re-adjust flow allocations in order to satisfy a global accuracy threshold for all tasks

Two main steps:

1. To quantify the effect of adaptations on report accuracy
2. To recover accuracy gaps by re-adjusting flow allocation

Task generic online accuracy estimation

Accuracy estimated through the recall

$$Recall_{iw} = N_{iw}^{Found} / (N_{iw}^{Found} + N_{iw}^{Miss})$$

unknown

Let's expand N^Miss

$$N_{iw}^{Miss} = F_{iw}^{Miss} \cdot E[X_{iw}^{Miss}]$$

flows dropped for task i from adaptation

events for a missing flow

Objective: find an estimator for X^Miss

Task generic online accuracy estimation (con't)

Our solution: risk minimization strategy

$$R(\hat{x}^{Miss}) = \sum_{l=0}^{\infty} L(x_l^{Miss}, \hat{x}^{Miss}) Prob(X_{iw}^{Miss} = x_l^{Miss})$$

Best estimator for X^Miss = one minimizing risk function R

$$\hat{x}_{Best}^{Miss} = \underset{\hat{x}^{Miss}}{\operatorname{argmin}} R(\hat{x}^{Miss}, L)$$

Recovering accuracy gaps

Solution: the Richs give to the Poors

Re-allocation of flows from monitoring states with rich tasks to monitoring states with poor tasks

Algorithm 2: Recover Accuracy Gaps

function UPDATESTEPSIZE(x, S_x)
    Compute accuracy decrease $D = A_{x,w-1} - A_{x,w}$
    Update residual accuracy $H = A_{x,w} - threshold$
    if $D > H$ **then return** INCREASE(S_x)
    else return DECREASE(S_x)
function REBALANCEBYSTEP(s_{Rich}, s_{Poor}, S)
    Compute $\Delta^- = S/n_p$, where n_p number of poor states
    Retrieve $E[t_s^{Poor}]$ from $T_{Poor,j}$, $j \in 1, .., n$
    Compute Δ^+ from equilibrium condition (9)
    return Δ^-, Δ^+
procedure RECOVERYGAPS(A_w, M, T_w)
    Find set of rich, poor states $\{s_{Rich}\}$, $\{s_{Poor}\}$ using A_w
    if $\{s_{Poor}\} == \emptyset$ or $\{s_{Rich}\} == \emptyset$ **then return**
    for each x in $\{s_{Rich}\}$ **do**:
        S_x = UPDATESTEPSIZE(x, S_x)
    for each (x, y) with $x \in \{s_{Rich}\}, y \in \{s_{Poor}\}$ **do**:
        REBALANCEBYSTEP(x, y, S)

MONA implementation

- Implemented in C

  Generic monitoring pipeline based on a single flow-table

- Flow-table realized as a hash-table

  Table size = 2^20 entries to limit hash collisions

- Flow-entry size = 64 bytes (fit within a single cache)

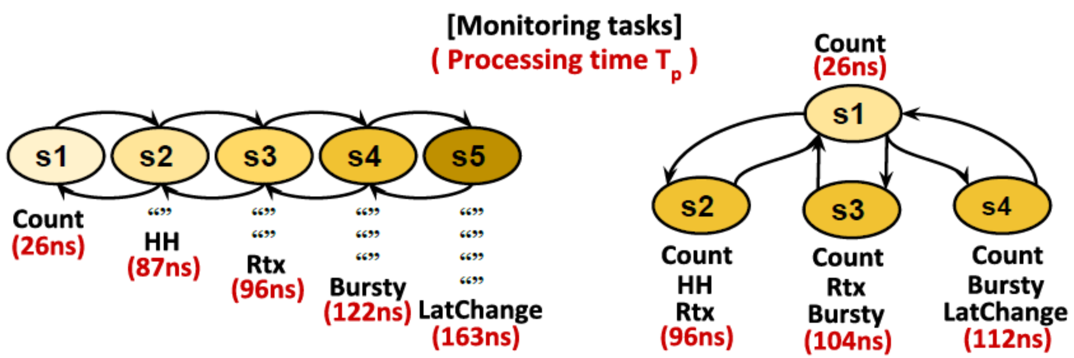
- Packet trace generated based on reported flow statistics in Facebook data centers [RoySIGCOMM15]

MONA evaluation setup

Evaluation with four measurement tasks

- Heavy Hitter detection (HH)
- Bursty flow detection (Bursty)
- Latency Change detection (LatChange)
- ReTransmission detection (RTx)

Two monitoring state configurations



How do we validate MONA?

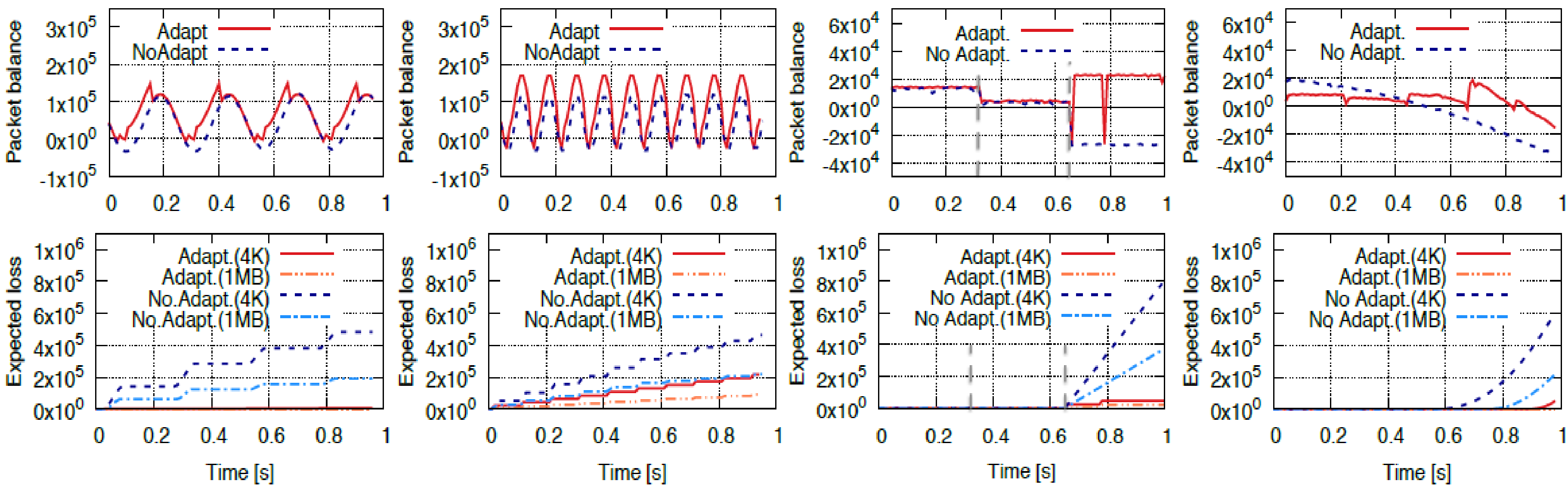
1. How does adaptive traffic monitoring perform in terms of packet loss risk and adaptation responsiveness ?

2. What is the impact of monitoring adaptation and accuracy control on the measurement tasks?

3. What are the throughput limiting factors for MONA?

4. What is the overhead of our solution?

MONA robustness to changing conditions



(a) Rate variation ($T = 250ms$)

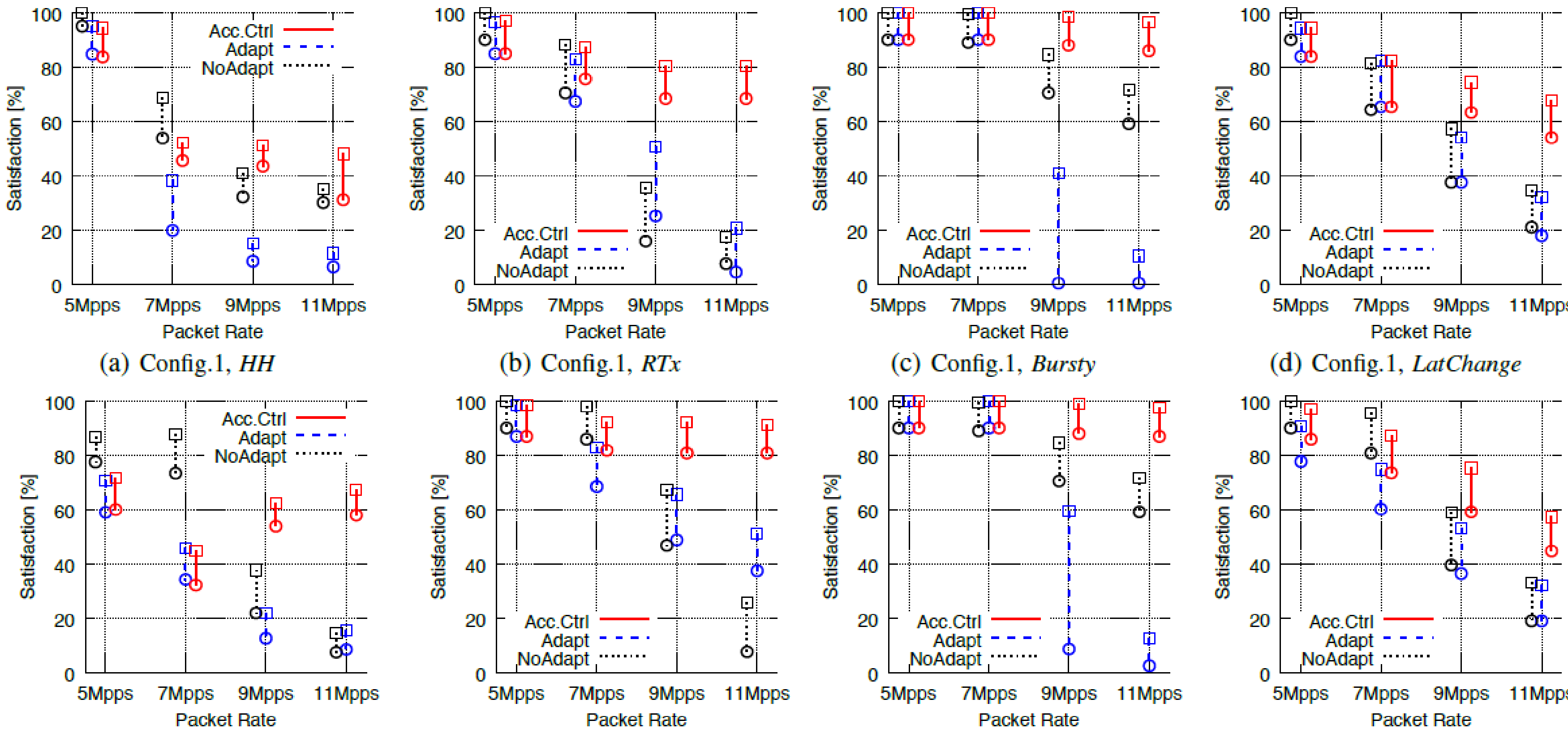
(b) Rate variation ($T = 100ms$)

(c) Shared resource contention

(d) Traffic skew variation

MONA prevents packet loss and preserves packet balance

MONA measurement tasks accuracy



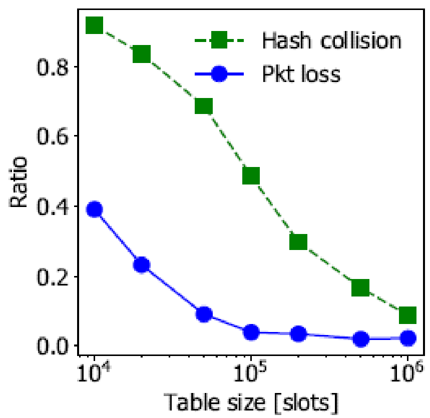
(a) Config.1, *HH* (b) Config.1, *RTx* (c) Config.1, *Bursty* (d) Config.1, *LatChange*

(e) Config.2, *HH* (f) Config.2, *RTx* (g) Config.2, *Bursty* (h) Config.2, *LatChange*

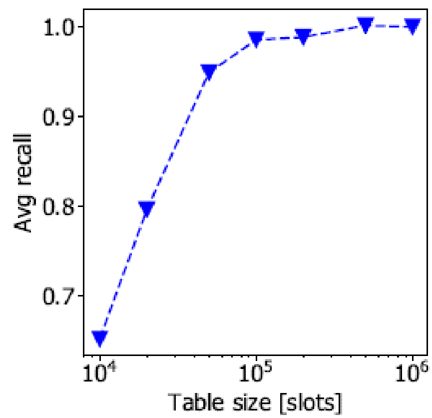
MONA maintains monitoring report accuracy

MONA throughput limiting factors

Impact of hash collisions

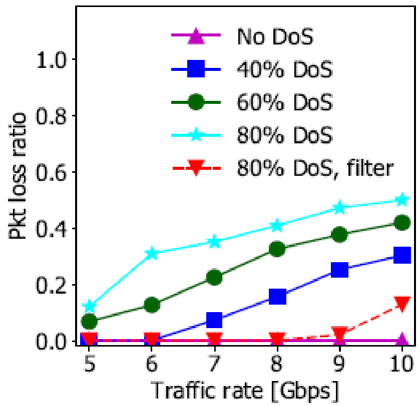


(a) Hash collision and pkt loss ratio

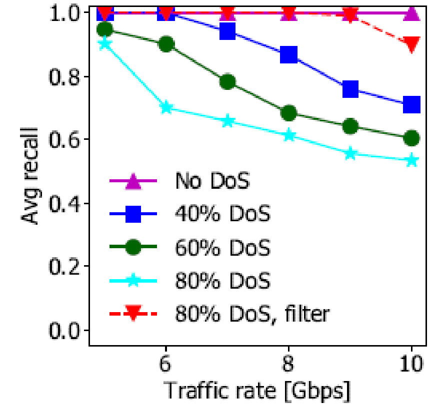


(b) Average task accuracy (recall)

Impact of uniform traffic (DoS attack)

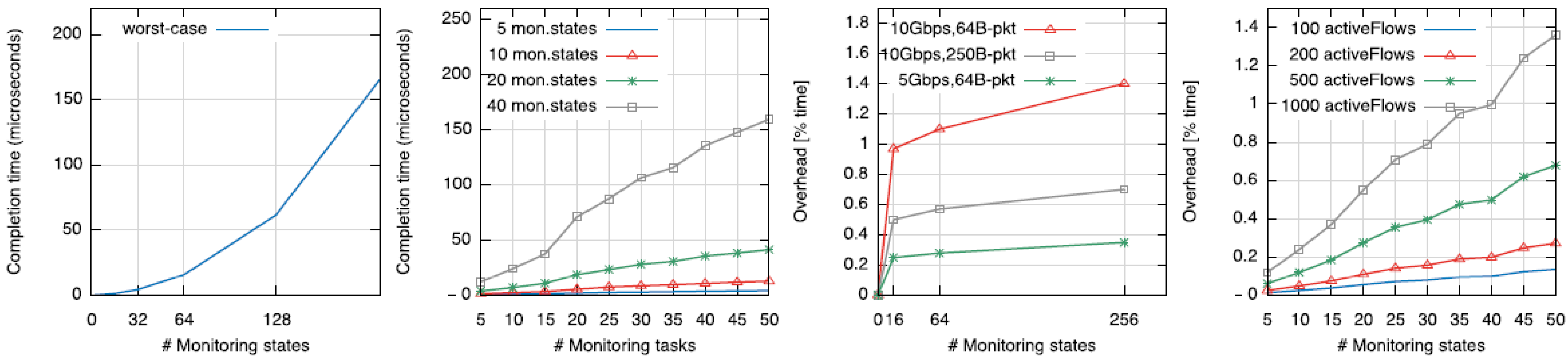


(a) DoS-induced packet loss



(b) Accuracy (recall) reductions

MONA overhead



(a) Adaptation execution time (b) Accuracy control execution time (c) Estimation overhead (d) Reconfiguration overhead

MONA enables **short timescale** re-configurations (every 10ms), with **no additional** processor **core**(s) and **minimal CPU-time** overhead (~1-2%),

Concluding remarks

1. Timely and accurate resource monitoring fundamental for any network management system

2. Self-adaptive monitoring approaches as drivers to responsiveness and flexibility

3. Our solution: MONA

   Adaptive monitoring framework offering resilience to bottlenecks
   + preservation of monitoring accuracy

Used references

[FirestoneNSDI18] D. Firestone, et al., "Azure accelerated networking: SmartNICs in the public cloud," *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI),* 2018.

[RoySIGCOMM15] A. Roy, et al., "Inside the social network's (datacenter) network," *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM),* 2015.

Our papers

[TON20] G. Tangari, M. Charalambides, D. Tuncer, G. Pavlou, "Accuracy-Aware Adaptive Traffic Monitoring for Software Dataplanes, " to appear in IEEE Transactions on Networking (ToN), 2020.

[TNSM18] G. Tangari, D. Tuncer, M. Charalambides, Y. Qi and G. Pavlou, "Self-Adaptive Decentralized Monitoring in Software-Defined Networks," in IEEE Transactions on Network and Service Management (TNSM), 2018.

[CNSM17] G. Tangari, M. Charalambides, D. Tuncer, G. Pavlou, "Adaptive Traffic Monitoring for Software Dataplanes," in the Proc. of the 13th IFIP/IEEE International Conference on Network and Service Management (CNSM'17), Tokyo, Japan, November 2017.

[IM17] G. Tangari, D. Tuncer, M. Charalambides and G. Pavlou, "Decentralized Monitoring for Large-Scale Software-Defined Networks," to appear in the Proc. of the IFIP/IEEE Integrated Management Symposium (IM'17), Lisbon, Portugal, May 2017.