A change detection primitive for the network data plane

Gonçalo Matos, Salvatore Signorello, Fernando M. V. Ramos

TÉCNICO LISBOA

inesc id lisboa

Ciências ULisboa

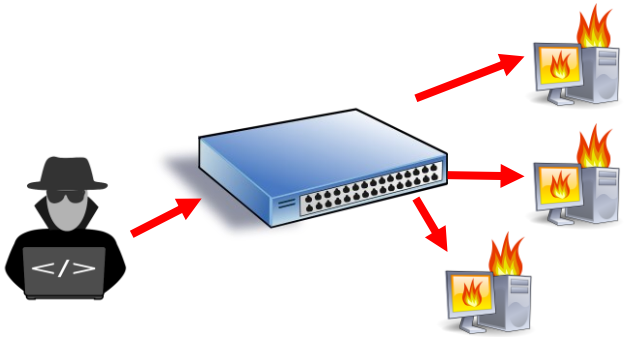
Problem

Traffic changes might indicate an anomaly in the network.

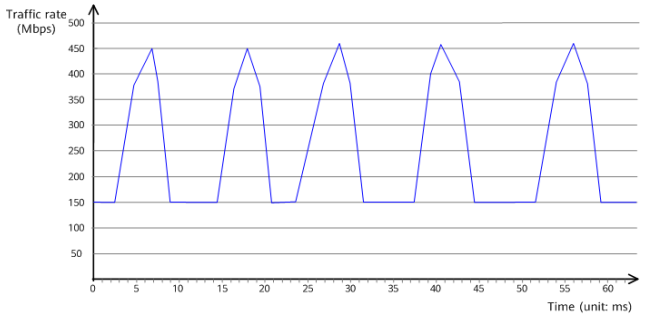
Problem

Traffic changes might indicate an **anomaly** in the network.

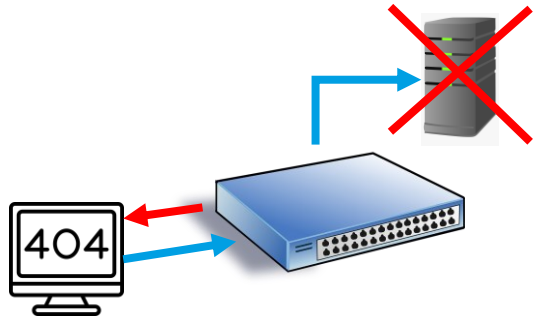
Their **timely detection** is of paramount importance to network operation!



Network Attack



Microburst



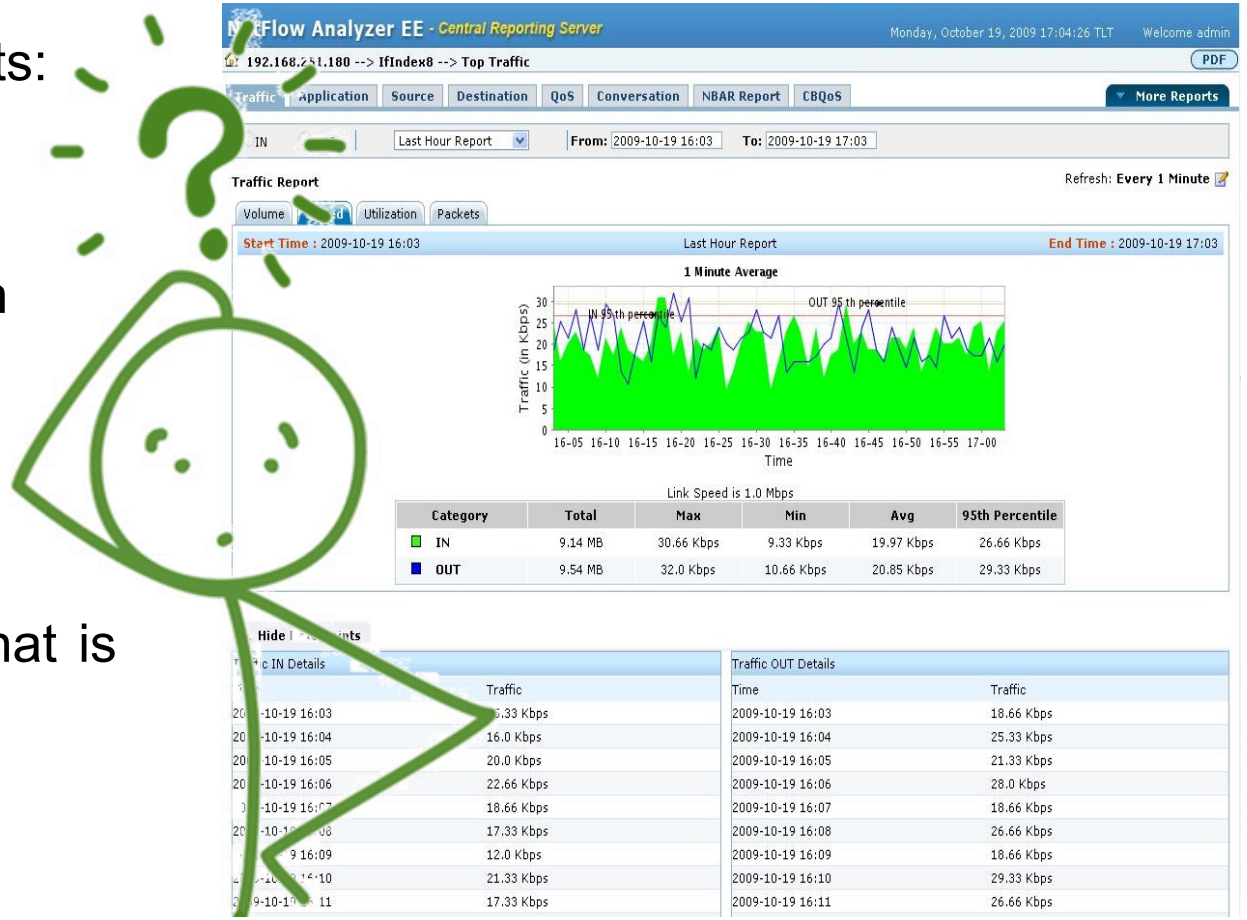
Faulty Hardware

Traditional change detection

Sampling via Flow-based measurements:

- Standard, e.g., Netflow and sFlow
- Sampling → CPU, memory, bandwidth
- Low sampling rate → **Low Fidelity**

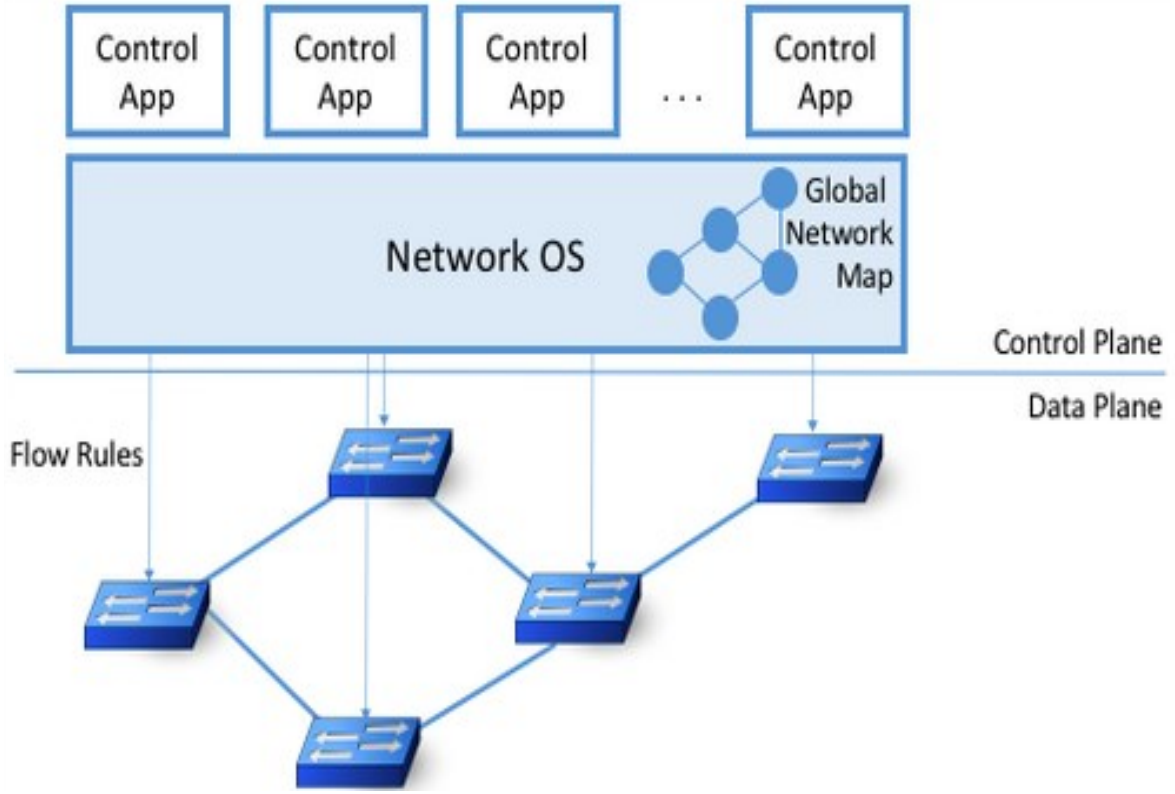
Based on **fixed-function equipment** that is **difficult to manage** and **to configure**.



Networks are changing!

Software Defined Networking (SDN):

- Separation of the network planes.
- Programmability of each plane.



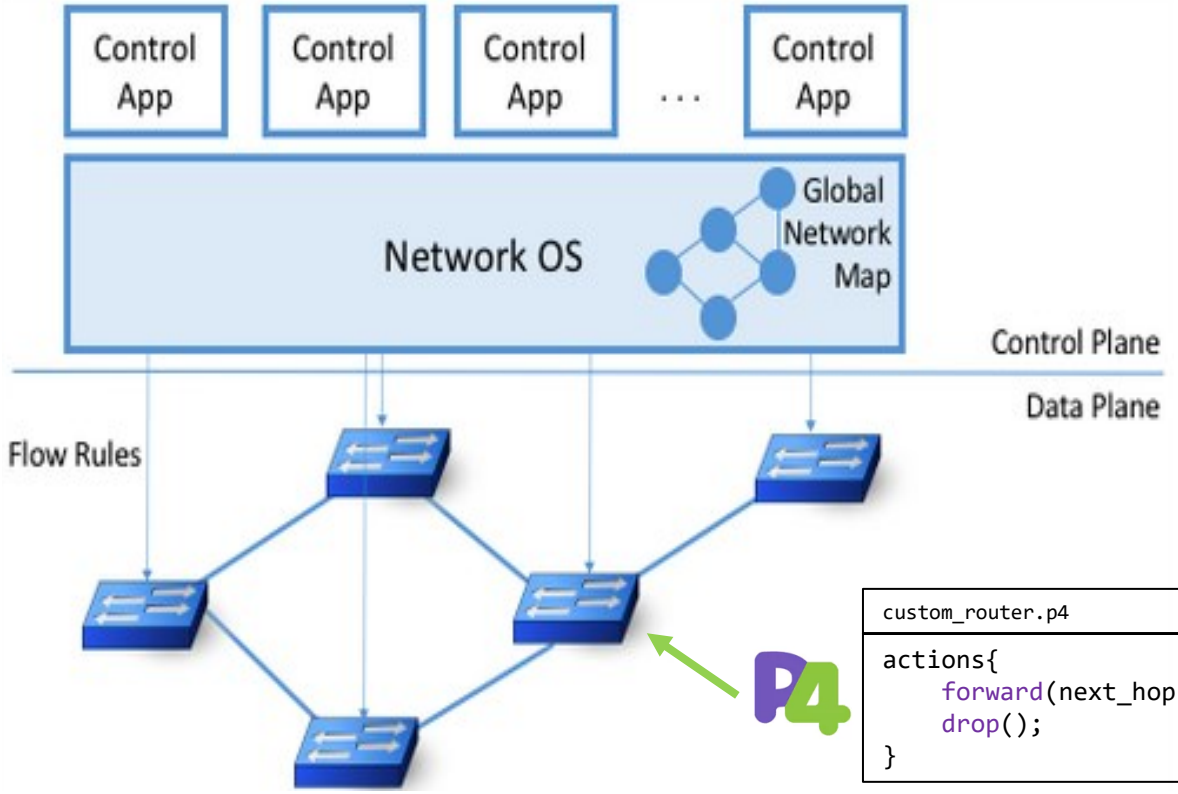
Networks are changing!

Software Defined Networking (SDN):

- Separation of the network planes.
- Programmability of each plane.

Programmable Switching Chips:

- Detection of traffic changes **inside the data plane**.



In-network change detection

Some solutions already enable change detection using P4 to program the data plane.

However, these are often **heavy-hitter** based.

But sometimes the problem is with smaller flows!

Approach	Tbps performance P4	Heavy-Hitter Change Detection	Generic Change Detection
Hashpipe[1]	X	X	
Elastic Sketch[2]	X	X	
SketchLearn[3]	X	X	
K-ary Sketch[4]		X	X

In-network change detection

Some solutions already enable change detection using P4 to program the data plane.

However, these are often **heavy-hitter** based.

But sometimes the problem is with smaller flows!

Approach	Tbps performance P4	Heavy-Hitter Change Detection	Generic Change Detection
Hashpipe[1]	X	X	
Elastic Sketch[2]	X	X	
SketchLearn[3]	X	X	
K-ary Sketch[4]		X	X

In-network change detection

Some solutions already enable change detection using P4 to program the data plane.

However, these are often **heavy-hitter** based.

But sometimes the problem is with smaller flows!

Approach	Tbps performance	Heavy-Hitter Change Detection	Generic Change Detection
Hashpipe[1]	X	X	
Elastic Sketch[2]	X	X	
SketchLearn[3]	X	X	
K-ary Sketch[4]		X	X

In-network change detection

Some solutions already enable change detection using P4 to program the data plane.

However, these are often **heavy-hitter** based.

But sometimes the problem is with smaller flows!

Approach	Tbps performance	Heavy-Hitter Change Detection	Generic Change Detection
Hashpipe[1]	X	X	
Elastic Sketch[2]	X	X	
SketchLearn[3]	X	X	
K-ary Sketch[4]		X	X

Contributions

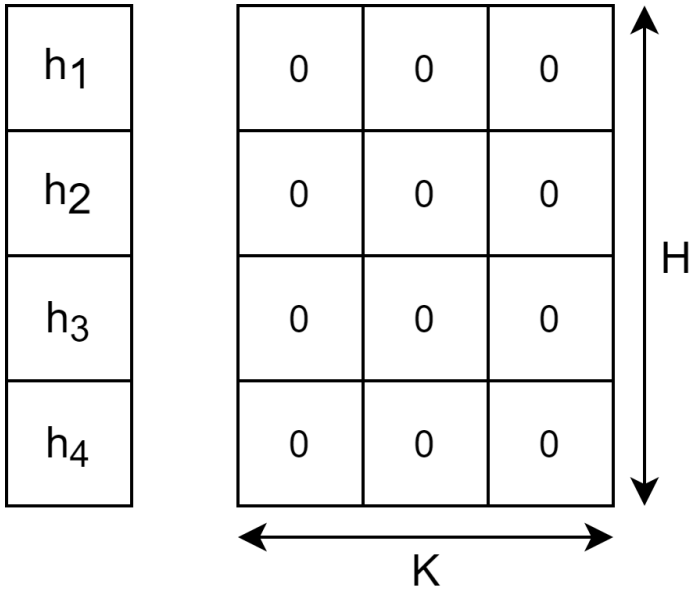
- The design of **K-meleon**, an *online change detection system* that leverages programmable switches.
- The implementation of a **prototype in P4** for Tofino switch.
- An evaluation using the software switch bmv2 that demonstrates K-meleon achieves the same level of **accuracy** as the **k-ary algorithm**.

Design

K-meleon

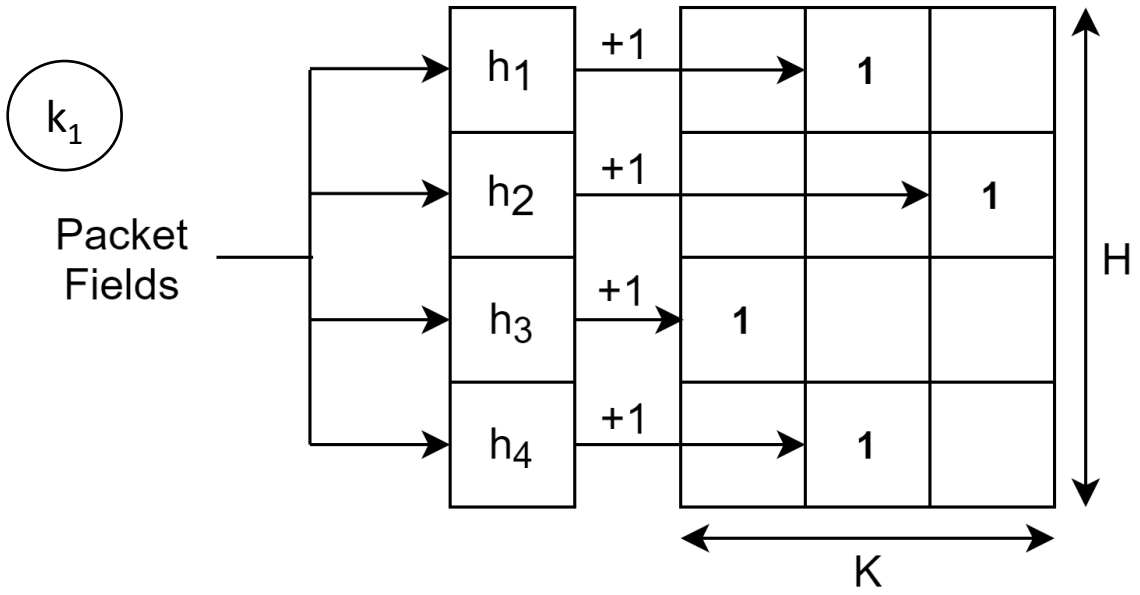
Sketches

Sketches are **compact summaries of network traffic**, which enable the design of memory-efficient network monitoring systems.



Sketches

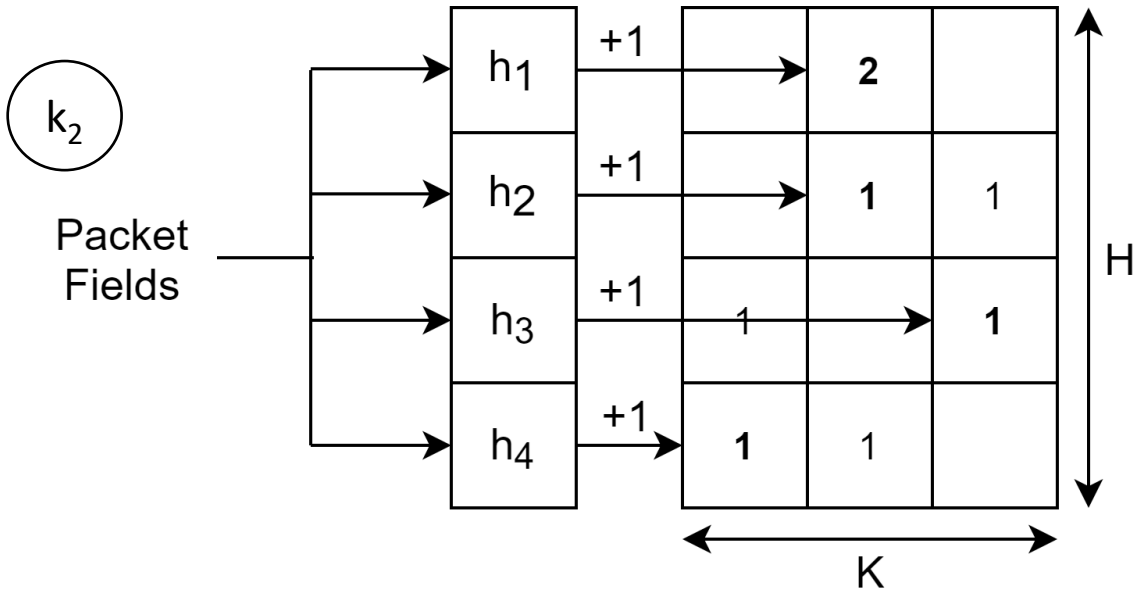
Sketches are **compact summaries of network traffic**, which enable the design of memory-efficient network monitoring systems.



UPDATE

Sketches

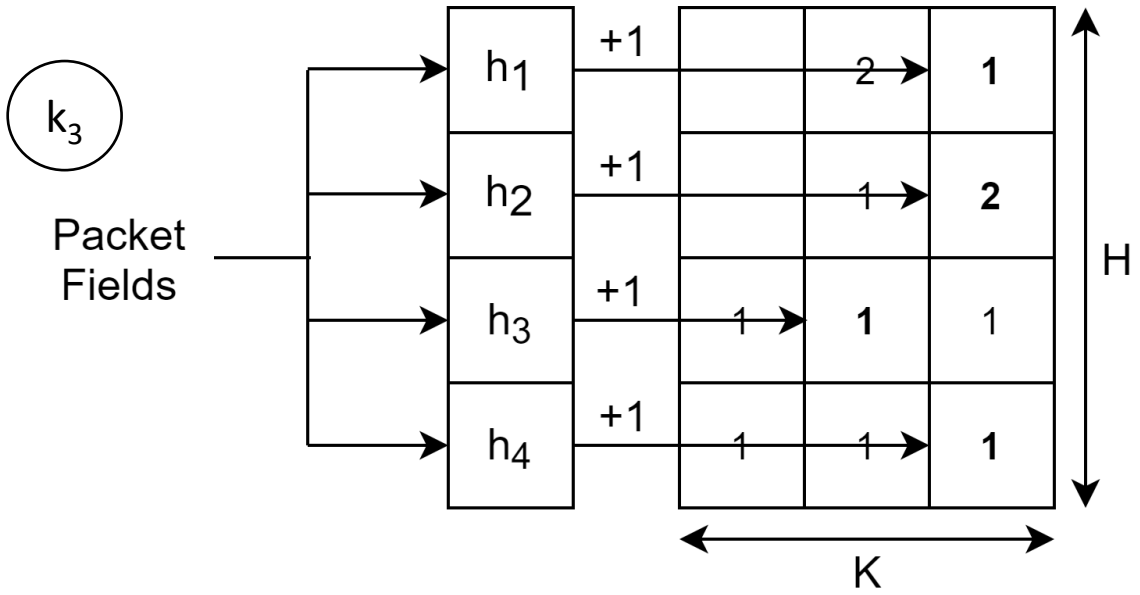
Sketches are **compact summaries of network traffic**, which enable the design of memory-efficient network monitoring systems.



UPDATE

Sketches

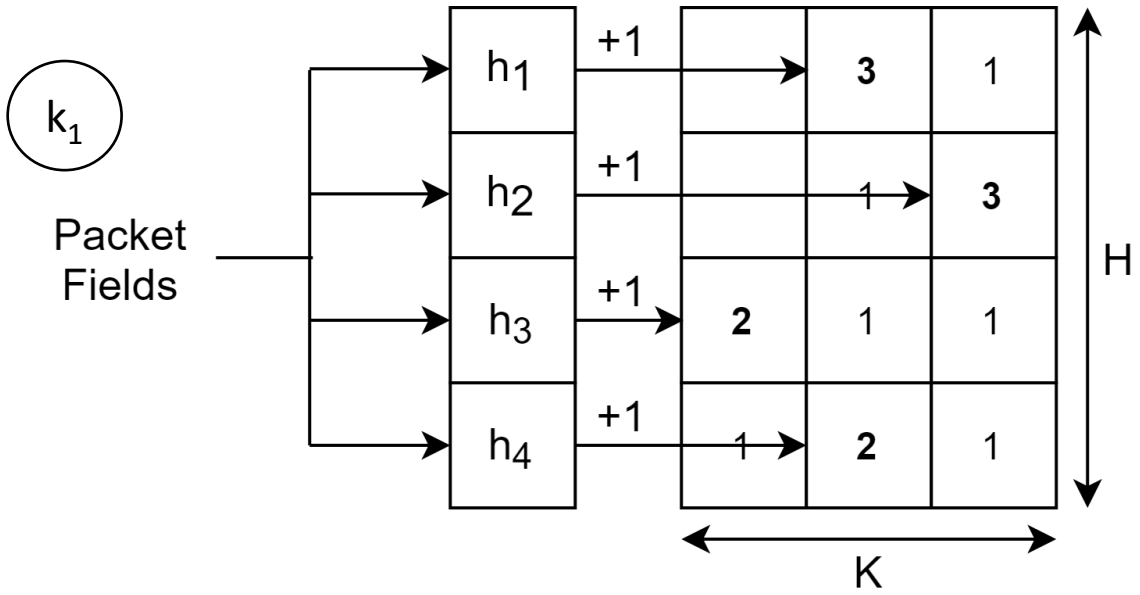
Sketches are **compact summaries of network traffic**, which enable the design of memory-efficient network monitoring systems.



UPDATE

Sketches

Sketches are **compact summaries of network traffic**, which enable the design of memory-efficient network monitoring systems.



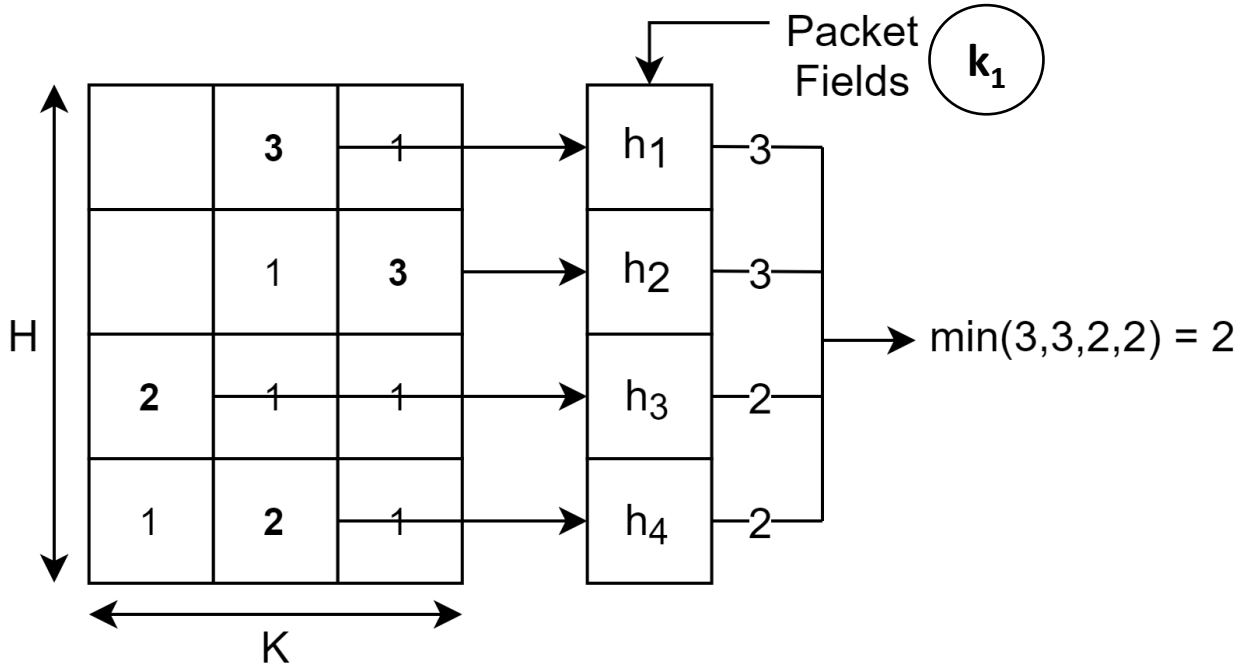
UPDATE

Sketches

Sketches are **compact summaries of network traffic**, which enable the design of memory-efficient network monitoring systems.

ESTIMATE

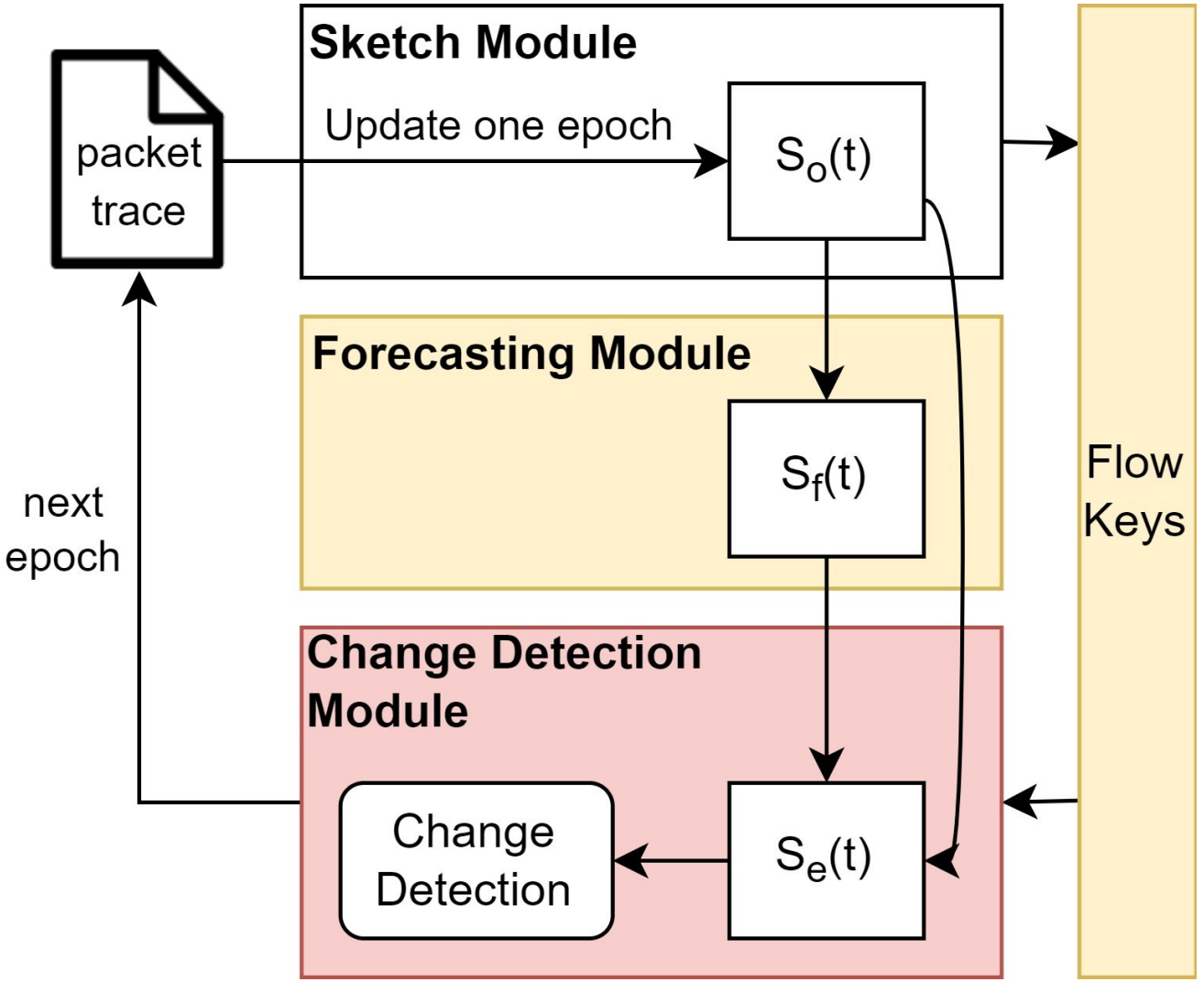
with Count-min



K-ary Algorithm

Batch-based approach.

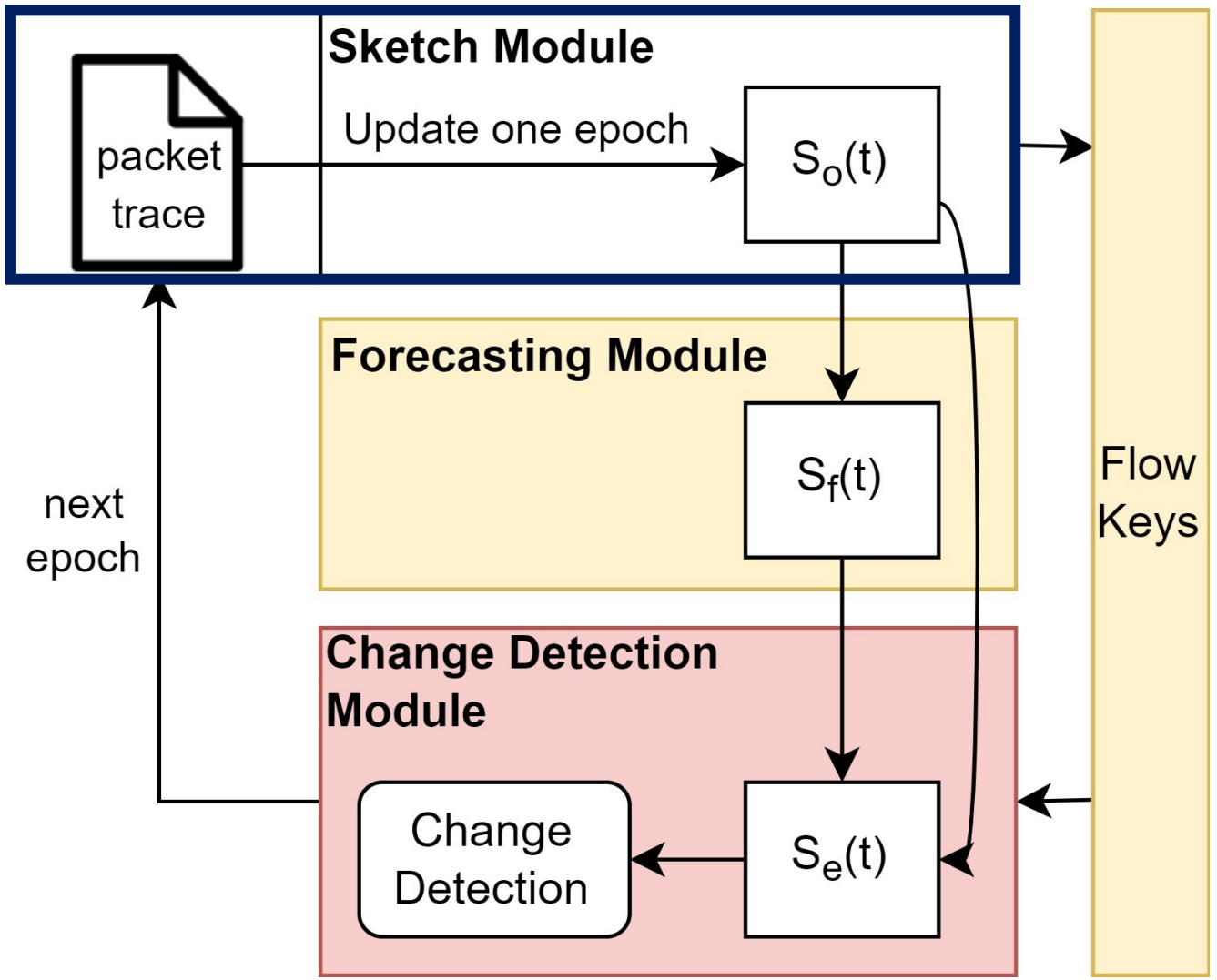
- Splits time into a series of time intervals, called **epochs**.
- Performs all operations at the end of each epoch.



K-ary Algorithm

Batch-based approach.

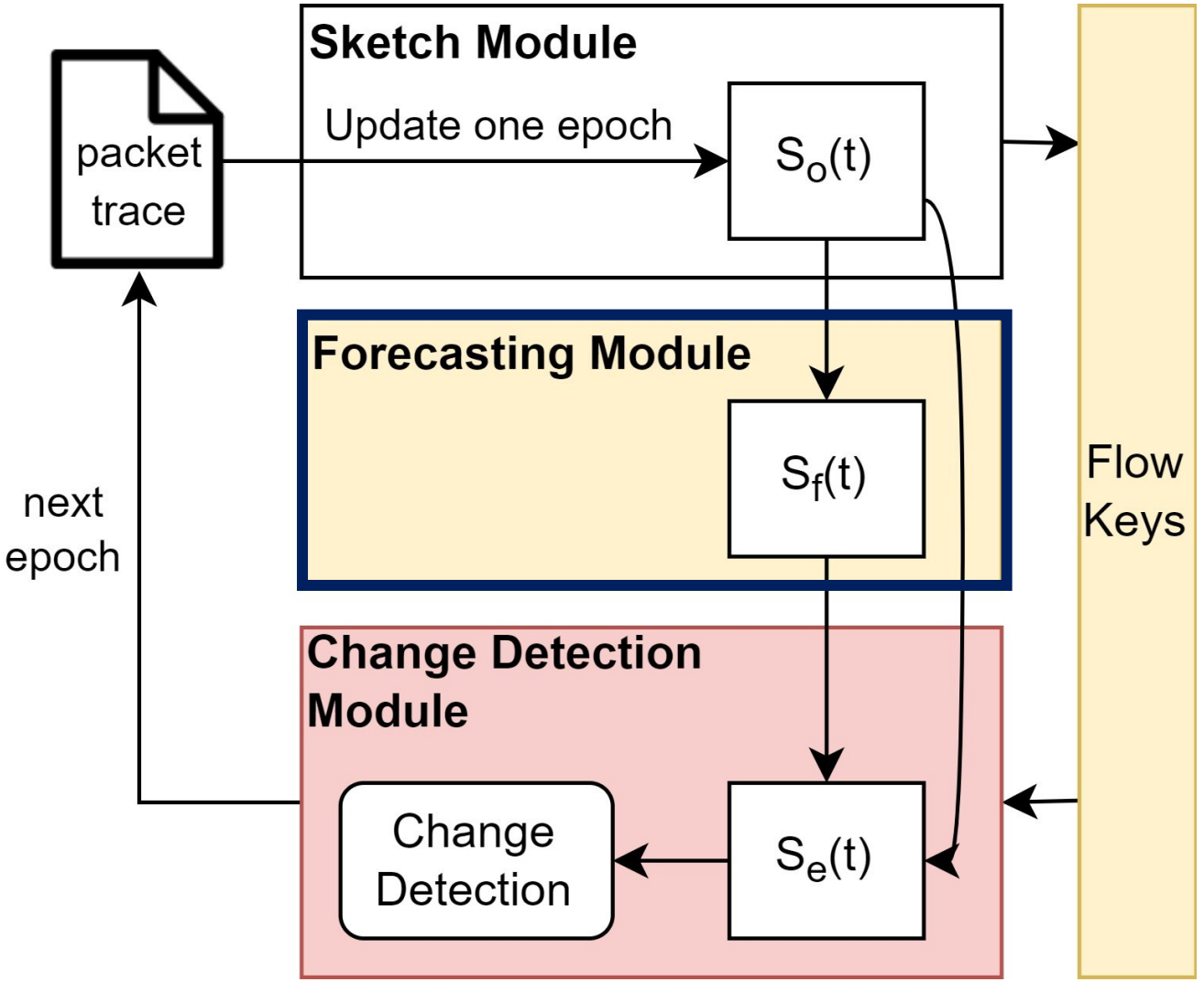
- Splits time into a series of time intervals, called **epochs**.
- Performs all operations at the end of each epoch.



K-ary Algorithm

Batch-based approach.

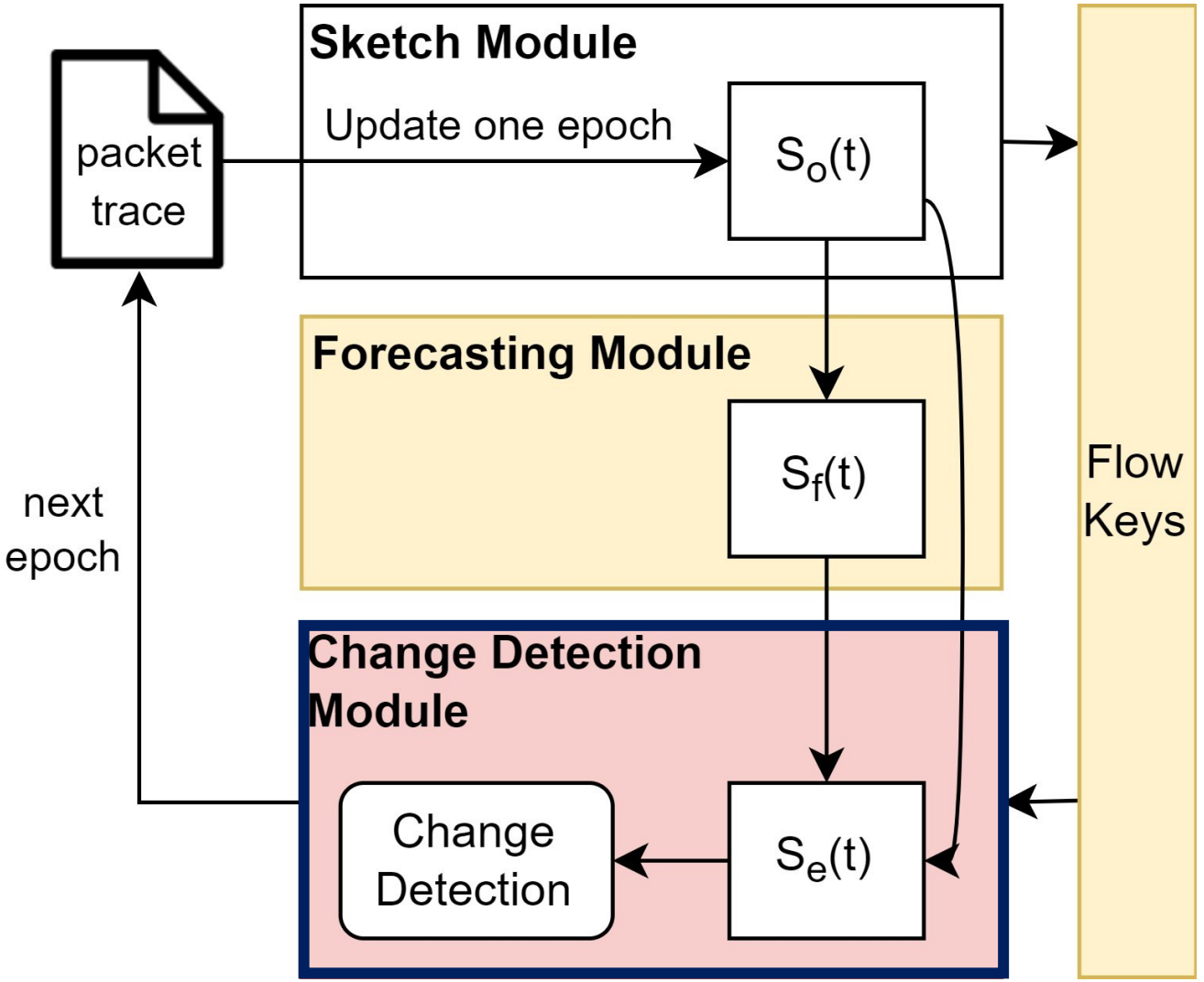
- Splits time into a series of time intervals, called **epochs**.
- Performs all operations at the end of each epoch.



K-ary Algorithm

Batch-based approach.

- Splits time into a series of time intervals, called **epochs**.
- Performs all operations at the end of each epoch.



K-ary Operations

The forecasting module uses **forecasting models**, such as the **EWMA**:

$$S_f(t) = \begin{cases} \alpha \cdot S_o(t-1) + (1-\alpha) \cdot S_f(t-1), & t > 2 \\ S_o(1), & t = 2 \end{cases}$$

Floating-point multiplications

The change detection module computes **estimates** on top of the error sketch:

$$v_a^{\text{est}} = \text{median}_{i \in [H]} \{ v_a^{h_i} \}$$

medians

$$F_2^{\text{est}} = \text{median}_{i \in [H]} \{ F_2^{h_i} \}$$

where

$$v_a^{h_i} = \frac{T[i][h_i(a)] - sum(S)/K}{1 - 1/K}$$

Sum of all values in the sketch

where

$$F_2^{h_i} = \frac{K}{K-1} \sum_{j \in [K]} (T_S[i][j])^2 - \frac{1}{K-1} (sum(S))^2$$

squares

Programming the data plane

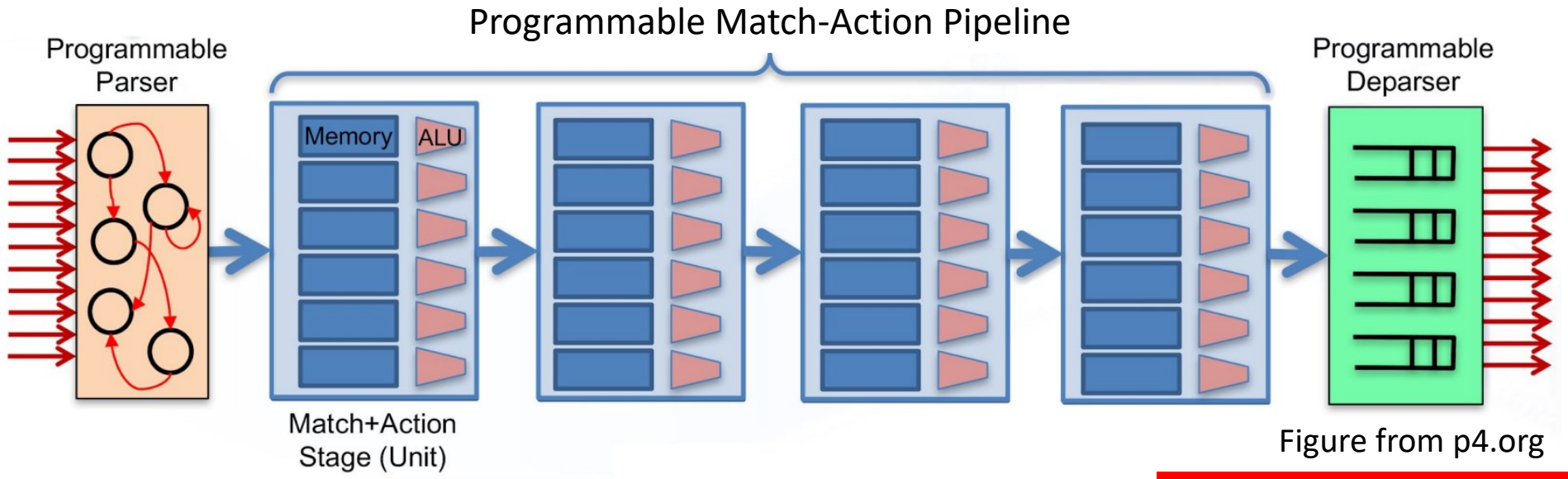


Figure from p4.org

- Protocol-Independent Switch Architecture (PISA)

Memory and budget of operations are limited!

K-ary algorithm - limitations

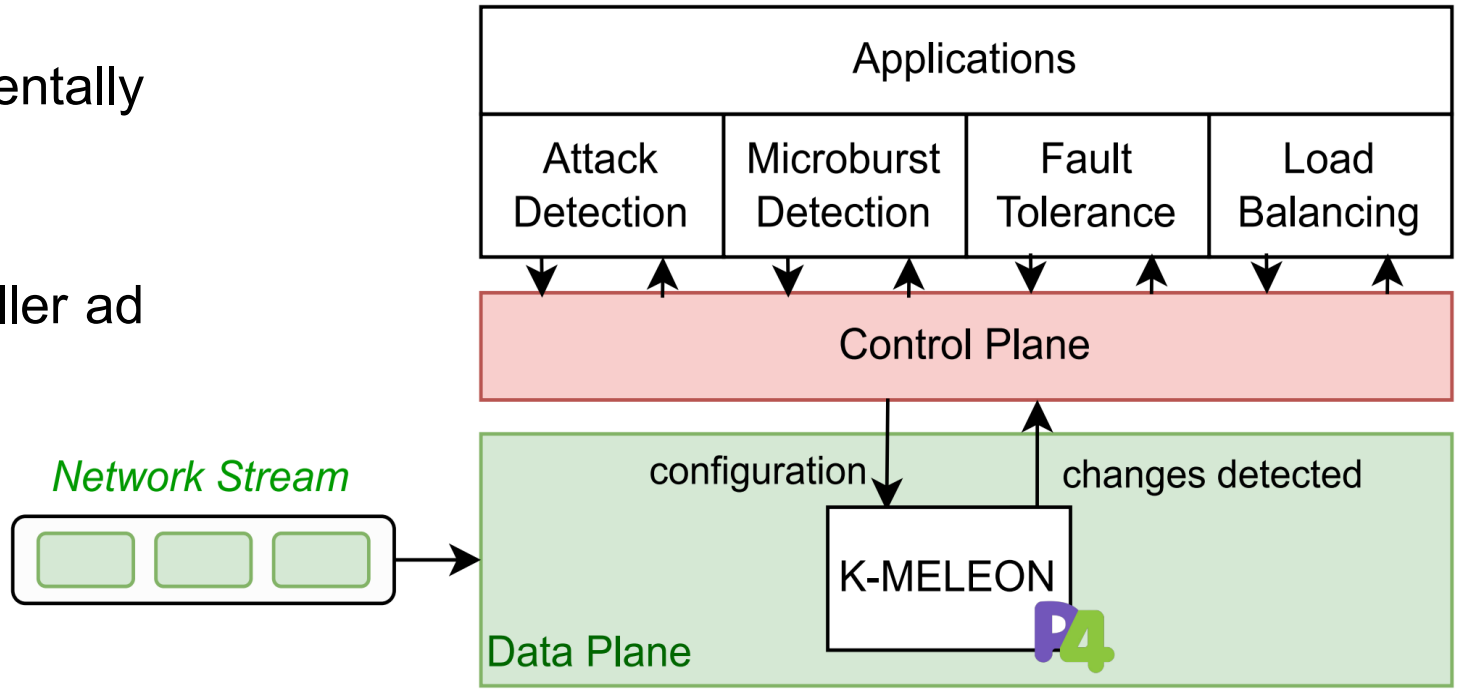
Performs batch-based operations at the end of each epoch, thus;

- Does not fit the **constraints of the data plane** programming model;
- Performs **complex operations** not supported in P4;

The k-meleon

Stream-based approach.

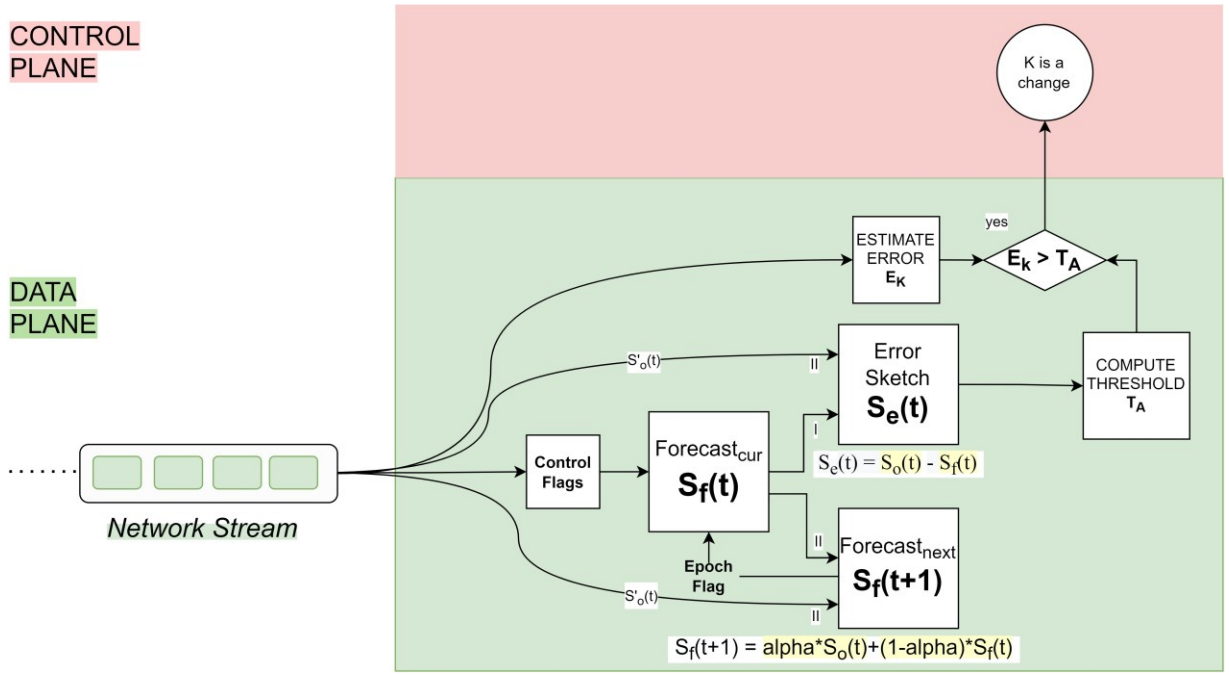
- Performs operations incrementally with each packet.
- Sends changes to the controller ad hoc.



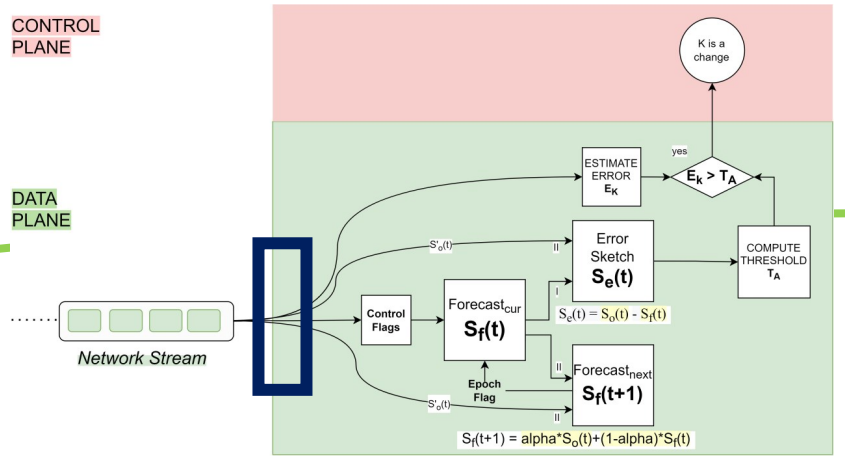
The k-meleon

Stream-based approach.

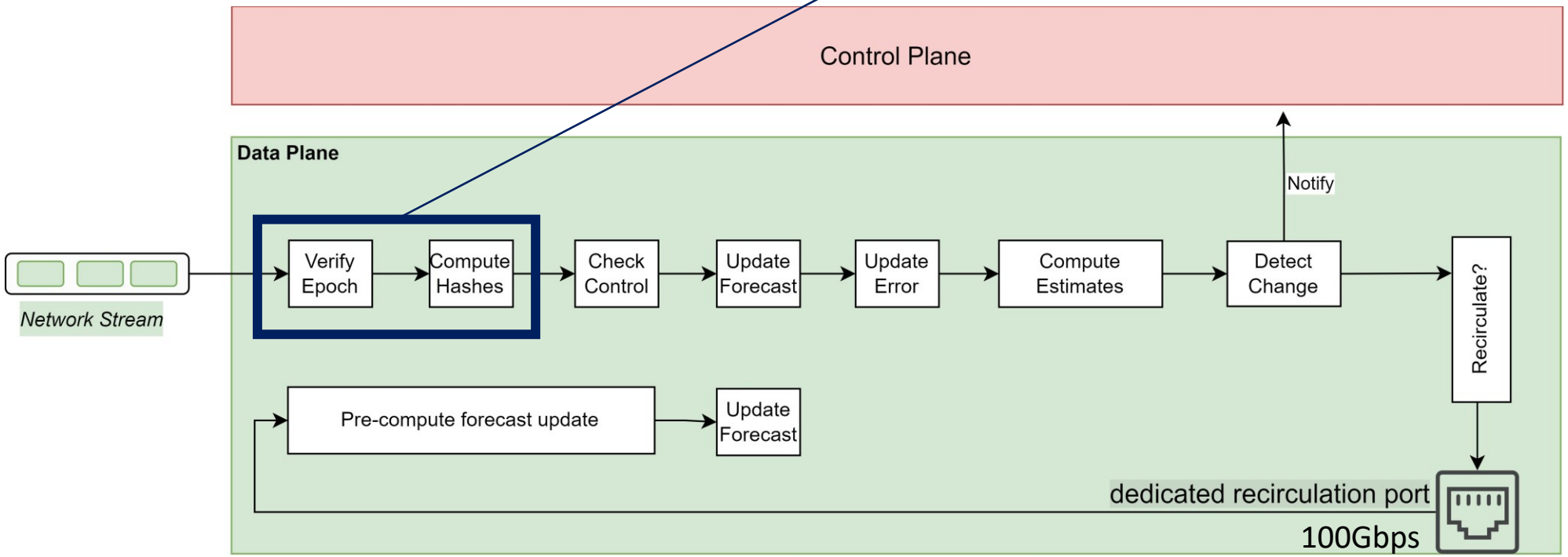
- Performs operations incrementally with each packet.
- Sends changes to the controller ad hoc.



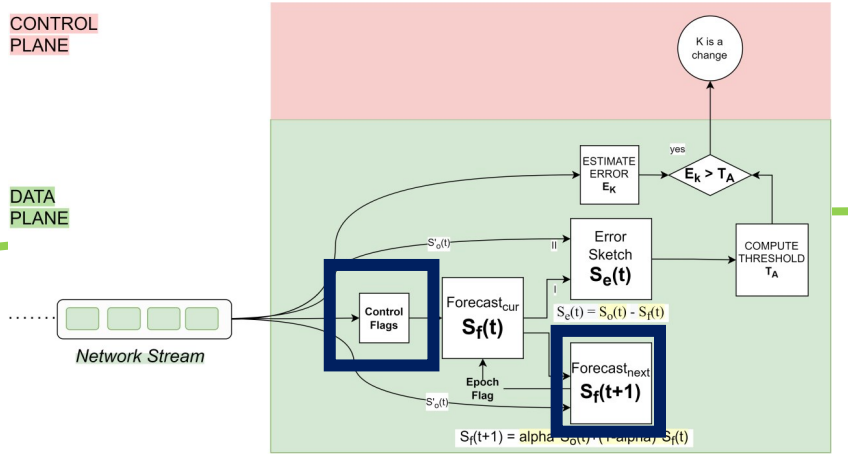
The k-meleon on TNA



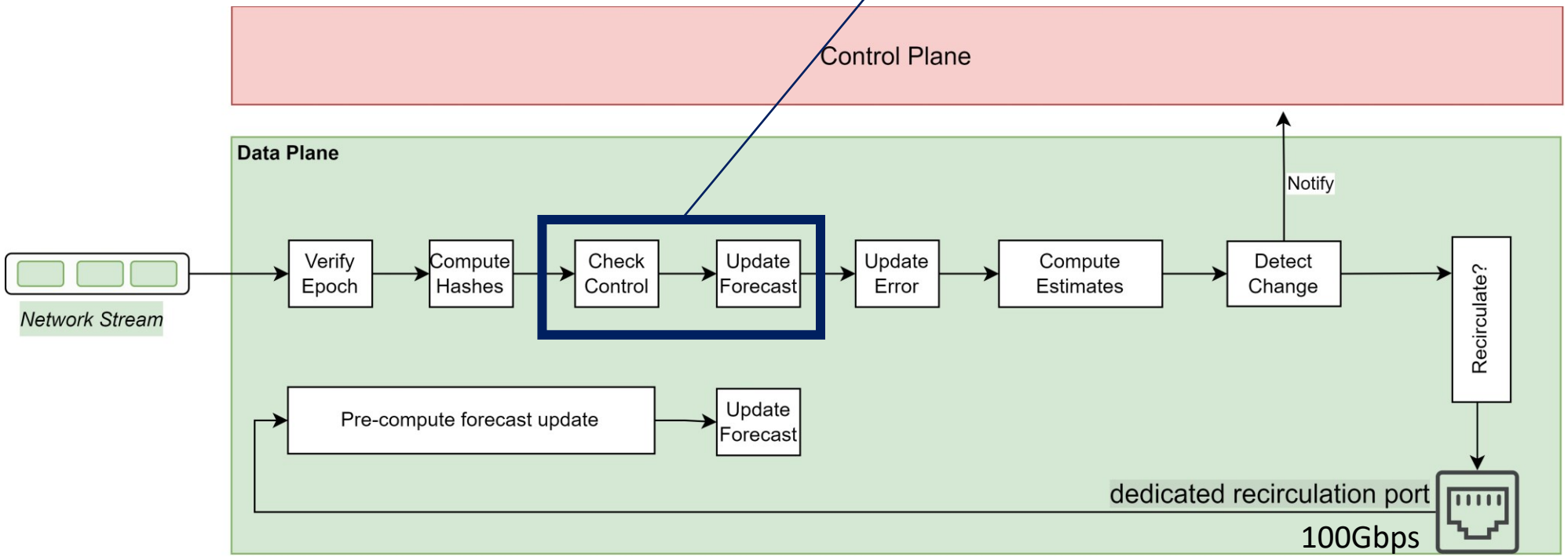
Change epoch if needed, compute hashes to index the sketches.



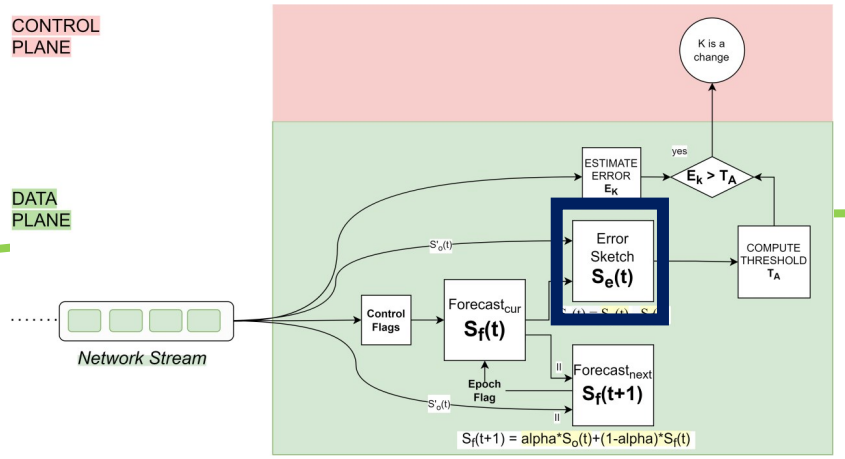
The k-meleon



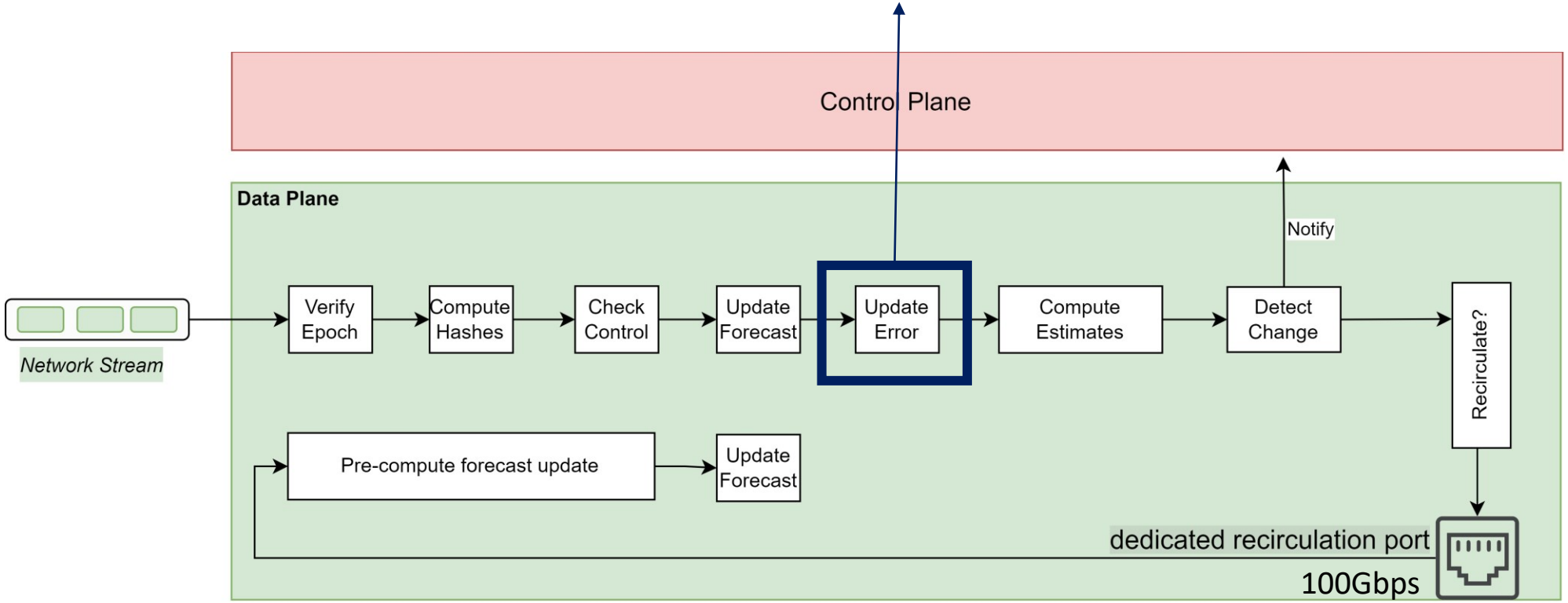
Check if epoch changed, and update forecast sketch accordingly.



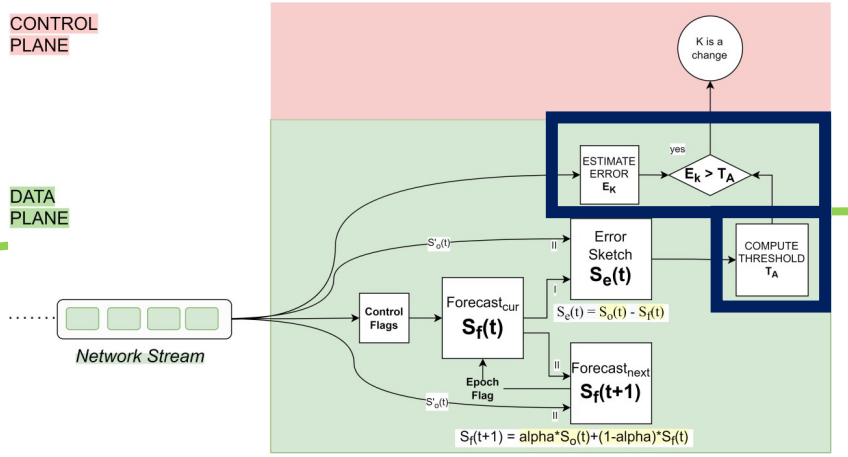
The k-meleon



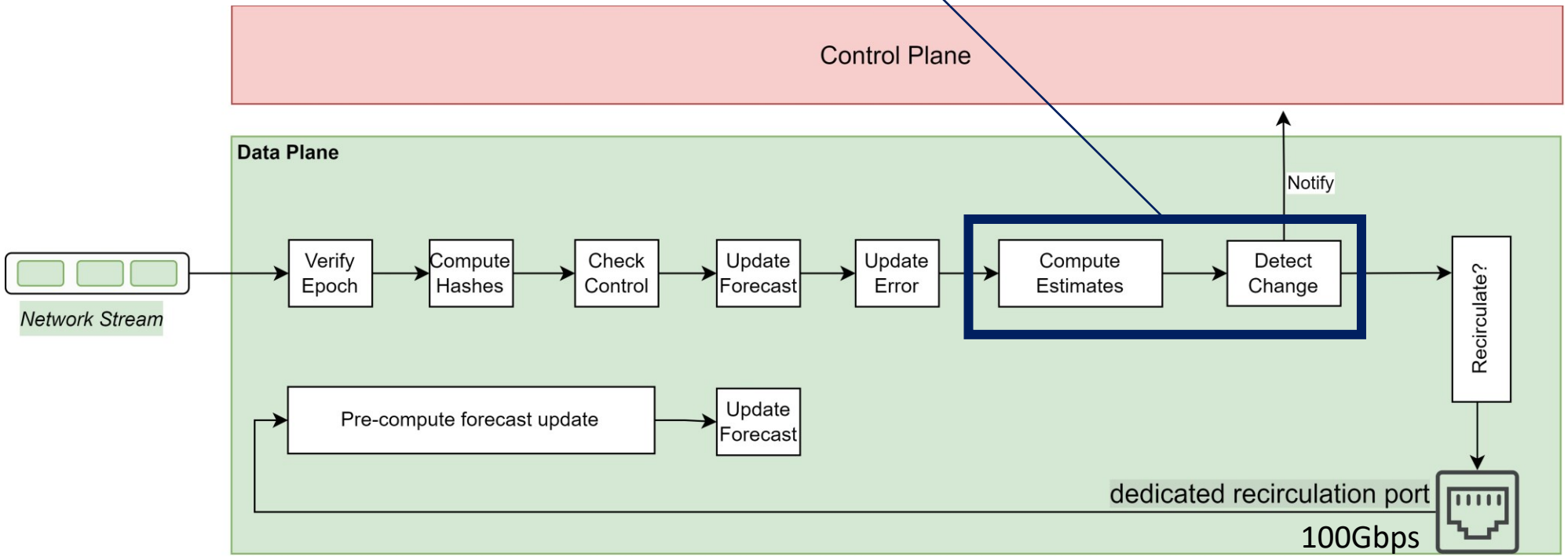
Update error sketch according to the control check.



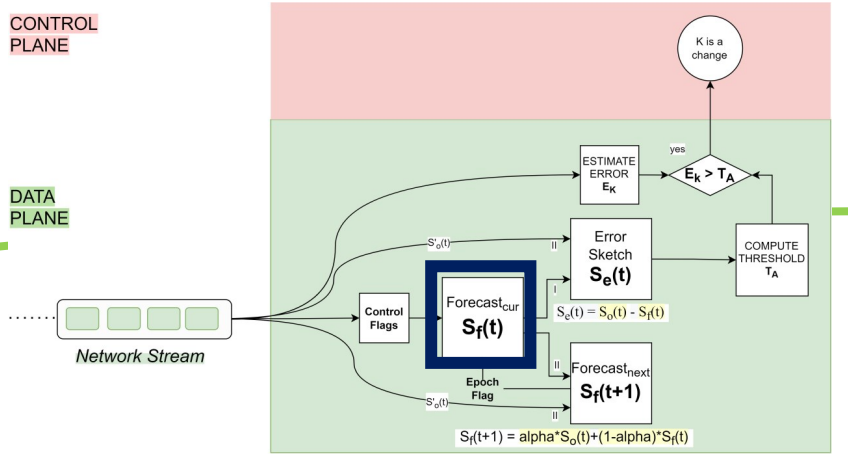
The k-meleon



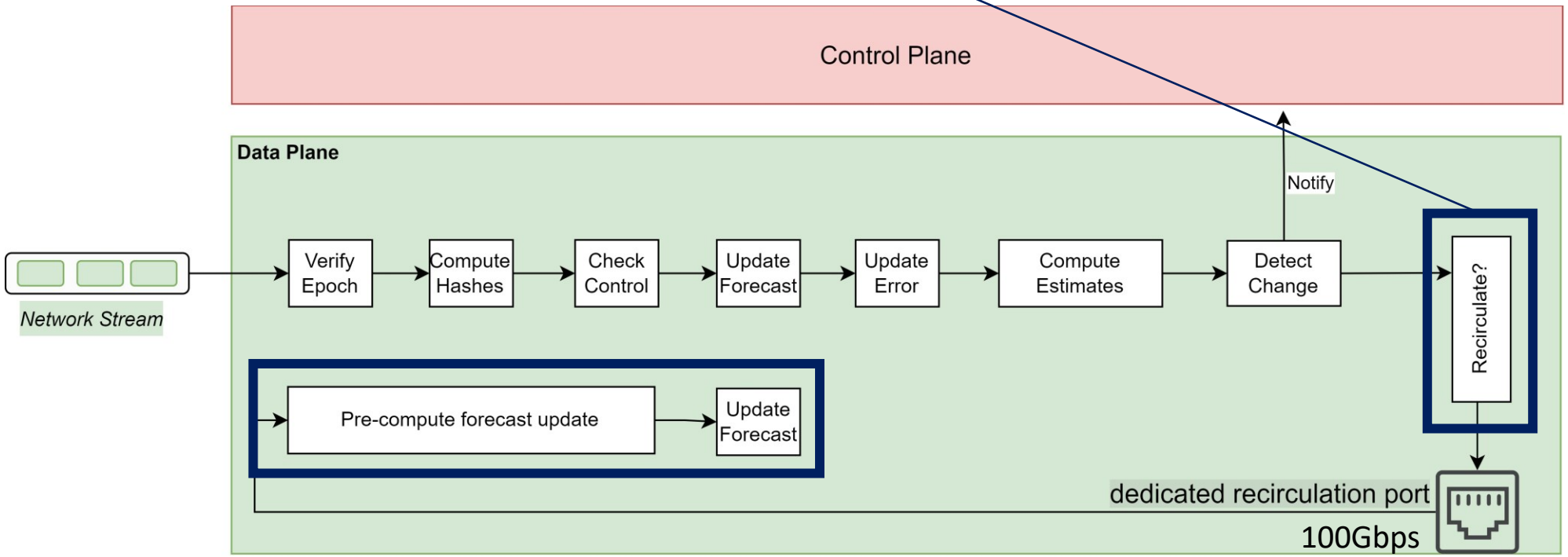
Compute estimates and perform detection.



The k-meleon



Recirculate to change epoch.



Evaluation

Questions

Q1. How does k-meleon perform compared to k-ary?

Q2. What is the expected performance of k-meleon?

Q3. What is the resource usage of k-meleon?

Experimental setup

The evaluation of k-meleon uses several packet traces from two datasets:

CSE-CIC-IDS2018[5] → Network attacks.

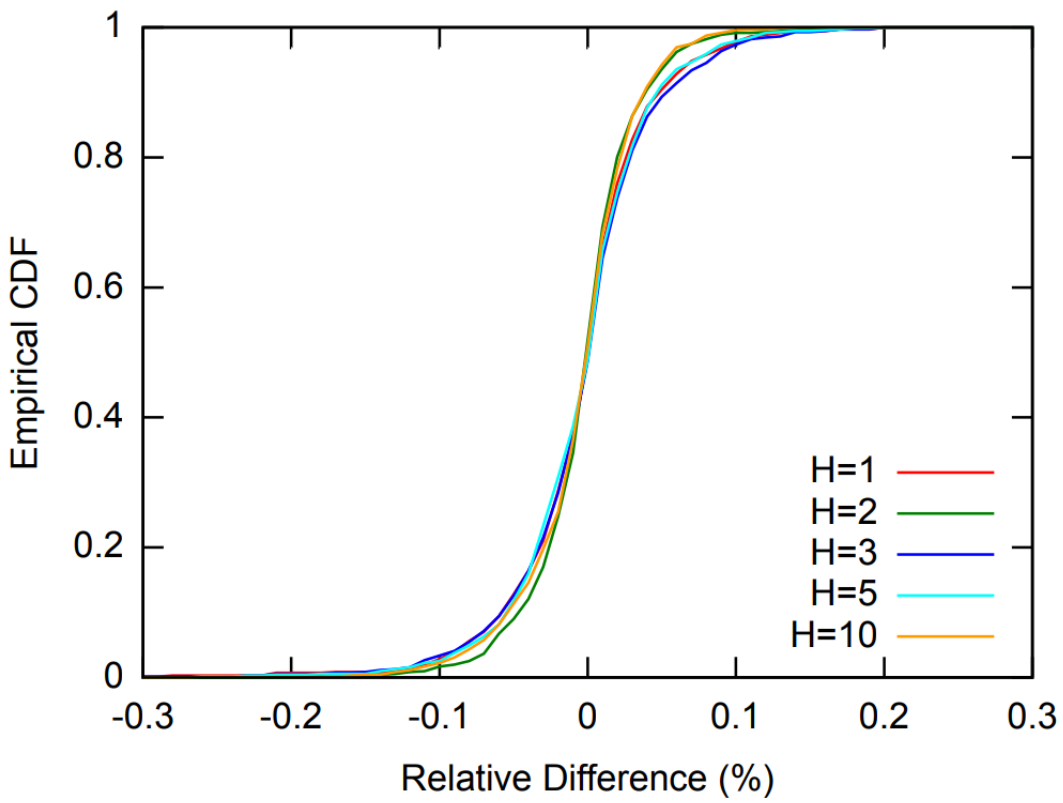
Data-center measurement (UN1)[6] → Microburst events.

Version	Purpose
K-ary in python	Establish a baseline
K-meleon in P4 for bmv2	Verify the correctness
K-meleon in P4 for TNA	Final target - Evaluation

Q1. K-meleon vs. k-ary

We vary the **size of the sketch**, which affects the number of complex computations performed.

Median relative difference is very low for any value of H (<0.2%).

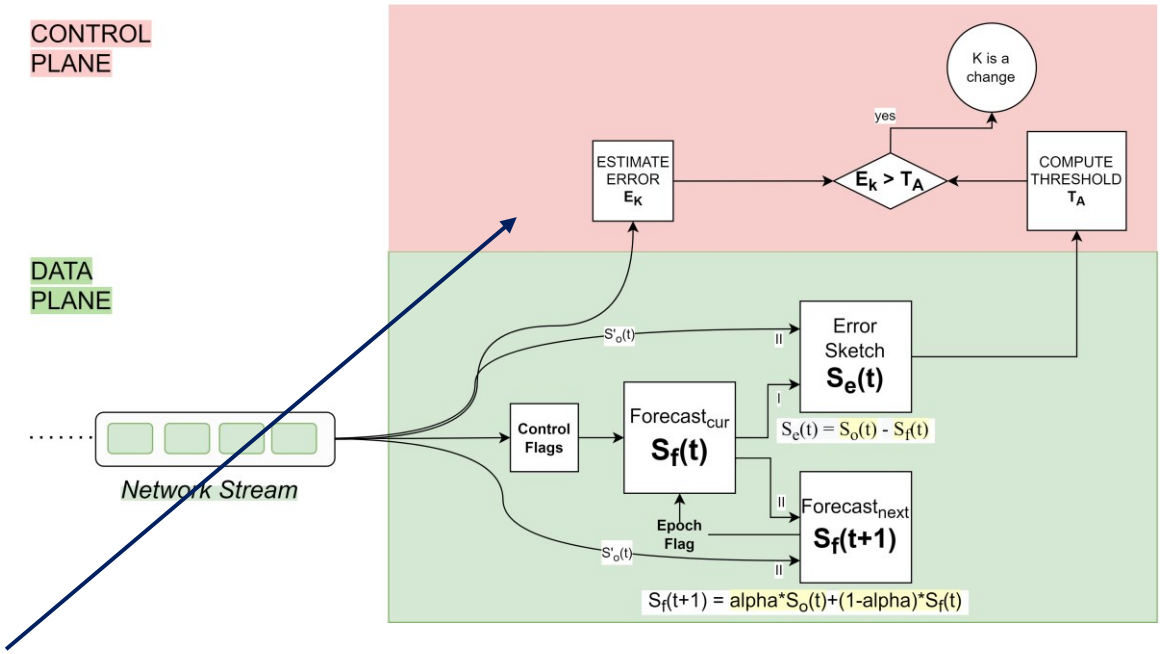


Q2. Performance

Our P4 prototype for TNA compiled successfully.

- Thus, it should run at line-rate, sustaining Tbps net traffic.

However, we have not yet moved all computations to the data plane.



Q3. Resource Usage on TNA

Only 7 stages are used by our preliminary prototype in P4 for TNA.

Only ingress is used for performing computation.

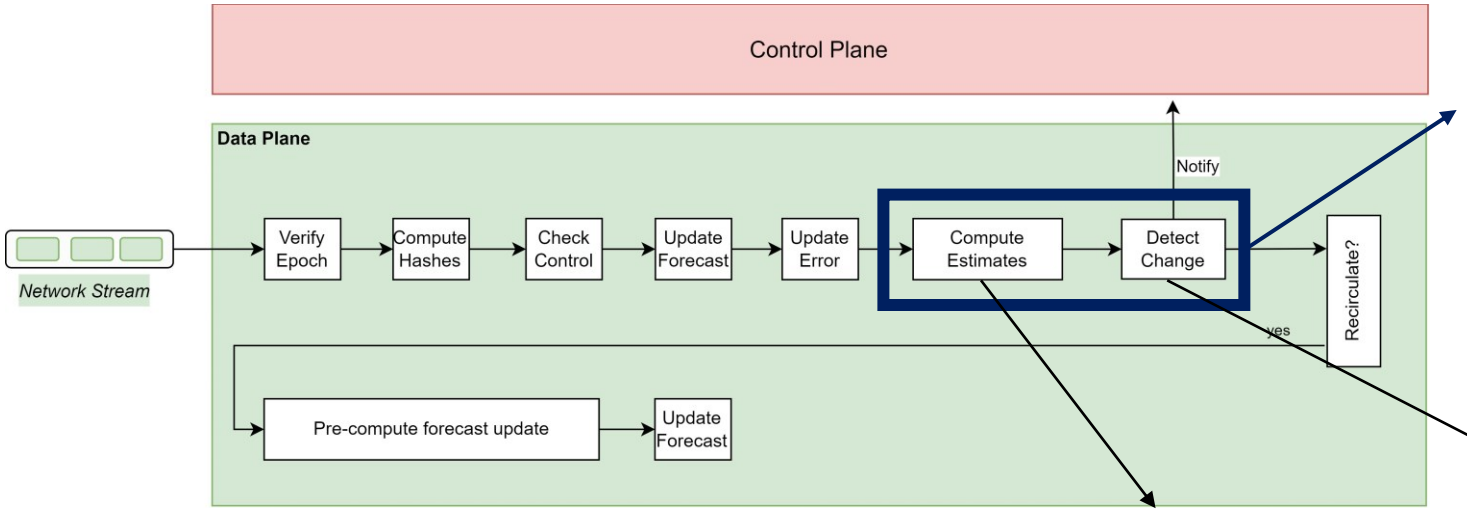
Computing	Usage
Stages	58%
Meter ALU	16.7%
Hash Dist Units	13.9%
VLIWs	4.7%
Memory	**Usage**
SRAM	2%
TCAM	0%

Conclusions

- The design of **K-meleon**, an *online change detection system* that leverages programmable switches.
- The implementation of a **prototype in P4** for Tofino Switch.
- An evaluation using the software switch bmv2 that demonstrates K-meleon achieves the same level of **accuracy** as the **k-ary algorithm**.

Discussion and future work

Detection entirely in the data plane



Still not implemented in the data plane.

But we still have several stages available in the ingress pipeline!

$$F_2^{\text{est}} = \text{median}_{i\in[H]}\{F_2^{h_i}\}$$

where

$$F_2^{h_i} = \frac{K}{K-1}\sum_{j\in[K]}(T_S[i][j])^2 - \frac{1}{K-1}(sum(S))^2$$

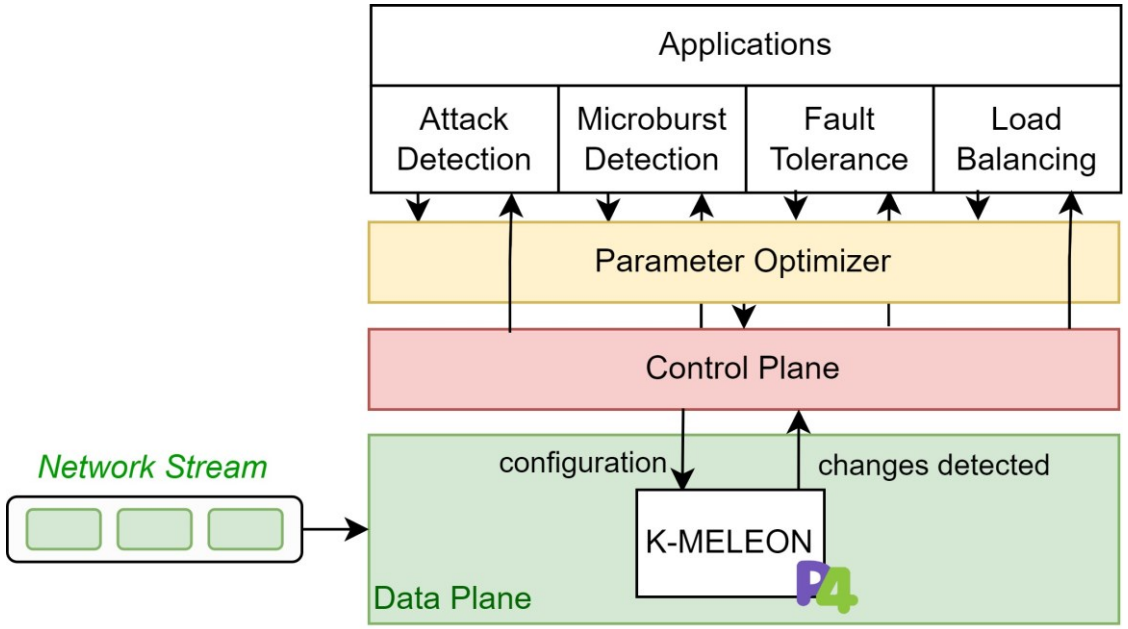
where

$$v_a^{\text{est}} = \text{median}_{i\in[H]}\{v_a^{h_i}\}$$

$$v_a^{h_i} = \frac{T[i][h_i(a)] - sum(S)/K}{1-1/K}$$

Discussion and future work

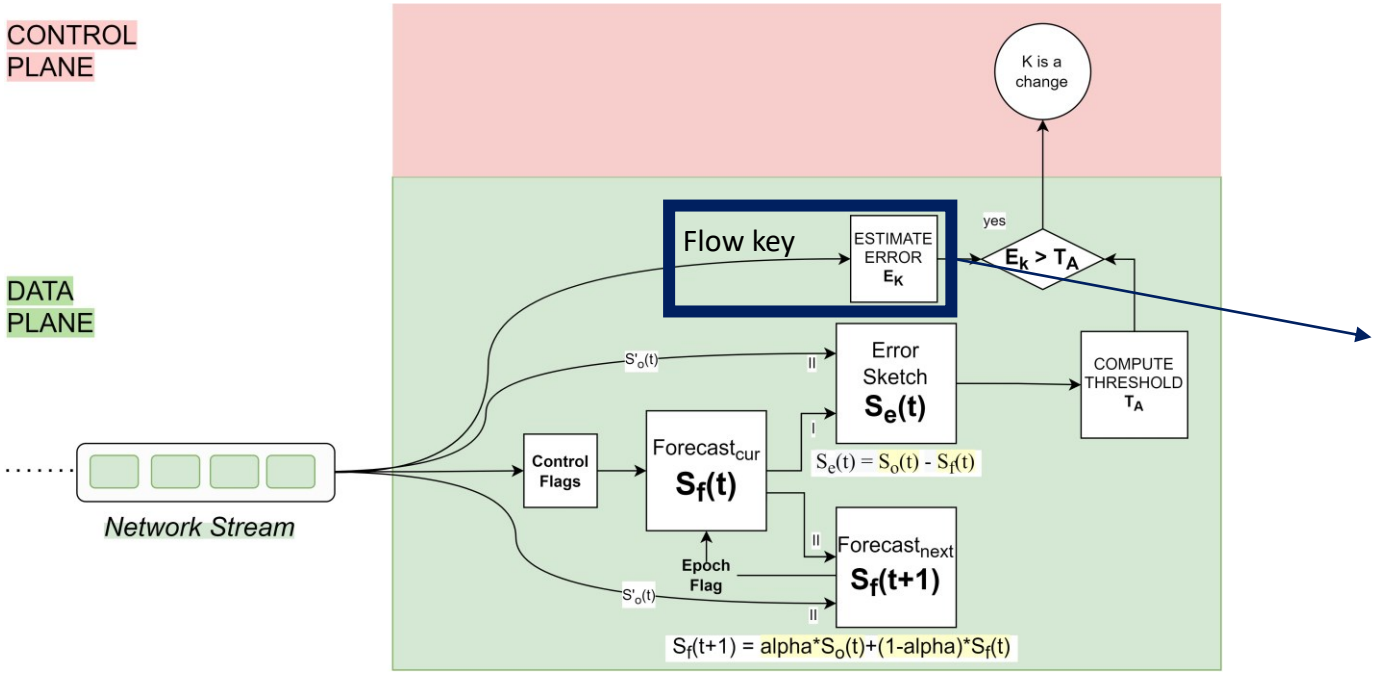
Use Cases



The tuning of some parameters improves detection accuracy of our primitive for different use cases.

Discussion and future work

Reversibility



The K meleon sketch does not maintain the culprit flow keys.

We are considering two options:

a) Using the flow keys directly from packets as they traverse the switch in the subsequent epoch.

b) Using a scheme based on some sort of voting mechanism (e.g., based on the MV Sketch[7]).

Thank You

Questions?

References

[1] Vibhaalakshmi Sivaraman et al. 2017. Heavy-hitter detection entirely in the data plane. In Proc. of ACM SOSR. 164–176.

[2] Tong Yang et al. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In Proc. of ACM SIGCOMM'18. 561–575.

[3] Qun Huang et al. 2018. Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference. In Proc. of ACM SIGCOMM'18. 576–590.

[4] Balachander Krishnamurthy et al. 2003. Sketch-based change detection: Methods, evaluation, and applications. In Proc. of the 3rd ACM IMC. 234–247.

[5] A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018) was accessed from https://registry.opendata.aws/cse-cic-ids2018.

[6] Chen, Xiaoqi, et al. 2018. Catching the microburst culprits with snappy. In Proc. of the Afternoon Workshop on Self-Driving Networks. 22-28.

[7] Lu Tang, et al. 2019. MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams. In Proc. Of the IEEE INFOCOM 2019. 2026-2034