Congestion control and Heavy hitter detection

Belma Turković, Jorik Oostenbrink, and Fernando Kuipers

TU Delft

Heavy Hitter Detection in the Dataplane

TU Delft

Heavy Hitter Detection…

- Detect flows with large traffic volumes
- Many applications, e.g.:
  - DoS and anomaly detection
  - Flow-size aware routing
- Heavy hitters form most of the traffic

  → Most important for traffic engineering

TU Delft

… in the Dataplane

- Avoid traffic sampling (e.g. NetFlow)
  - Slower detection time
  - Possibility of false negatives
- Faster reaction time – apply actions as packets are forwarded
- However:
  - Requires (more expensive) specialized hardware

Problem Statement

- Given the last *N* packets of an incoming stream and flow *f*
- Determine if *f* has a frequency above threshold
- No false negatives
- Probability of false positive should be ≤ ε

TU Delft

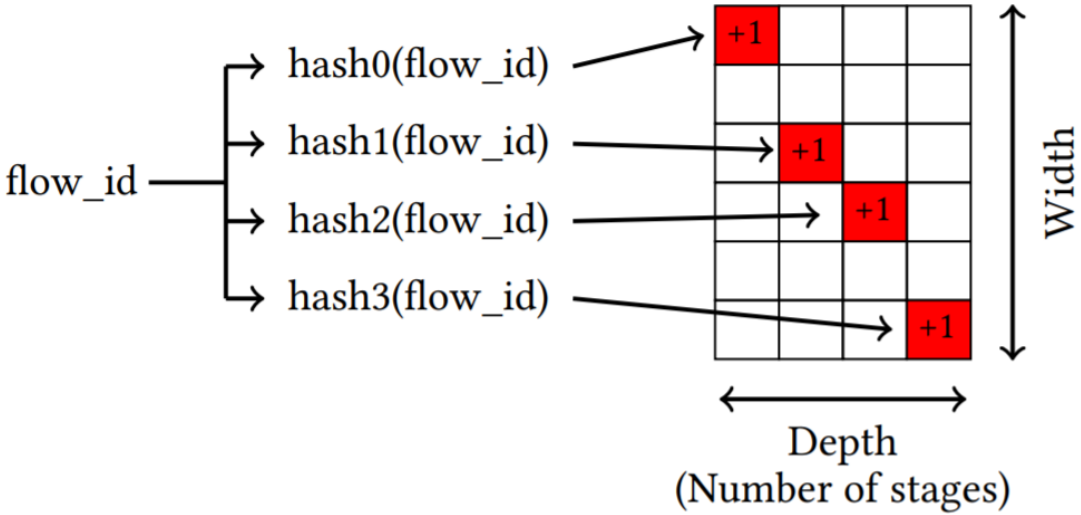
Additional Requirement

- Process packets as quickly as they arrive
- Severely limits processing time
- Limited memory :
  - typically just one read-modify-write action per register array
  - Limited memory available per stage (~1.5MB available for both forwarding and heavy hitter detection app)

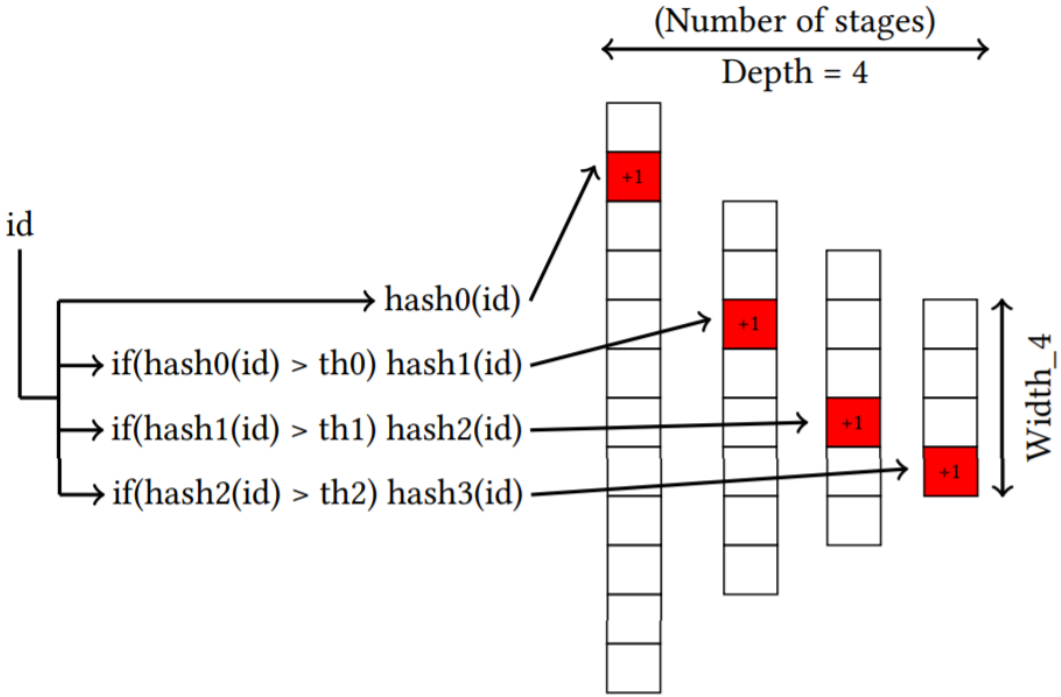
Sketches

- Compact data structure
- Only stores summary information
- Low memory usage
- Often tuneable in accuracy vs memory usage
- E.g. bloom filter

Count-Min sketch

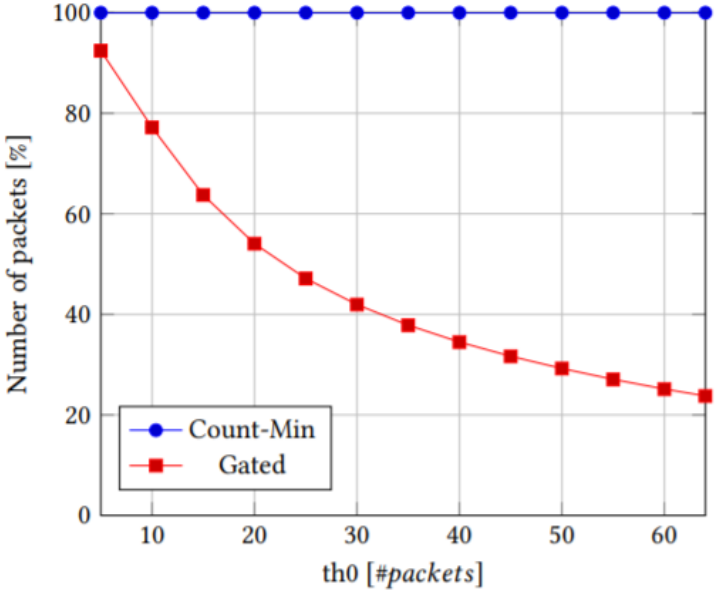


Gated Sketch

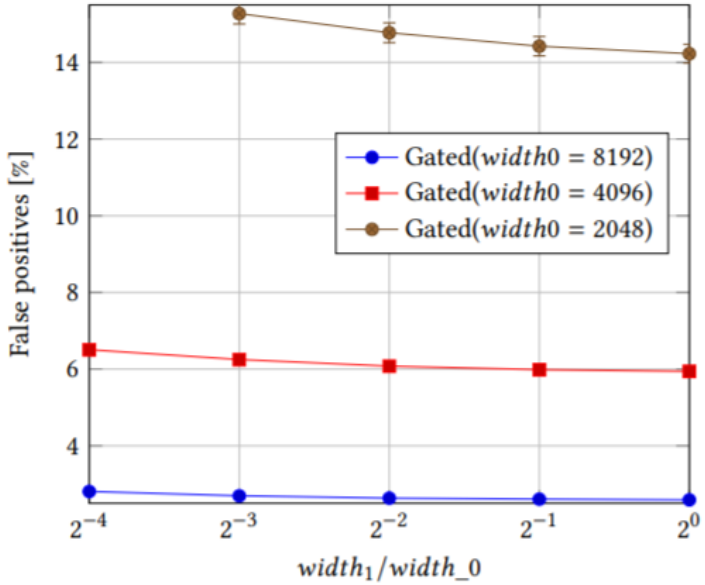


Gated Sketch

%packets processed by second stage vs threshold of the first stage

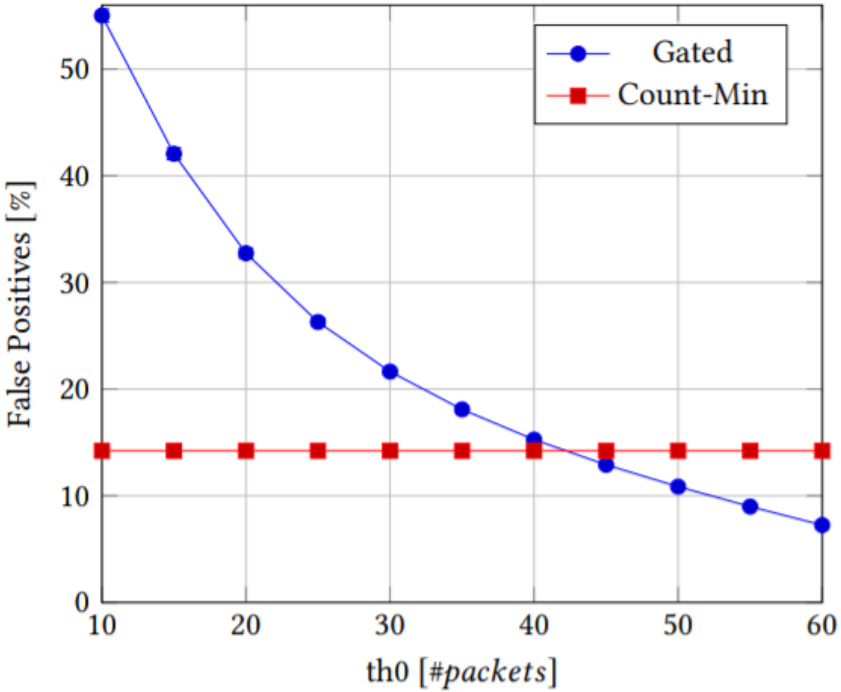


Accuracy vs width second stage



Gated Sketch

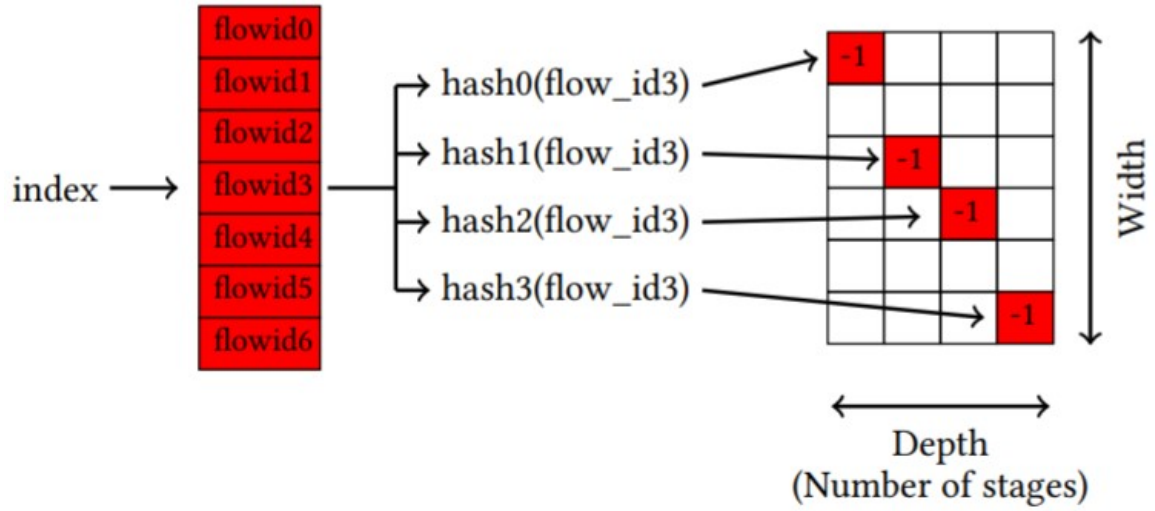
Effect of Threshold on First Stage



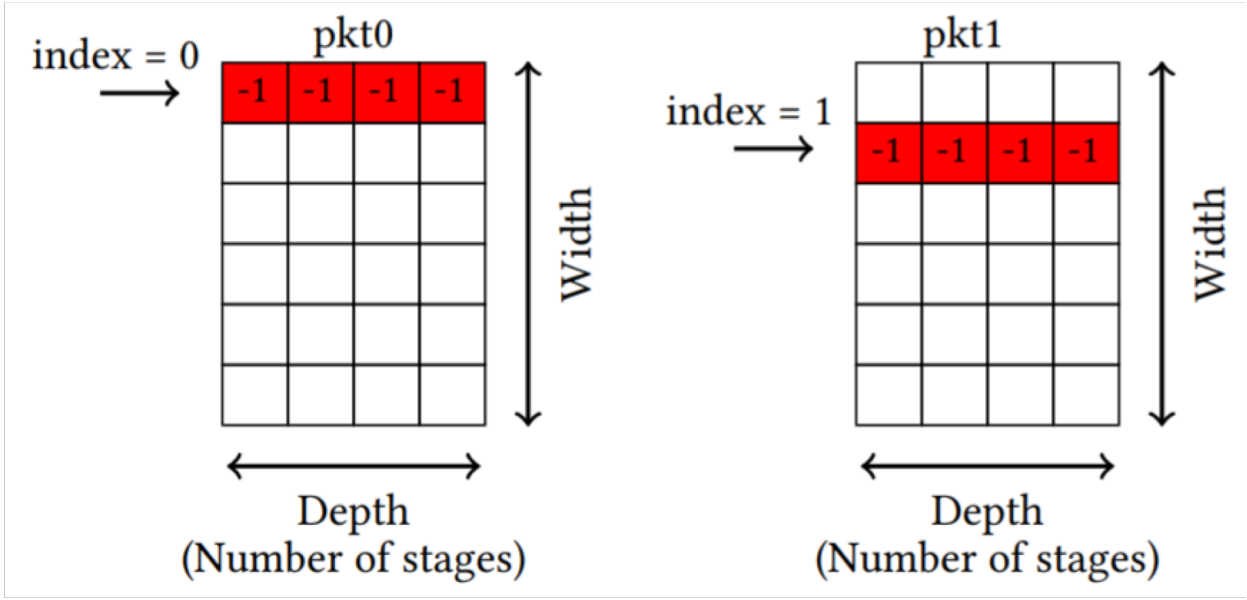
Window structure

- Remove outdated flows and counts from the counting sketch
- All current dataplane approaches flush the registers every x seconds
  - Increases the number of false negatives and false positives
  - Additional actions from the control plane

Ring Window


index
flowid0
flowid1
flowid2
flowid3
flowid4
flowid5
flowid6
hash0(flow_id3)
hash1(flow_id3)
hash2(flow_id3)
hash3(flow_id3)
-1
-1
-1
-1
Width
Depth
(Number of stages)

Sequential Window



New Window approach

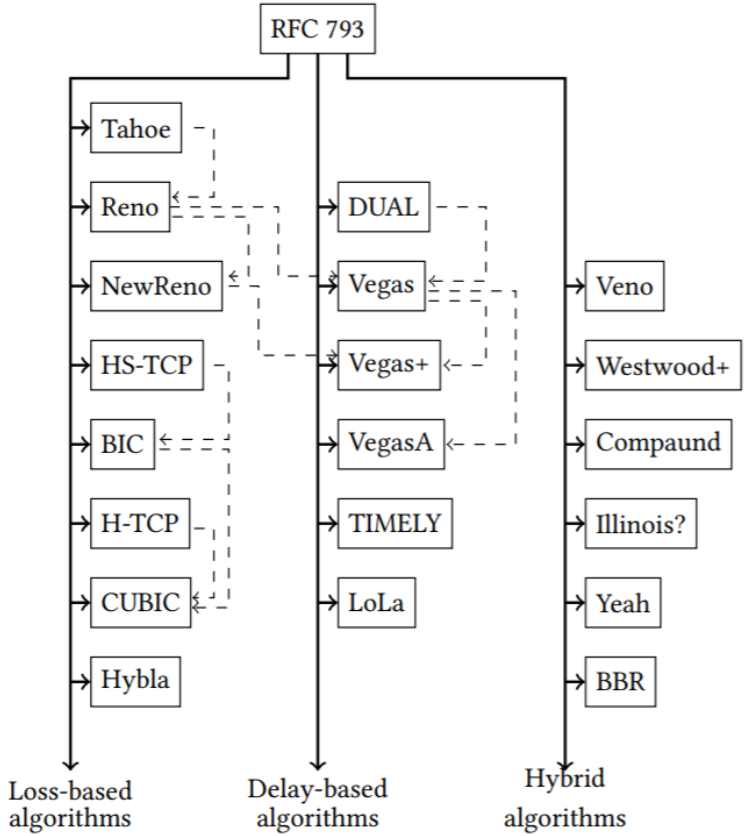
- One memory access per register array (read/write/modify)
- Lower memory usage
- Similar accuracy

TU Delft

Congestion Control and Avoidance

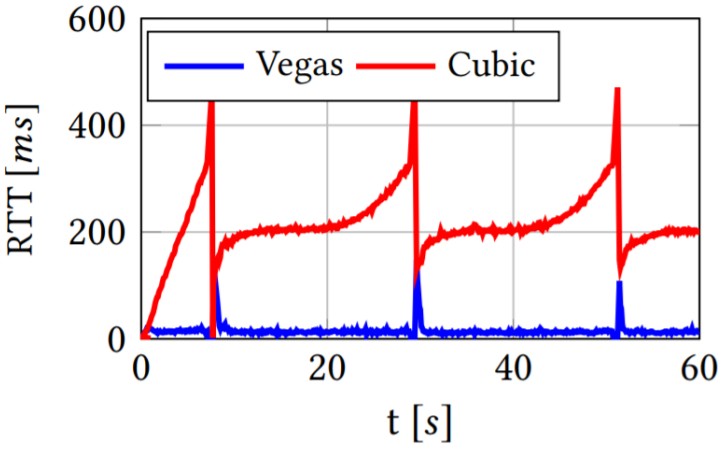
Congestion control at transport layer

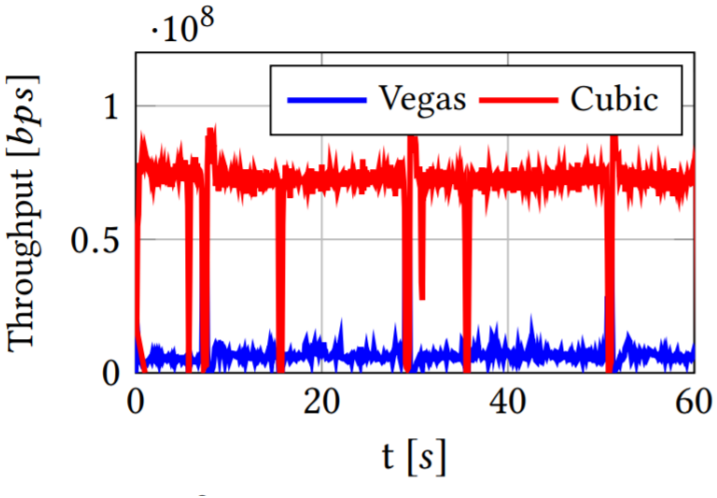
- Clasification:
  - Loss-based
  - Delay-based
  - Combination



Interoperability

- delay based algorithms <u>can not compete</u> with loss-based algorithms



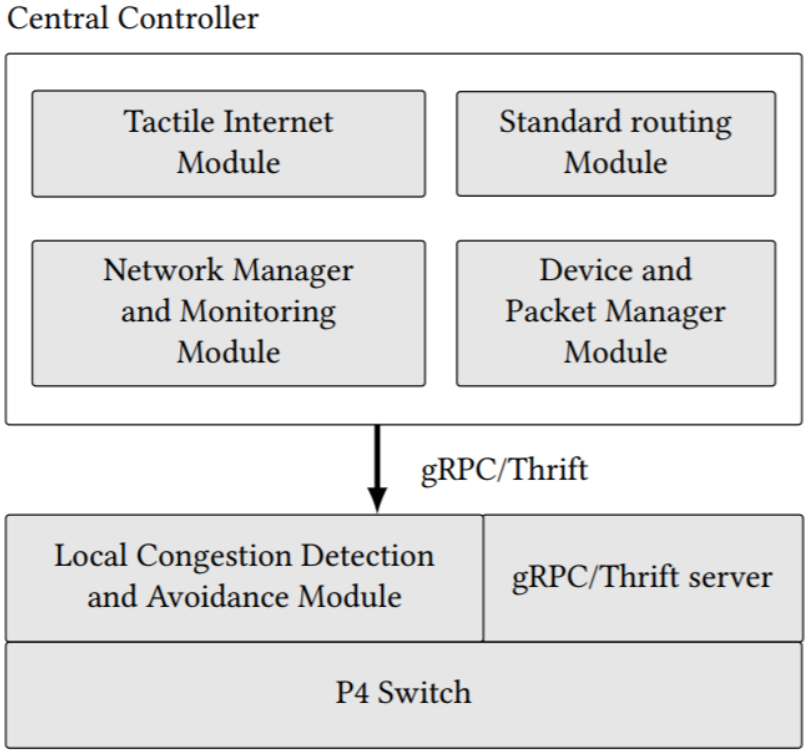


Problem statement

How to enable congestion control and avoidance **in the forwarding nodes**, instead of at the source or via a controller?

Hierarchical control plane

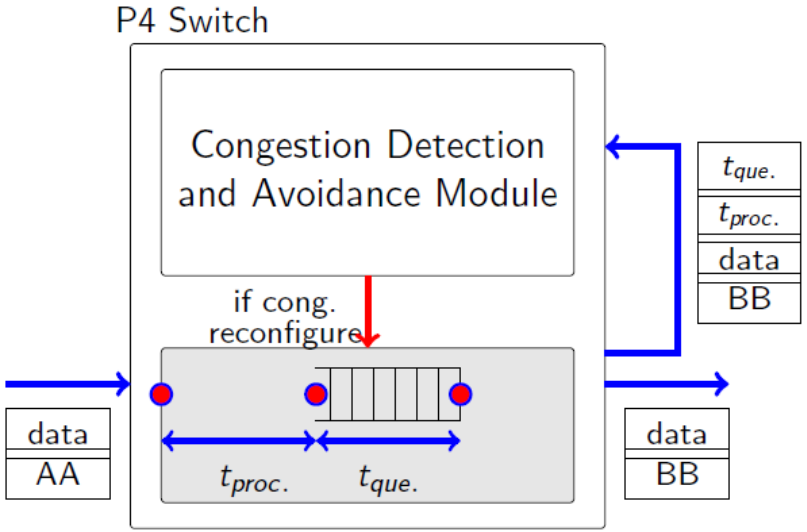
- **Central controller** – Configures and monitors paths for different service classes
- **Local controller** – Reduces congestion detection and reaction time for latency sensitive flows



Local congestion detection

Every network node collects statistics for low latency flows, such as:

- Processing delay
- Queuing delay
- Enqueue length
- Dequeue length
- Number of packets affected



If these values reach preconfigured thresholds, congestion is detected!

Congestion avoidance

- Inform the traffic sources about congestion using ECN
- Reroute latency sensitive traffic to a non-congested backup path

Conclusion

- Main advantages:
  - detection time is reduced and congestion is detected per flow
  - after detection, the reaction time is minimized, as a local controller intervenes by configuring a better route.
- Possible extension to a hybrid network where only some nodes are programmable

Questions/Comments/Suggestions?

- Contact Info:
  - Belma Turković (B.Turkovic-2@tudelft.nl)
  - Jorik Oostenbrink (J.Oostenbrink@tudelft.nl)
  - Fernando Kuipers (F.A.Kuipers@tudelft.nl)

TU Delft