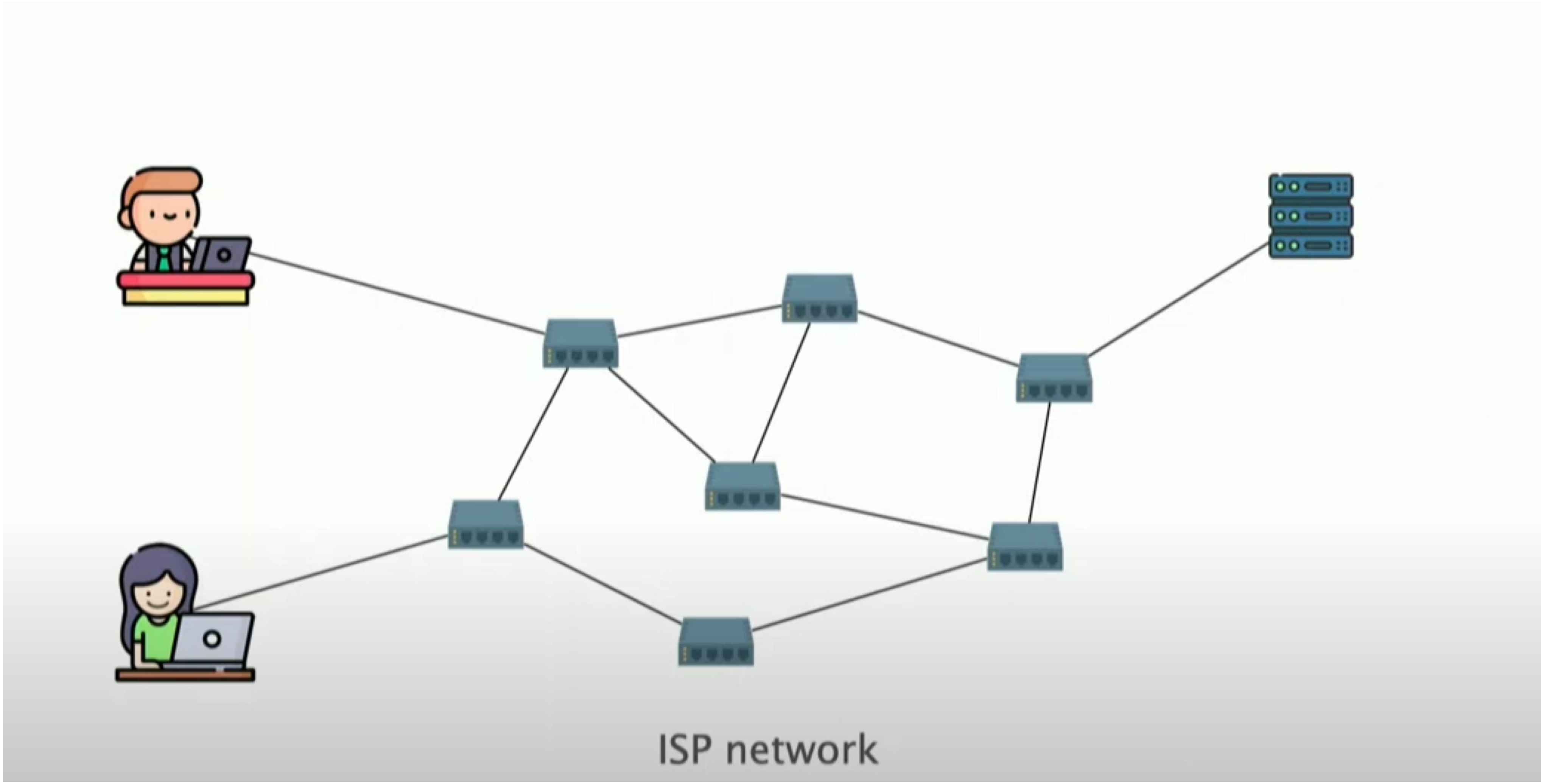
FAst In-Network *GraY* Failure Detection for ISPs

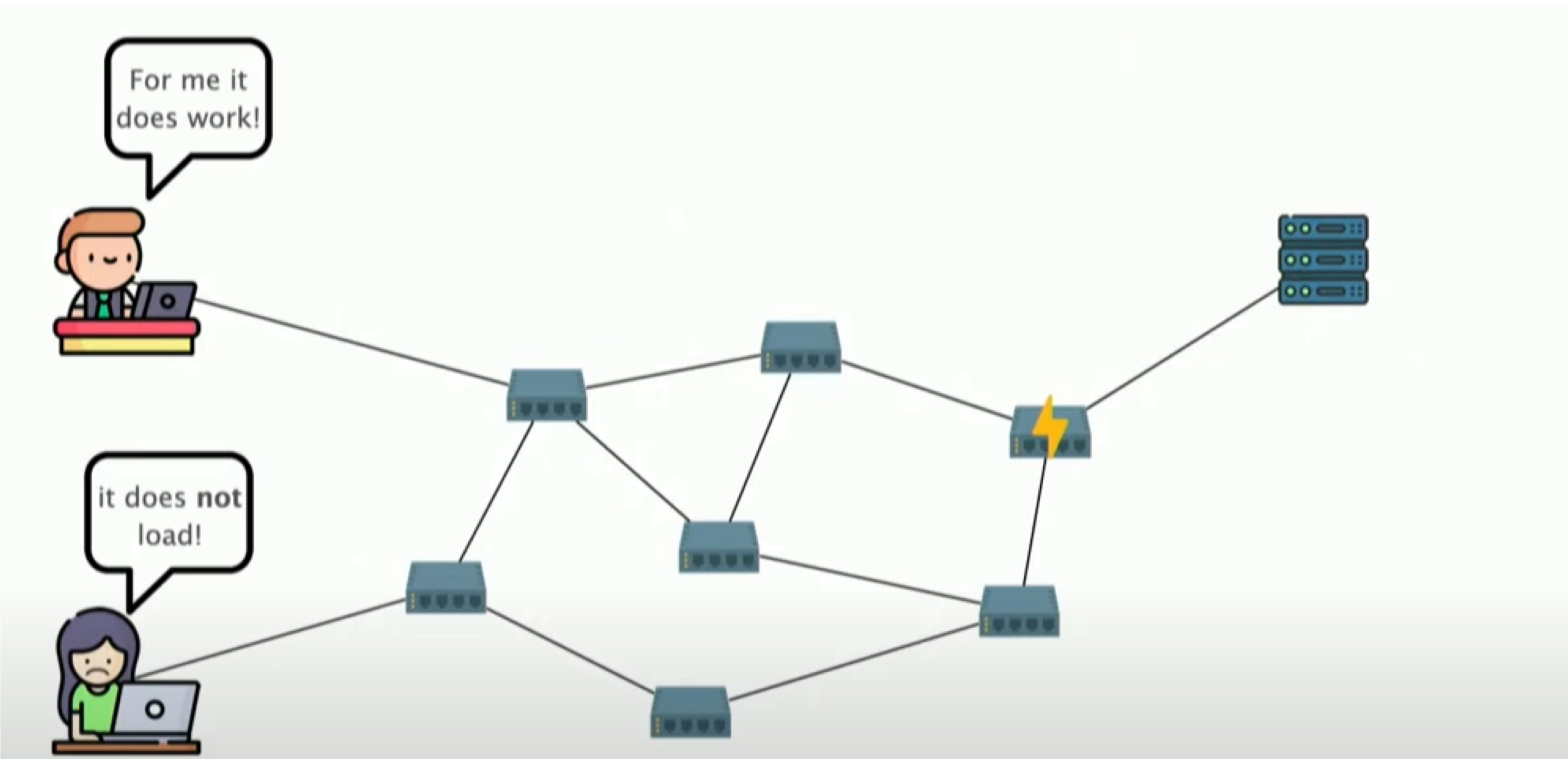
Edgar Costa Molero
ETH Zurich
cedgar@ethz.ch

Stefano Vissicchio
University College London
s.vissicchio@ucl.ac.uk

Laurent Vanbever
ETH Zurich
lvanbever@ethz.ch

Gray failures are ***Permanent*** packet loss caused by ***a* malfunctioning device** affecting a ***subset of the traffic***


ISP network


For me it does work!
it does **not** load!

Gray failures are ***Permanent*** packet loss caused by ***a*** **malfunctioning device** affecting a ***subset of the traffic***

Gray failures…

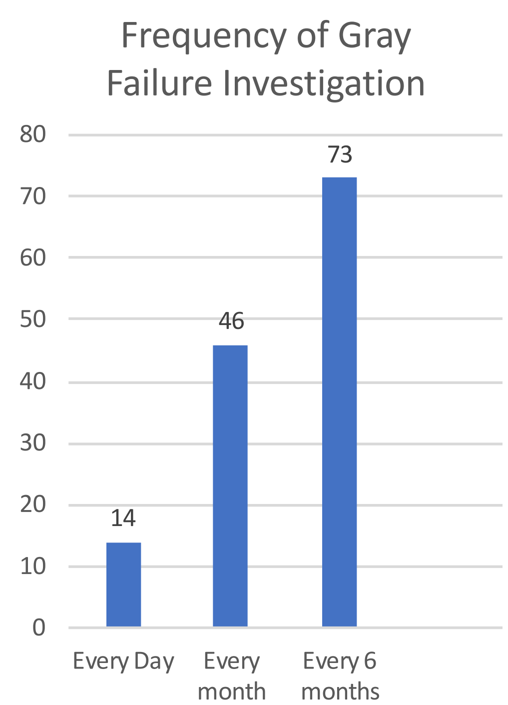
can be caused by

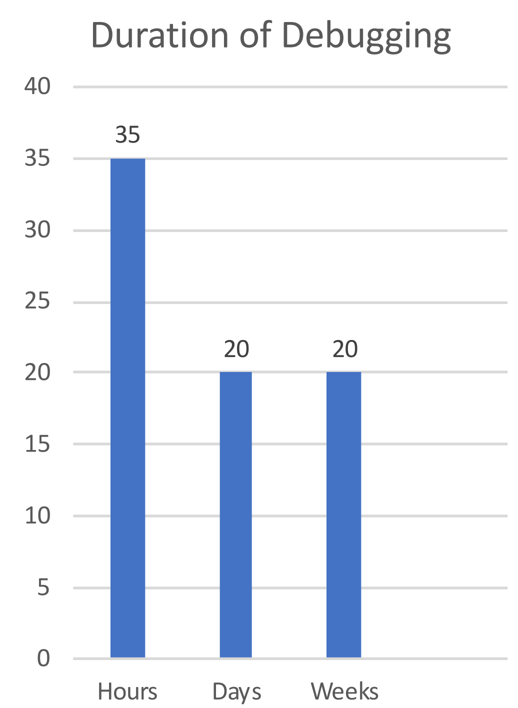
- TCAM bit flips and memory corruption
- bent fibers and not well seated line-cards
- CRC checksum errors
- software bugs and misconfigurations

can affect

- ***single, some*** or ***all*** traffic entries
- ***some*** or ***all*** the packets

Gray failures are a problem for a majority of operators





Detecting and ***locating*** ***gray*** failures requires ***two*** operations

1 **to collect** statistics of ***all*** the traffic

2 ***to compare*** the statistics

Existing ***ISP*** monitoring techniques fall short because they do not ***collect*** statistics on all the traffic

active	x	**Heartbeat protocols (e.g. BFD)**
		only the heartbeat packets
	x	**Sending traffic probes**
		only selected probes
passive	x	**Packet counters (e.g. SNMP)**
		only available switch counters
	x	**NetFlow or sFlow**
		only if sampled

Most ***data center*** ***gray*** failure detection solutions do ***collect statistics*** on all traffic and ***compare*** them.

However, they still ***fall*** short in ***ISP networks.***

The characteristics of ***ISP networks*** make ***data center*** failure detecting systems ***not operational***

- **No end-point control**

  only control network devices

- **High link bandwidth**

  100 Gbps and increasing

X

- **High latency between devices**

  in the order of ms

Data center *gray* failure detection systems require more ***memory*** than available in switches to operate in ***ISP networks***

required memory to operate

collection rate **X** ***collection*** complexity **X** ***min*** time to ***compare*** **>>** **switch memory**

collection rate		***collection*** complexity		***min*** time to ***compare***		**switch memory**
link bandwidth	X	*memory required per packet*	X	*link delay* *control plane reading speed*	>>	*fixed*

Introducing

FANcY: Fast In-network ***Gray*** Failure Detection for ISPs

We designed FANcY to work with **ISP network characteristics**

required memory to operate

---

collection rate **X** ***collection*** complexity **X** ***min*** time to ***compare*** **<** **switch memory**

collection rate		***collection*** complexity		***min*** time to ***compare***		**switch memory**
link bandwidth		*memory required per packet*		*link delay* *control plane reading speed*		*fixed*

We designed FANcY to work with **ISP network characteristics**

required memory to operate

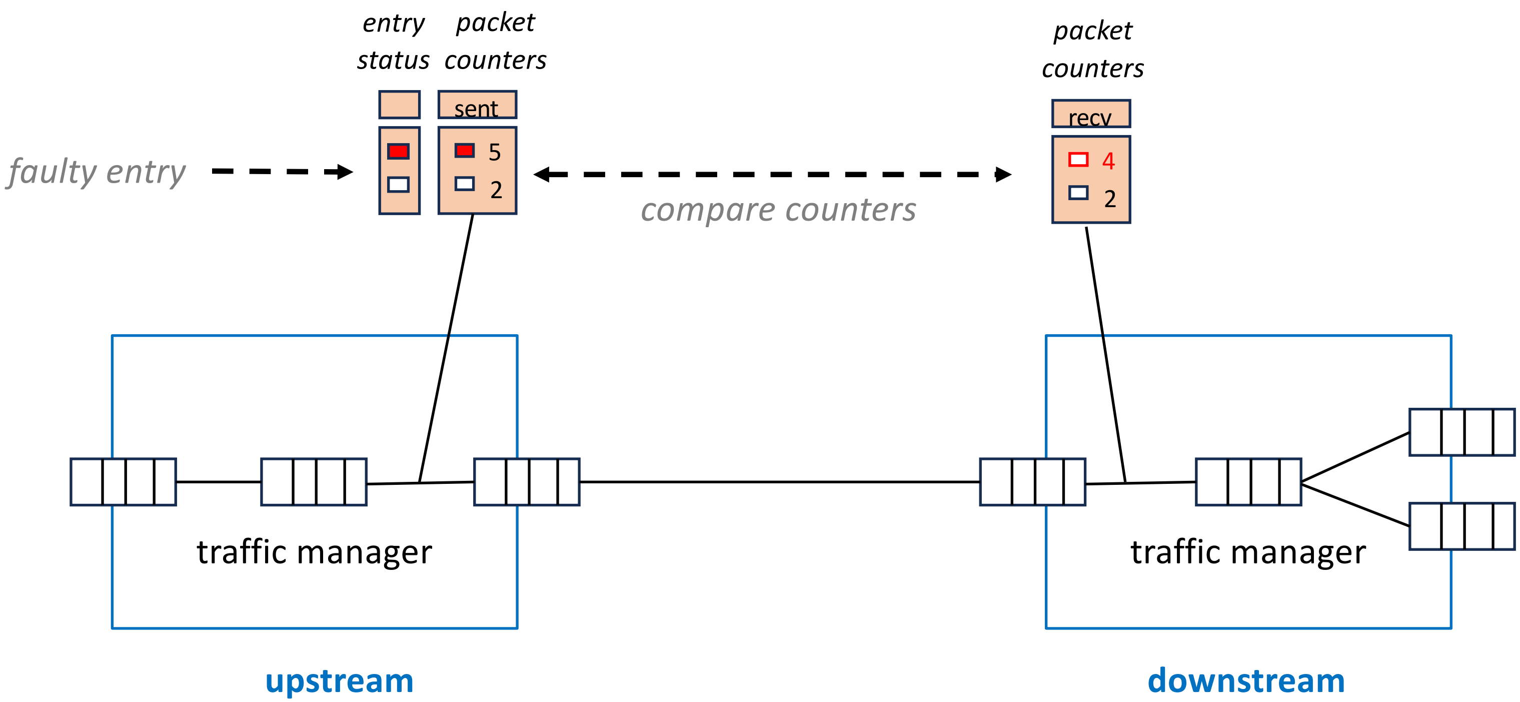
collection rate X ***collection*** complexity X ***min*** time to ***compare*** < **switch memory**

- ***collection*** rate: *link bandwidth*
- ***collection*** complexity: *memory required per packet*
- ***min*** time to ***compare***: *link delay*, ~~*control plane reading speed*~~
- **switch memory**: *fixed*

#1 Collected statistics are aggregated per traffic entry in simple counters

#2 FANcY compares the collected statistics directly in the data plane

If counters mismatch, the ***upstream*** flags the entry as ***faulty***



Our design has ***two*** main ***challenges***

#1 Synchronizing *our packet counts and make them reliable*

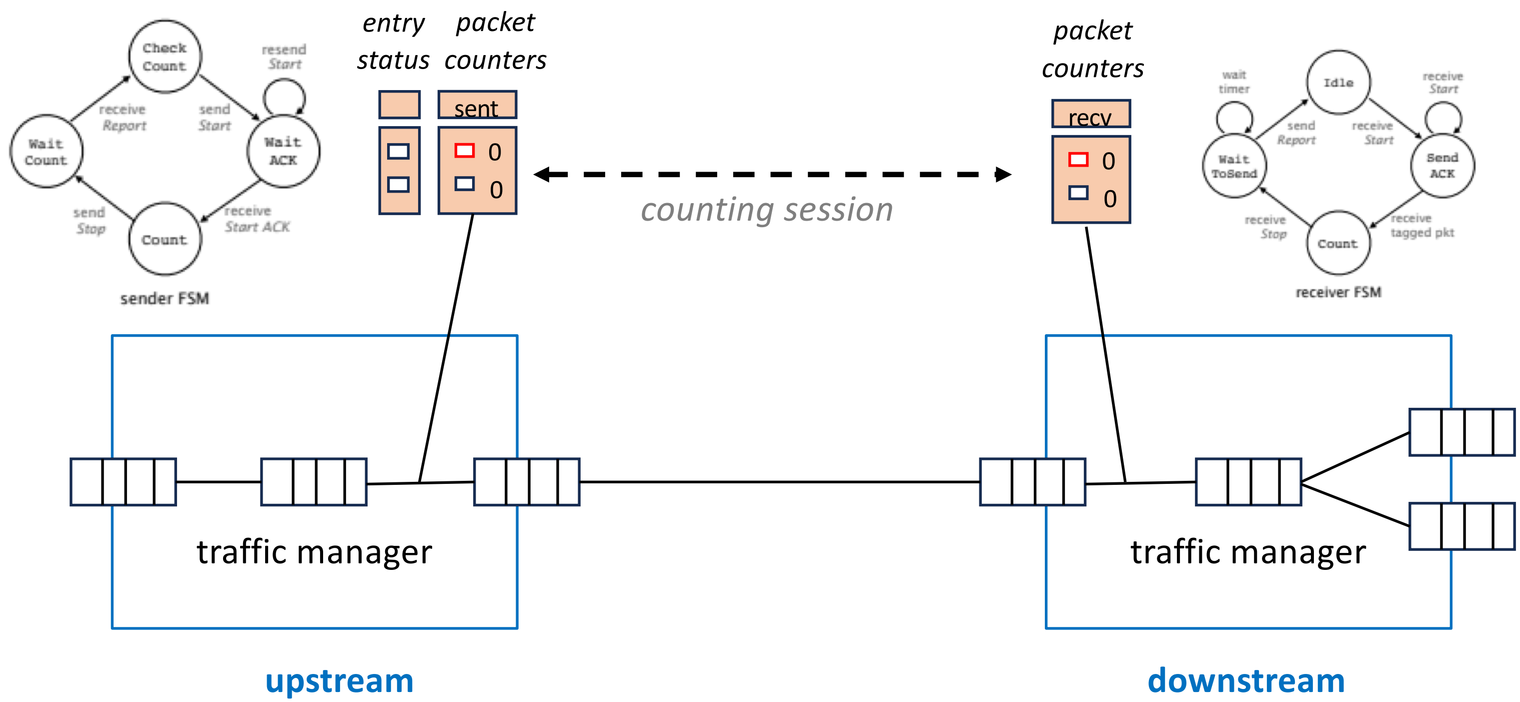
FANcY establishes counting sessions for each counter pair

#2 Scaling to *many traffic entries*

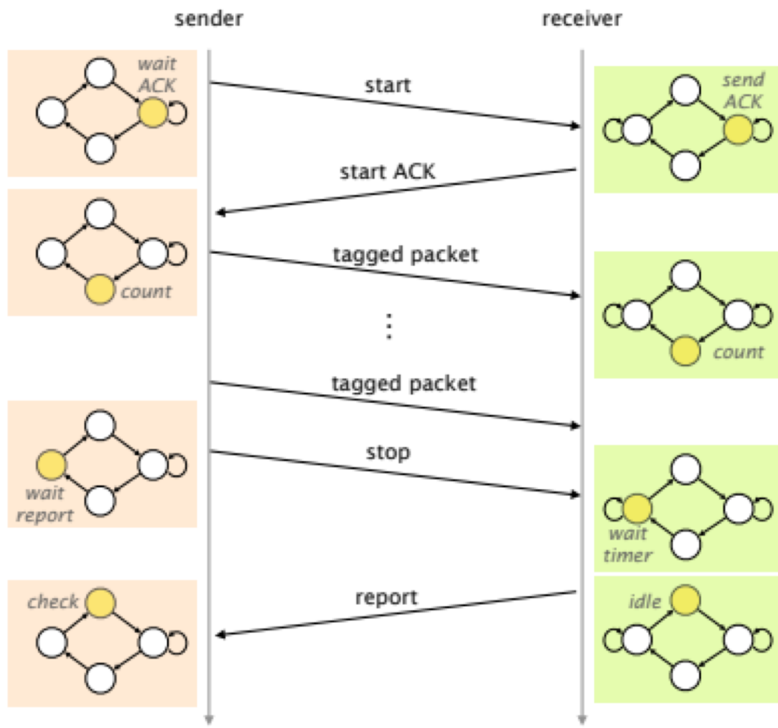
FANcY uses a hybrid approach to support a big number of entries

To achieve perfect **synchronization** and **reliability**

FANcY uses **state machines** for each counting session



Time sequence diagram showing the implementation of a counting session with ***FANcY*** state machines



Our design has ***two*** main ***challenges***

#1 Synchronizing *our packet counts and make them reliable*

FANcY establishes counting sessions for each counter pair

#2 Scaling to *many traffic entries*

FANcY uses a hybrid approach to support a big number of entries

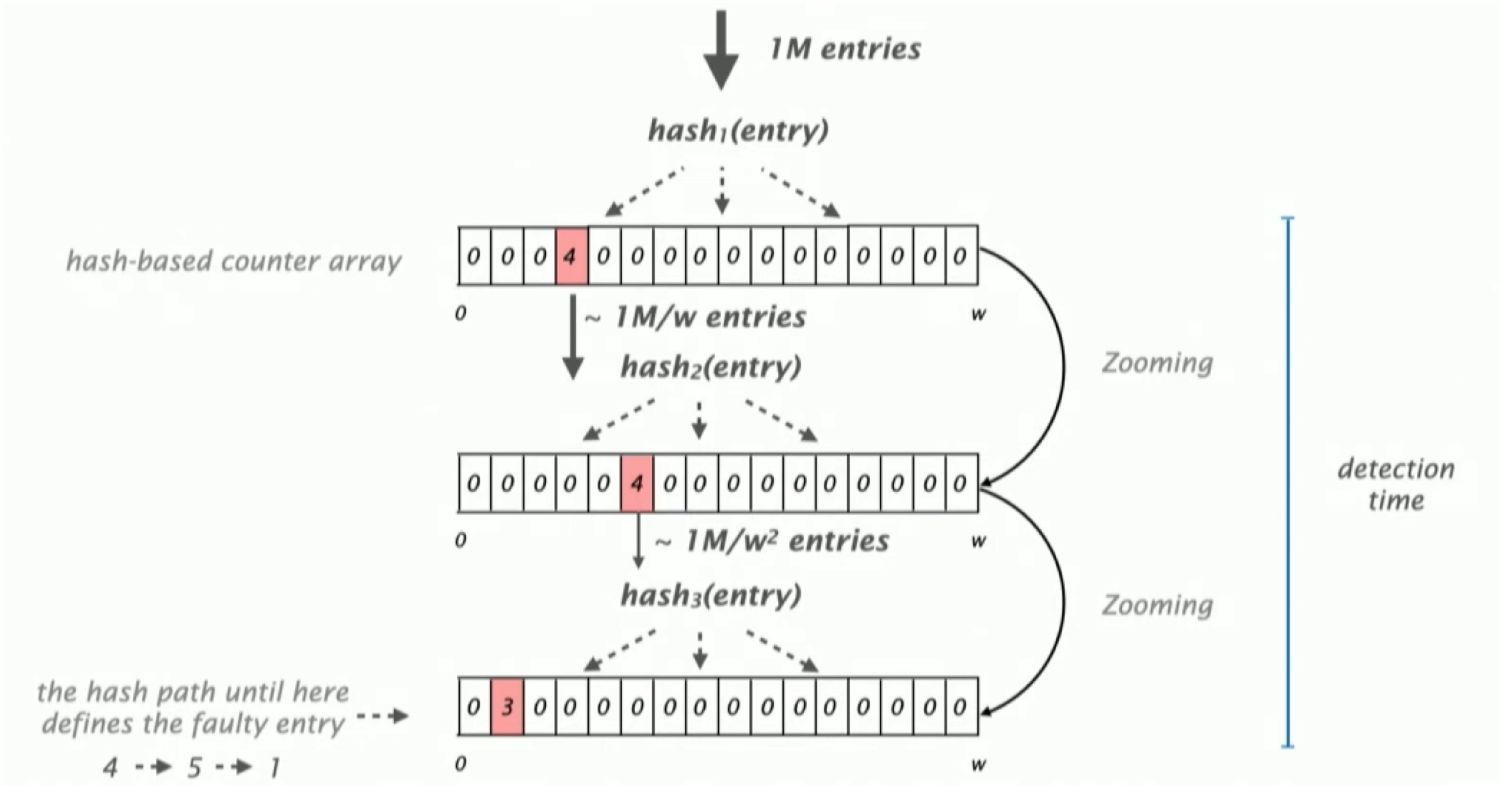
Having a ***pair of counters*** and ***state machines*** per traffic entry ***does not scale***

Each pair of counters and state machines requires 160 bits

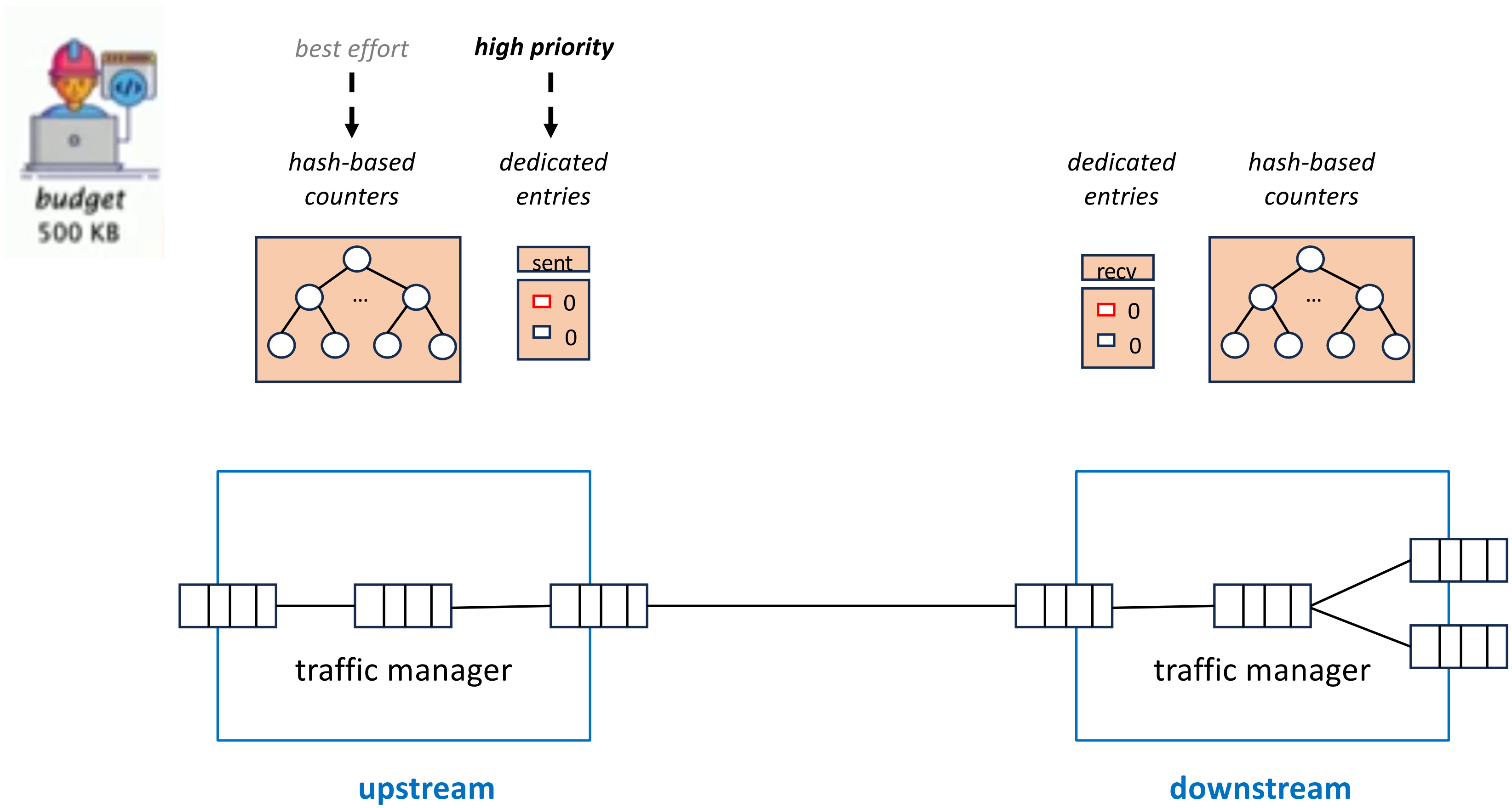
If you want to track 1M entries (i.e. all prefixes in the internet) we need:

~1.25 GB for a 64 port switch!

We can leverage the fact that *gray* failures tend to be ***sparse*** and ***aggregate*** multiple traffic entries into the ***same counter***



FANcY can combine ***dedicated counter entries*** with the ***hash-based counters***



We evaluated ***FANcY*** accuracy and speed

- Software simulations: ~9000 lines of C++ code extending ns-3
  - #1 How does FANcY perform depending on the gray failure type and the volume of traffic being affected
- Hardware implementation: ~3000 lines of P4 code
  - #2 Does FANcY work on Intel Tofino programmable switches?

#1 How does FANcY perform depending on the gray failure type and the volume of traffic being affected?

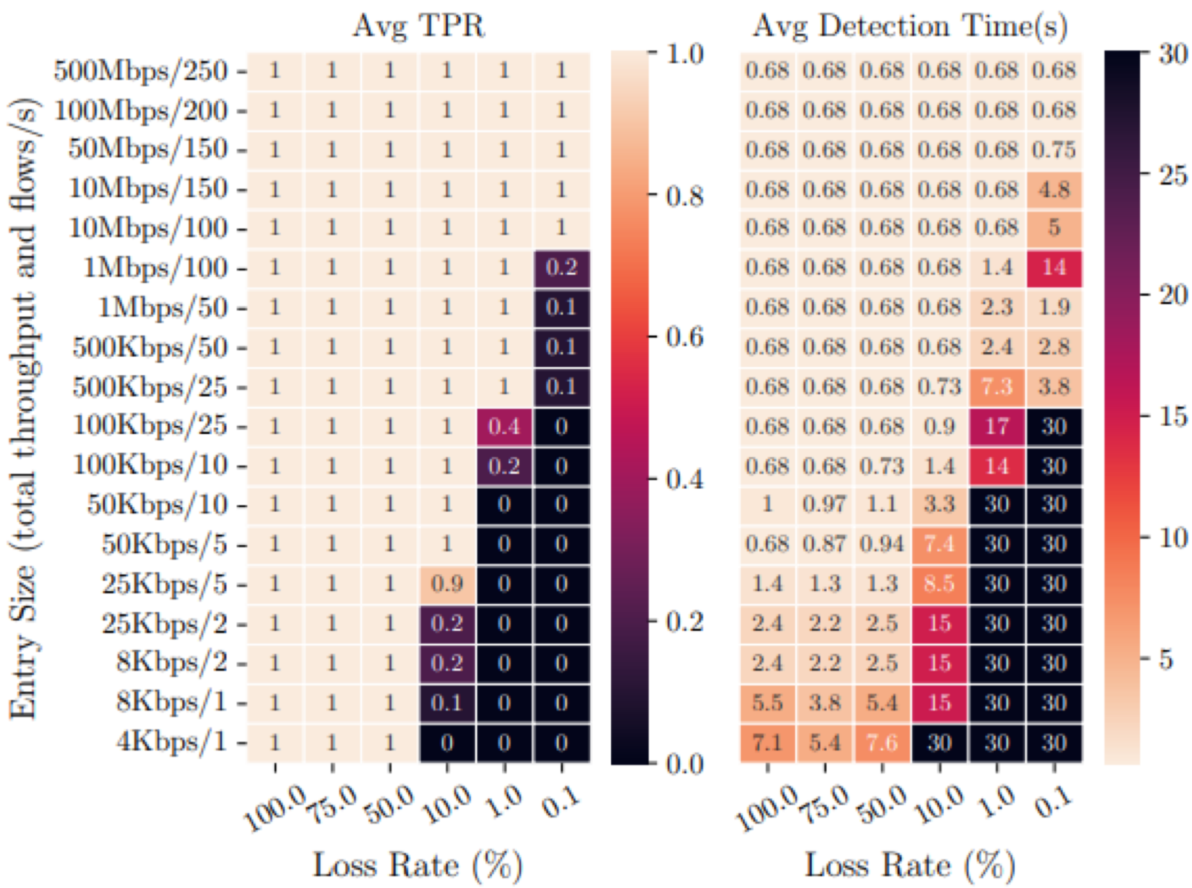
Methodology

We evaluate ***dedicated*** and ***hash-based*** counters on ***single-entry*** gray failures

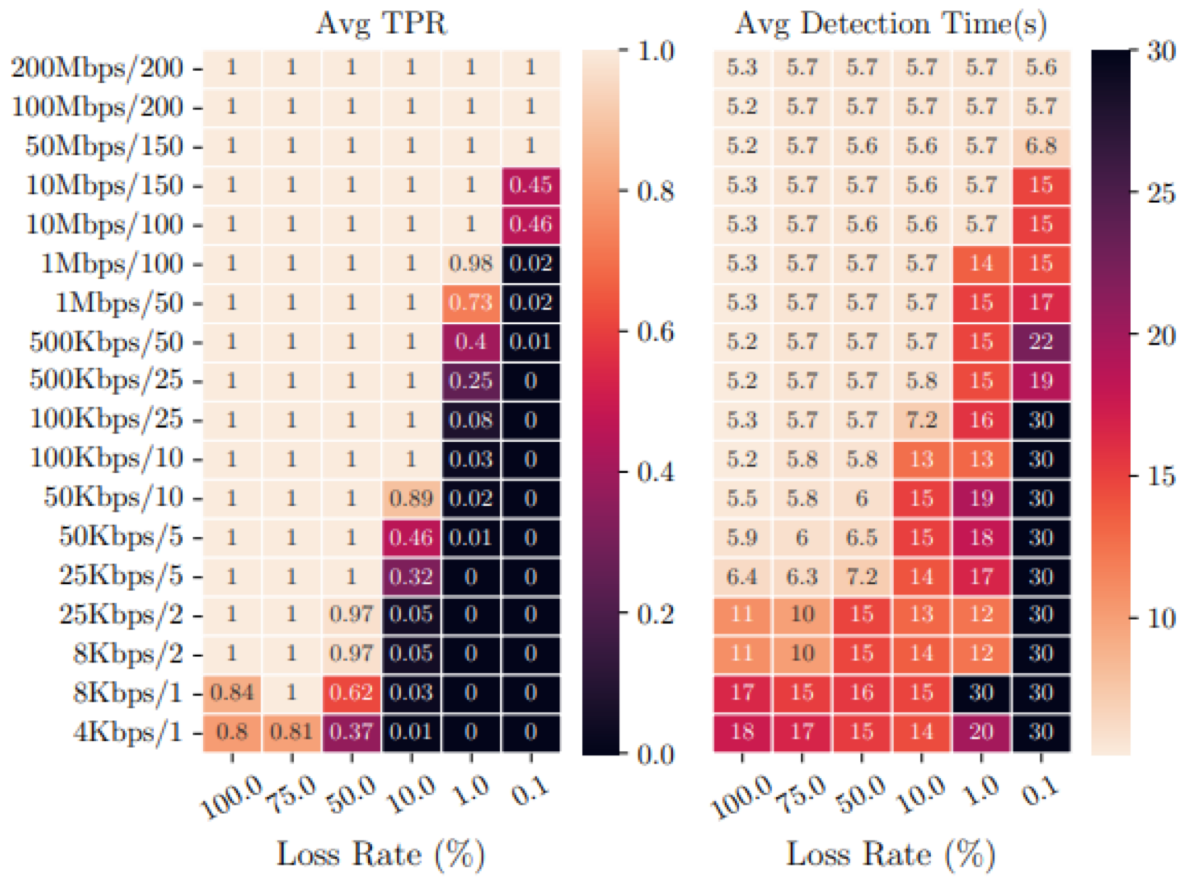
We set the inter-switch delay to ***10 ms***

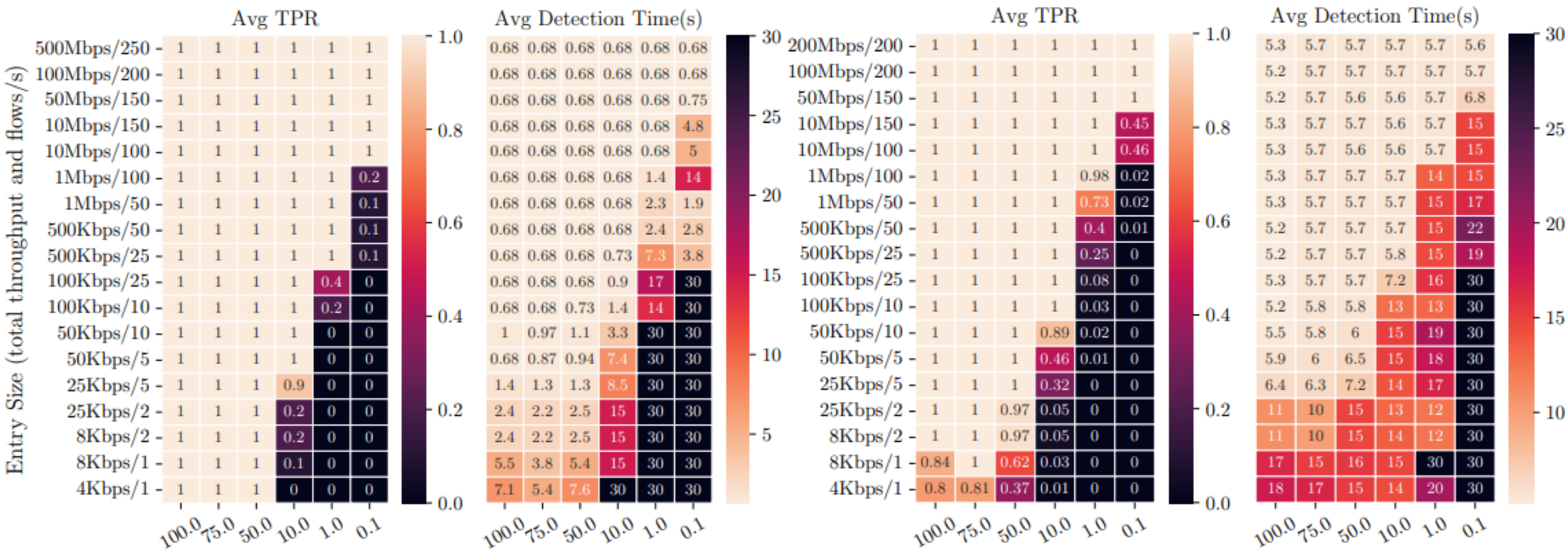
We run each experiment for ***30 seconds***

FANcY's ***hash-based counters*** performance with ***3 layers*** and a ***counting time of 200ms*** (single-entry failures)



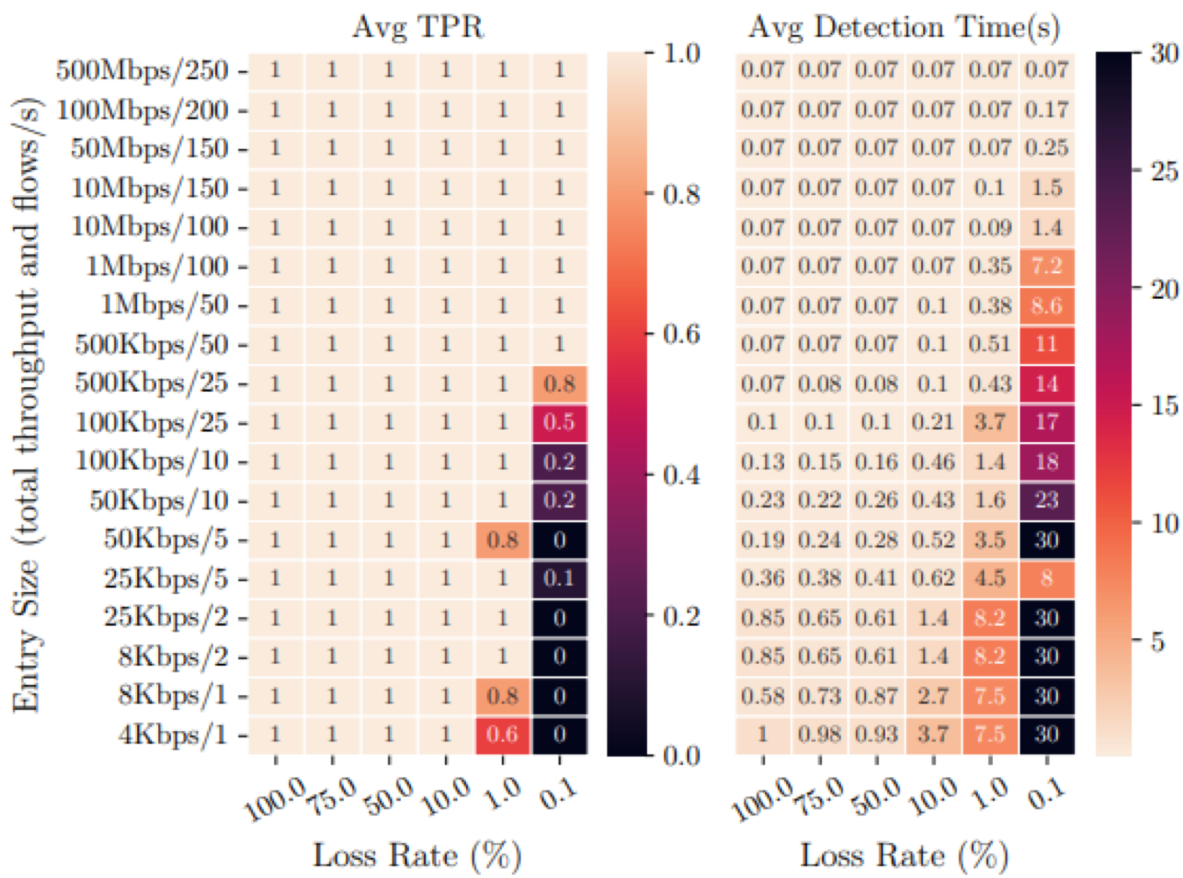
FANcY’s ***hash-based counters*** performance with ***3 layers*** and a ***counting time of 200ms*** (100-entry failures)





(a) Single-entry failures **(b) 100-entry failures**

FANcY's ***dedicated counters*** performance with different gray failures and traffic volumes



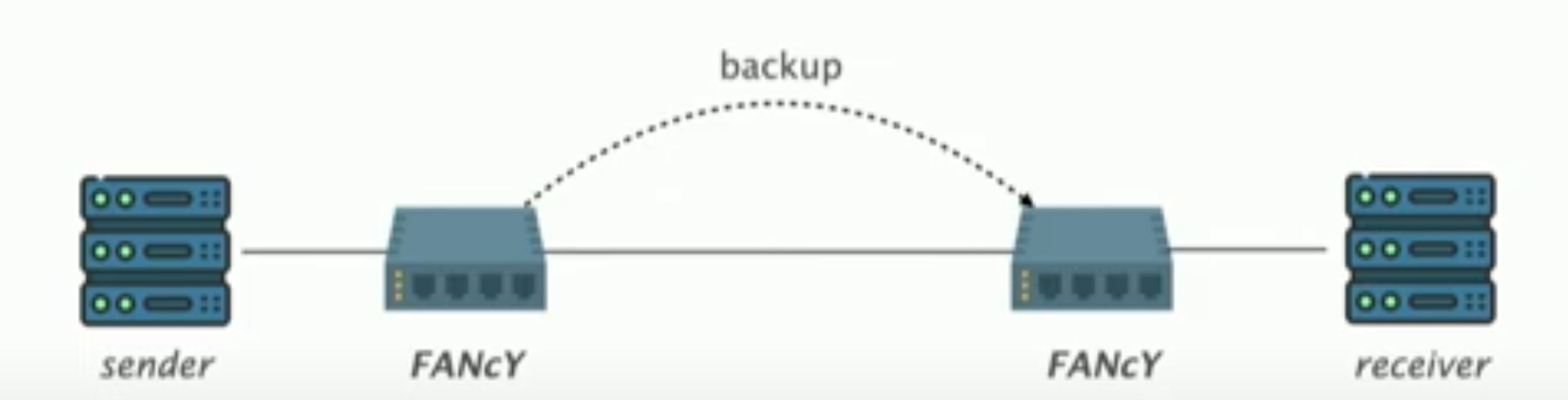
FANcY using ***CAIDA traces***

Methodology	Assigned dedicated counter to each of the 500 prefixes with the most bytes during the entire 1 hour long trace
	Randomly selected a 30 second slice
	Implement traffic generator to mimic 30 second slice
	Using slices simulated the top 10,000 prefixes, one by one, at random times
	Repeated 3 times with time of failure changing each time

FANcY accuracy and detection speed using ***CAIDA traces***

Loss Rate	TPR Bytes	TPR Prefixes			Detection time
		Total	Dedicated	Hash-Tree	
100%	91.3%	84.5%	100%	83.6%	2.03s
75%	96.0%	90.9%	100%	90.3%	2.59s
50%	98.7%	93.1%	100%	92.6%	2.65s
10%	96.5%	72.8%	100%	71%	4.96s
1%	77.5%	19.5%	98.9%	14.7%	8.91s
0.1%	56.6%	5%	86.7%	0.1%	6.29s

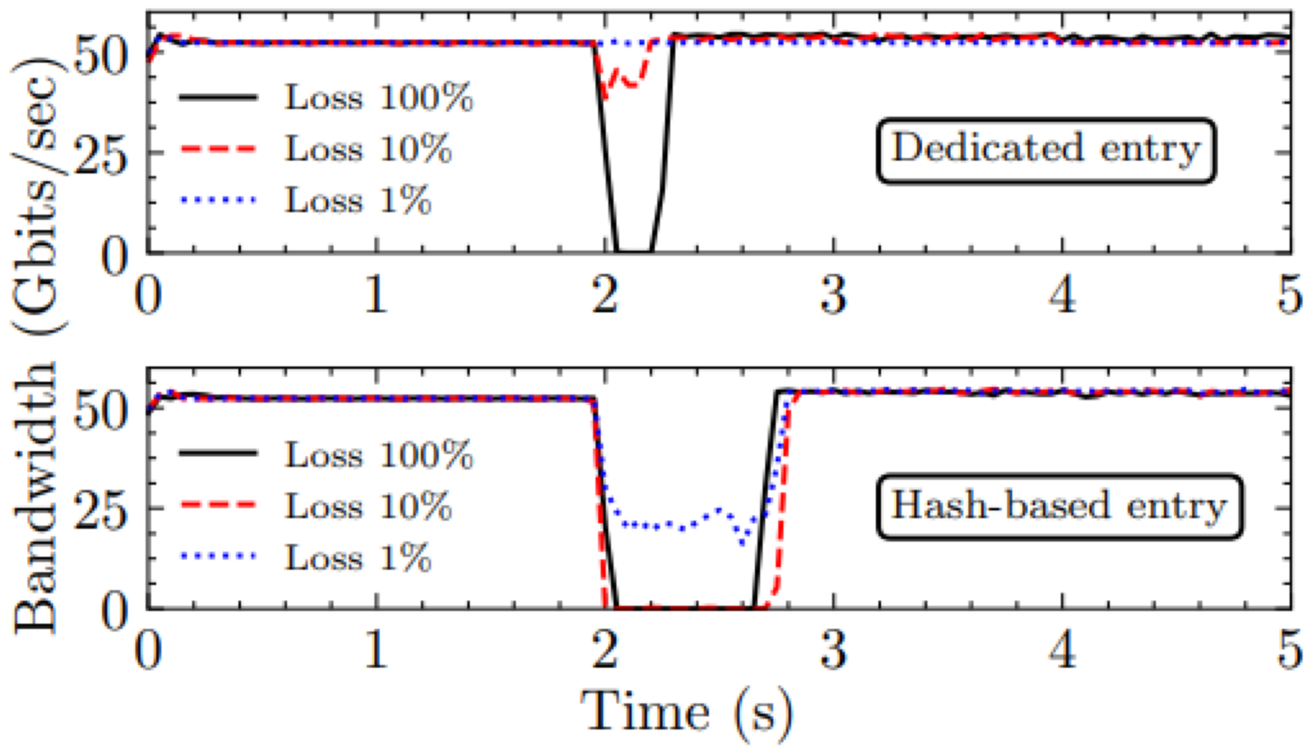
#2 Does ***FANcY*** work on ***Intel Tofino*** programmable switches?



Dedicated counters are exchanged every ***200 ms***

Hash-based counters have a depth of ***3*** and are zoomed every ***200 ms***

Dedicated counters can detect ***gray*** failures after the first counting session whereas ***hash-based counters*** need to zoom three times



Resource usage of ***FANcY*** using switch.p4 as a baseline

Resource	Dedicated Counters	Full FANcY	FANcY + Rerouting	switch.p4
SRAM	4.80%	6.65%	8.1%	29.58%
Statefu ALU	16.66%	27.08%	33.33%	14.58%
VLIW Actions	9.4%	14.1%	15.6%	36.72%
TCAM	1.4%	2.1%	2.1%	32.29%
Hash bits	5.8%	11.8%	13.1%	34.74%
Ternary Xbar	1.8%	3.10%	3.10%	43.18%
Exact Xbar	5.1%	10.8%	12.3%	29.36%

FANcY: Fast In-Network ***Gray*** Failure Detection for ISPs

detects gray failures by doing counter comparisons
reliable counter synch protocol directly in data plane

scales by using two types of counting data structures
uses dedicated counters and hash-based counters

runs on today's hardware
implemented and tested on Intel Tofino Switches