Improving Network Failure Detection and Recovery with Programmable Data Planes

Edgar Costa Molero

PhD Defense

May, 6 2024

ETH zürich

Since its creation in 1969,

the Internet has seen a tremendous growth

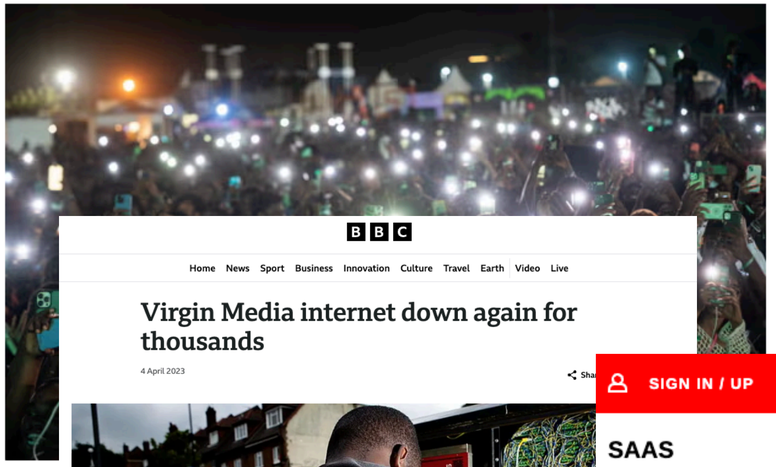
	ARPANET (1969)	Internet (2024)
size	4 nodes	80 000 networks (5.5 billion users)
traffic	56 kbps	+100s Tbps (100-800 Gbps links)
use case	remote access	information access entertainment shopping work …

Given its current ***scale*** and ***complexity***

the ***Internet*** is prone to ***all sorts*** of failures

Much of west and central Africa without internet after undersea cable failures

Ivory Coast, Liberia, Benin, Ghana and Burkina Faso among countries experiencing outages



Your
Decem
Ernest

BBC

Home News Sport Business Innovation Culture Travel Earth Video Live

Virgin Media internet down again for thousands

4 April 2023

Network problems causing ever more outages

Andy Lawrence, Uptime Institute

Andy Lawrence is founding member and Executive Director of Research at Uptime Institute

in

Power problems are no longer the top cause of outages

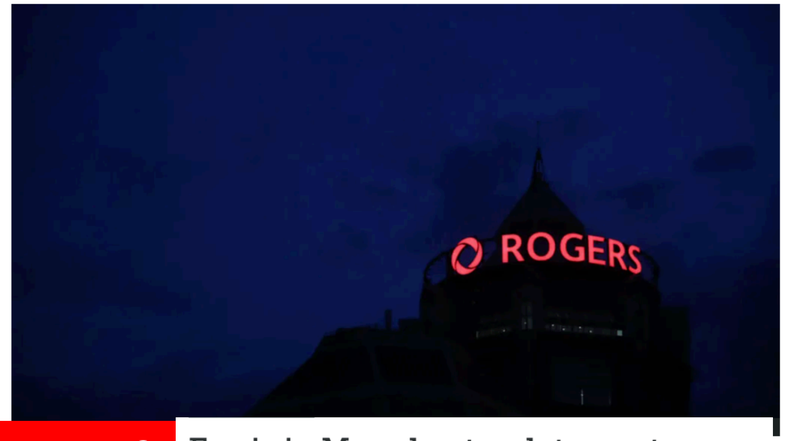
February 19, 2021 Have your say

Rogers outage: Why a network upgrade pushed millions in Canada offline

20 July 2022

Share

By Holly Honderich, BBC News



SIGN IN / UP The Register

SAAS

This article is more than **1 year old**

Slack fingers AWS auto-scaling failure in January outage postmortem

Slack says it has identified a scaling failure in its AWS Transit Gateways (TGWs) as the reason for the chat service's monumental outage on 4 January. As a result, Amazon's cloud computing arm said it is "reviewing the TGW scaling algorithms".

The situation got worse. There was "widespread packet loss" in Slack's network leading to "saturation of our web tier" and Slack became completely unavailable. Automated systems intended to maintain health instead made the problem worse.

Equinix Manchester data center experiences brief outage due to 'hardware failure'

Causing ISPs to briefly drop off

May 25, 2022 By: Georgia Butler Have your say

The MA1 Equinix data center in Manchester experienced a major outage this week, causing ISPs and other customers to briefly drop off.

Equinix confirmed that the Internet Exchange, Metro Connect, Equinix Fabric, and Equinix Connect services running out of MA1 were affected.

Hard failures

Port failure

Link failure

Device failure

Software bugs and misconfigurations

impact all traffic
(i.e., 100% packets for all entries)

Gray failures

TCAM bit flips and memory corruption

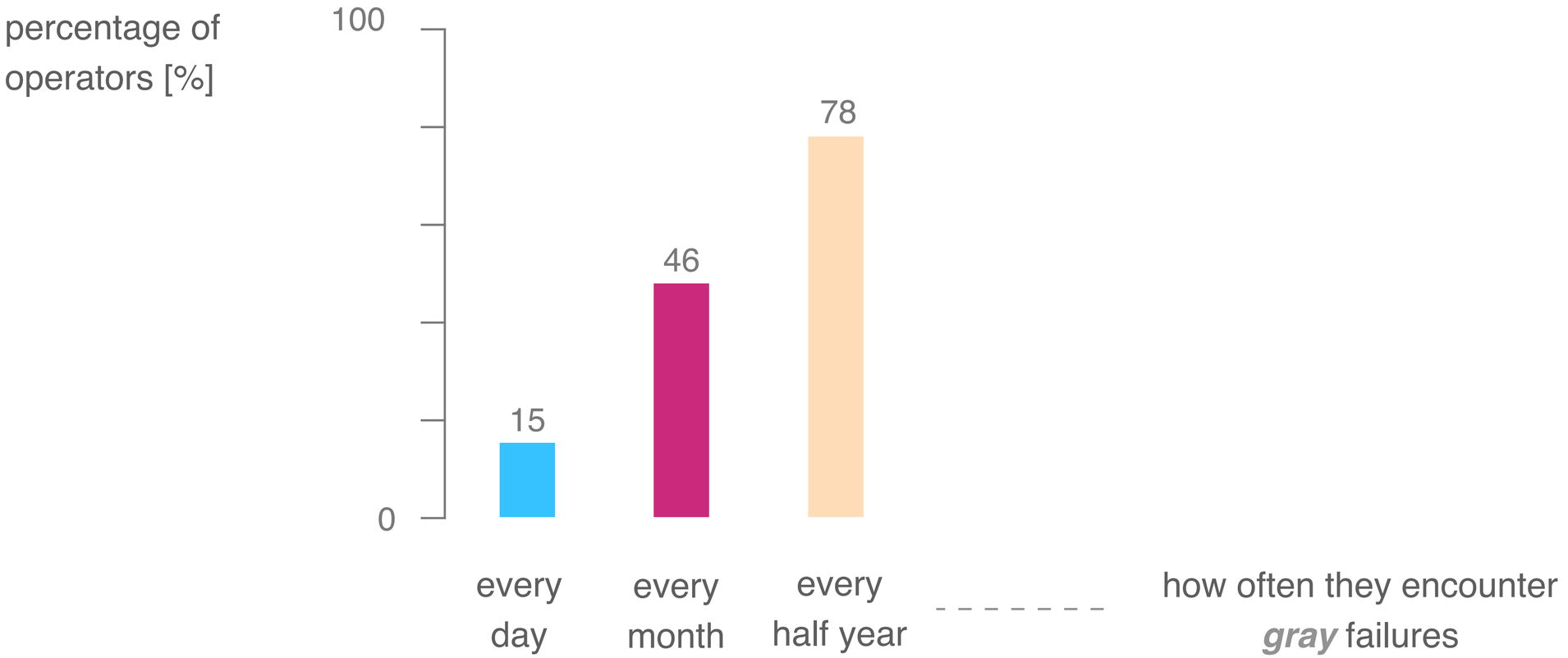
Bent fibers and not well seated line-cards

CRC checksum errors

Software bugs and misconfigurations

impact a subset of the
traffic (i.e., <1% or specific protocol)

Gray failures are a problem for a ***majority*** of ***operators***



Gray failures are ***hard*** to detect

and ***traditional mechanisms*** *are not enough*

Hard to detect	Require analyzing ***all*** the traffic, ***all*** the time *Hard to scale*
Traditional mechanisms	Typically analyze a ***subset*** of the traffic *Sampling, probing, or specific counters*

1 detecting network failures

Recovery involves

three extra steps

2 notifying remote devices of changes

3 computing new paths

4 updating the forwarding state

Except for notifying,

each of these tasks can take ***minutes*** to complete

1	detecting	network failures
3	computing	new paths
4	updating	the forwarding state

Except for notifying,

each of these tasks can take ***minutes*** to complete

1	detecting	***gray*** network failures
3	computing	new paths
4	updating	the forwarding state

Except from notifying,

each of these tasks can take ***minutes*** to complete

3 computing new paths for ***100,000s destinations***

4 updating the forwarding state

Except from notifying,

each of these tasks can take ***minutes*** to complete

4 updating the forwarding state for ***100,000s entries***

In this thesis…

FANcY
[SIGCOMM'22]

Local
gray and hard
failure detection

Blink
[NSDI'19]

Remote
failure detection

Hw-accelerated
control planes
[HotNets'18]

Hardware-based
control plane
implementation

FANcY
[SIGCOMM'22]

Local
gray and hard
failure detection

Blink
[NSDI'19]

Remote
failure detection

Hw-accelerated
control planes
[HotNets'18]

Hardware-based
control plane
implementation

FANcY
[SIGCOMM'22]

Local
gray and hard
failure detection

Hw-accelerated
control planes
[HotNets'18]

Hardware-based
control plane
implementation

FANcY
[SIGCOMM'22]

Blink
[NSDI'19]

Hw-accelerated control planes
[HotNets'18]

Local
gray and hard failure detection

Remote
failure detection

Hardware-based
control plane implementation

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

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

2 ***to compare*** the statistics

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

Most ***data center*** ***gray*** failure detection solutions

do ***collect statistics*** on all traffic and ***compare*** them.

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

Why?

The characteristics of ***ISP networks*** make

data center failure detection systems ***not operational***

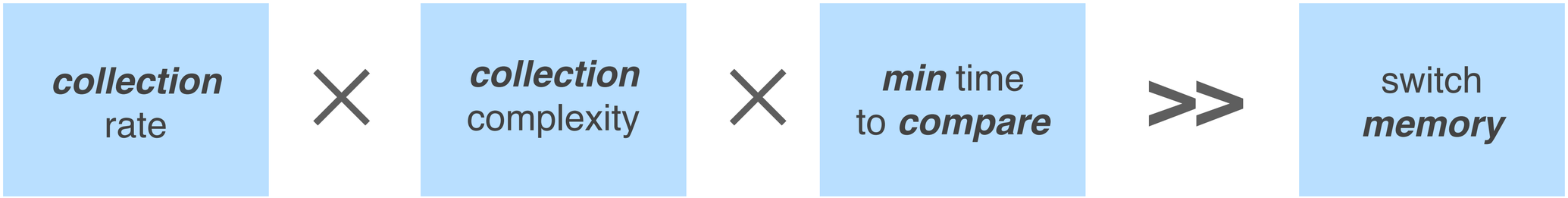
- No end-point control
  only control network devices

- High link bandwidth
  100 Gbps and increasing

- 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



link bandwidth

memory required per packet

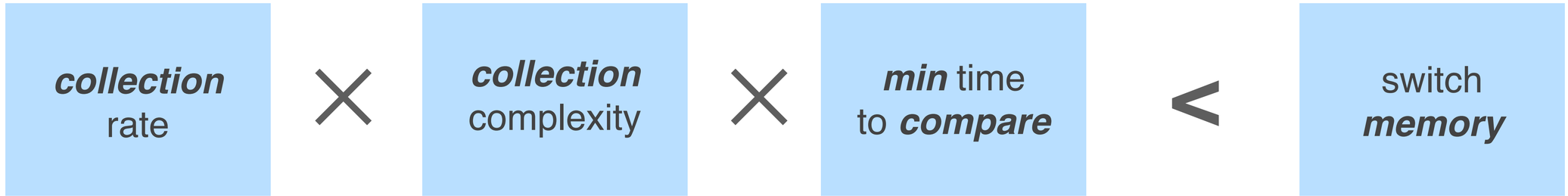
link delay

control plane reading speed

fixed

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

required memory to operate



link bandwidth

memory required per packet

link delay

control plane reading speed

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

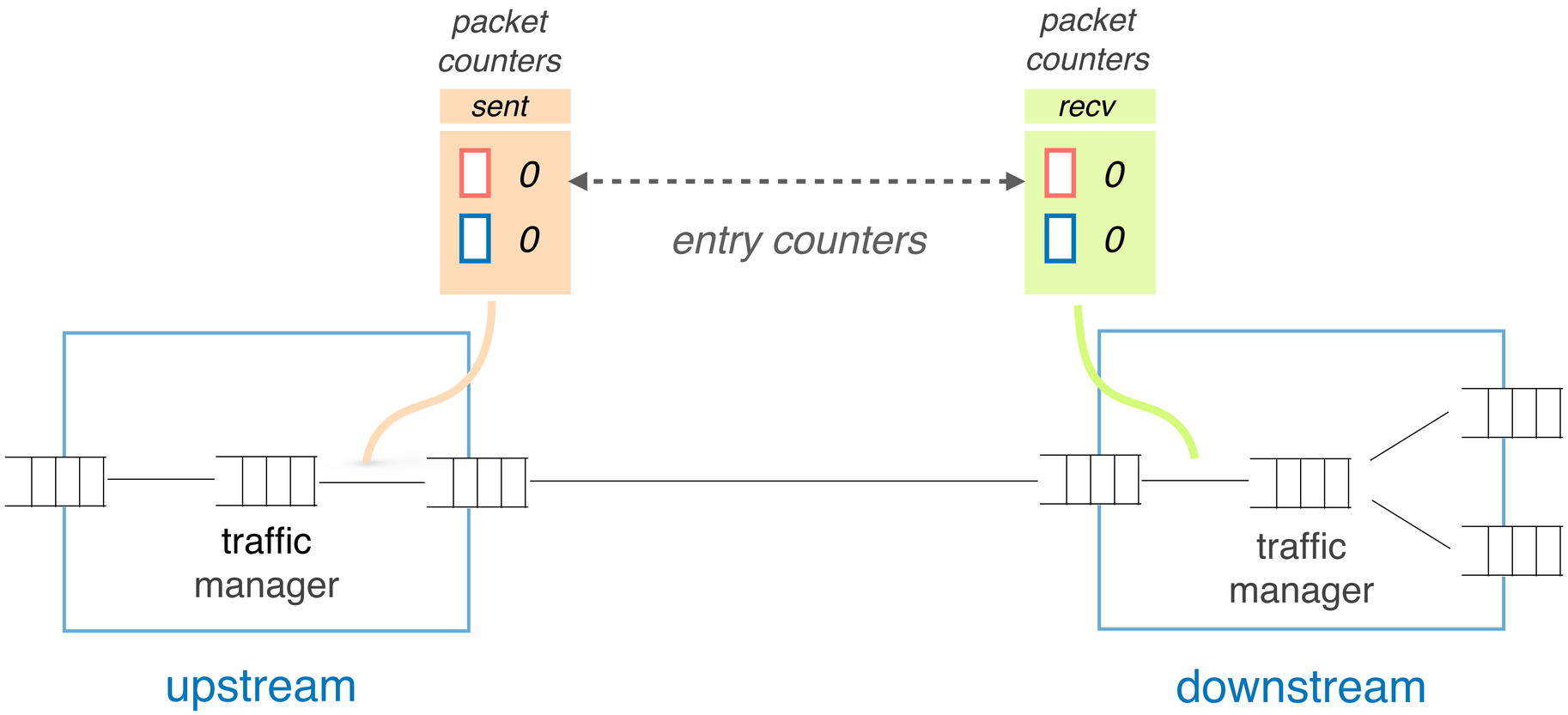
#2 FANcY compares the collected statistics directly in the data plane

FANcY works in switch ***pairs***

and detects ***faulty entries*** at the ***port level***

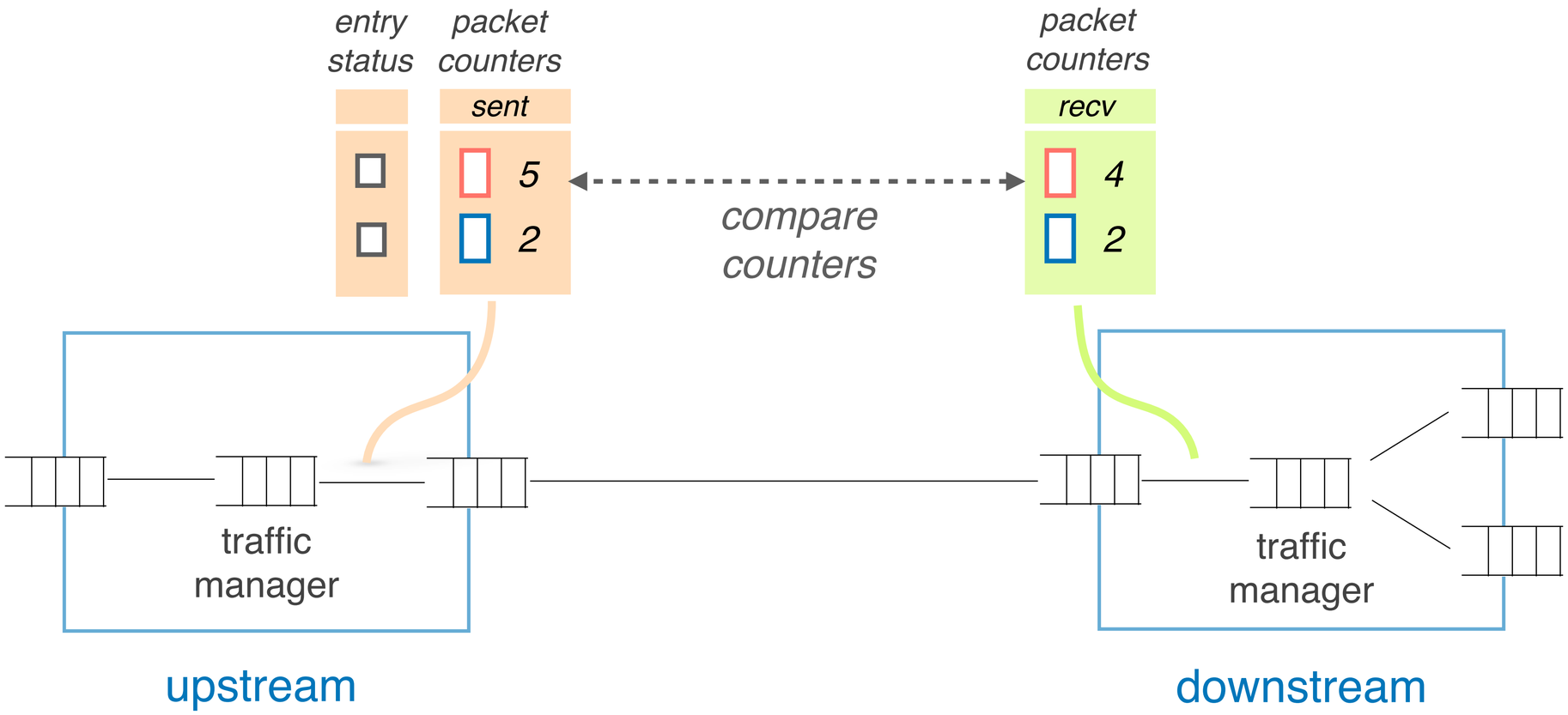
For each ***traffic entry*** the ***upstream*** and ***downstream***

switches use a packet ***counter*** to collect statistics



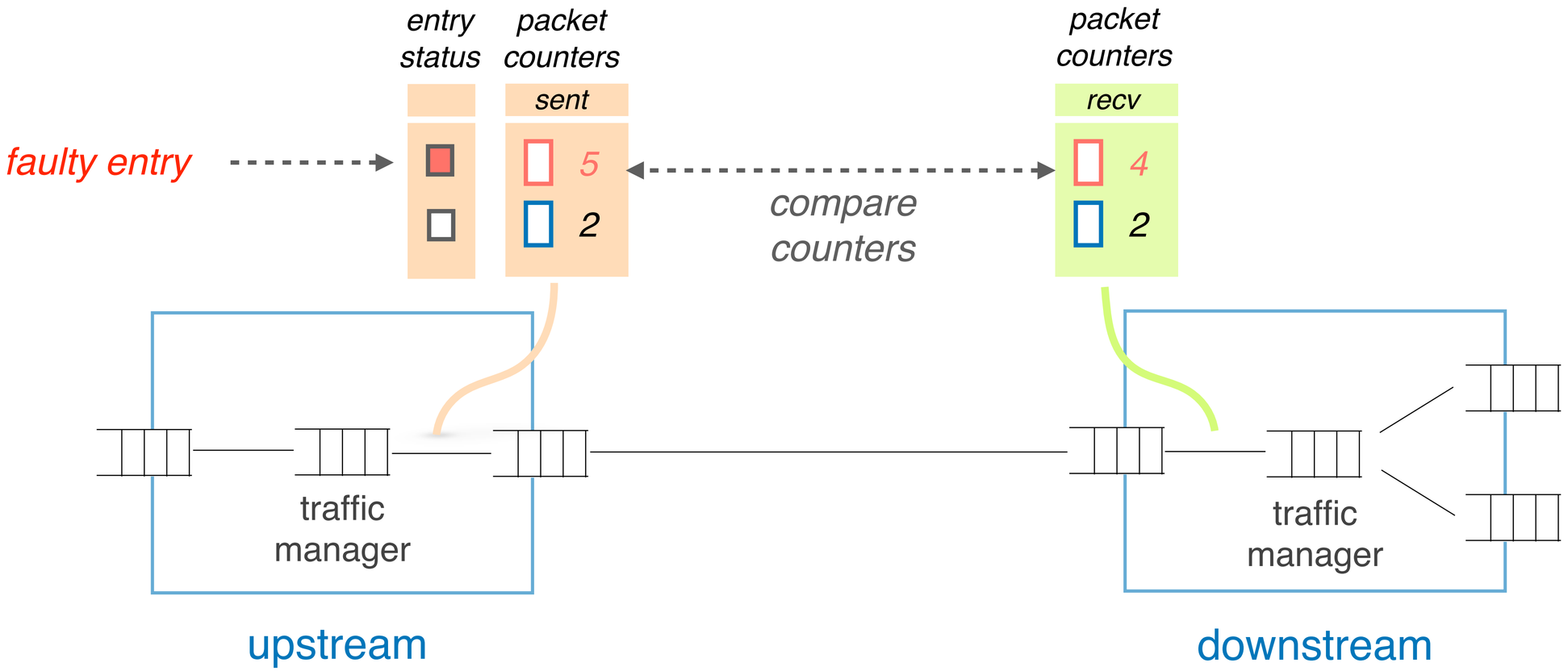
After collecting statistics, the **upstream** and the ***downstream***

compare its counters in order to find ***discrepancies***



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

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

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

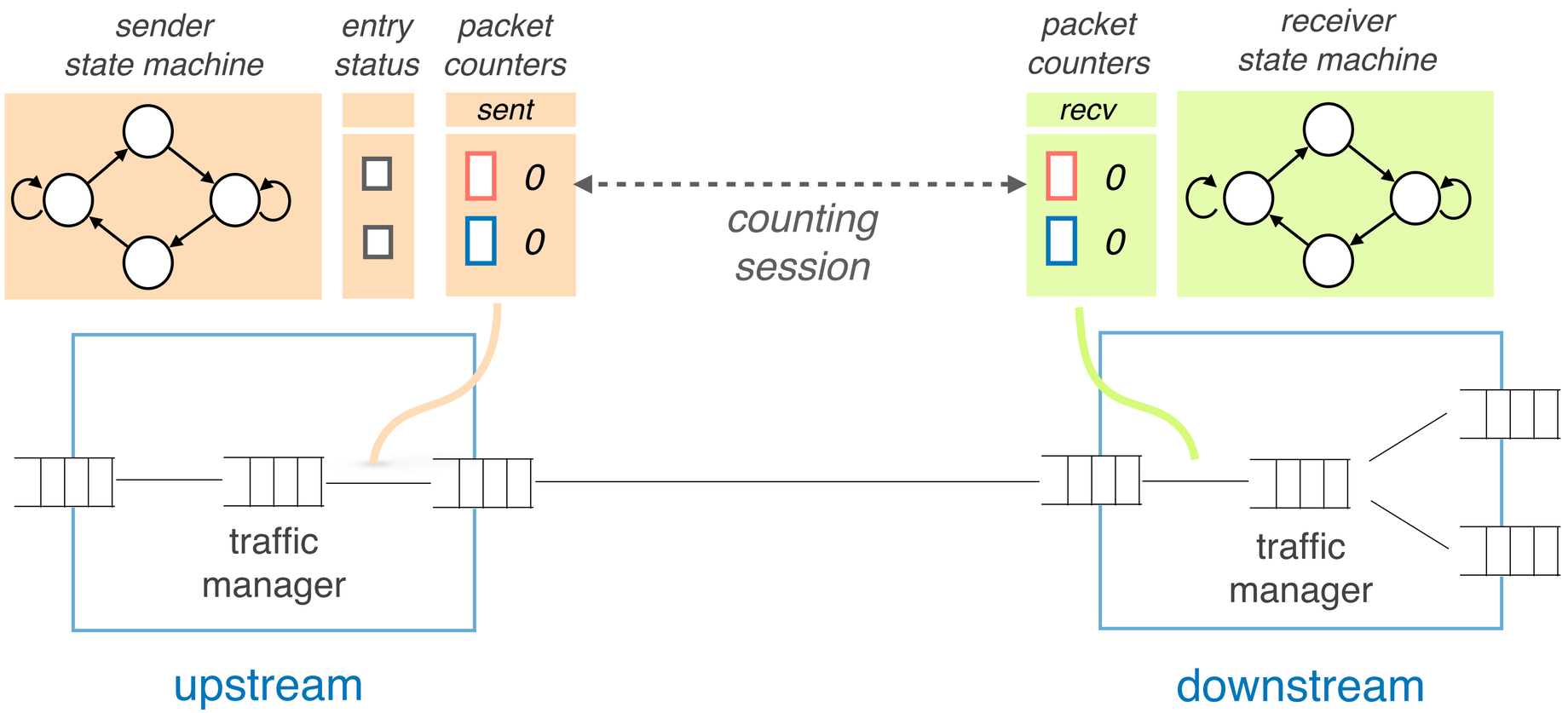
FANcY establishes counting sessions for each counter pair

#2 Scaling *to many traffic entries*

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

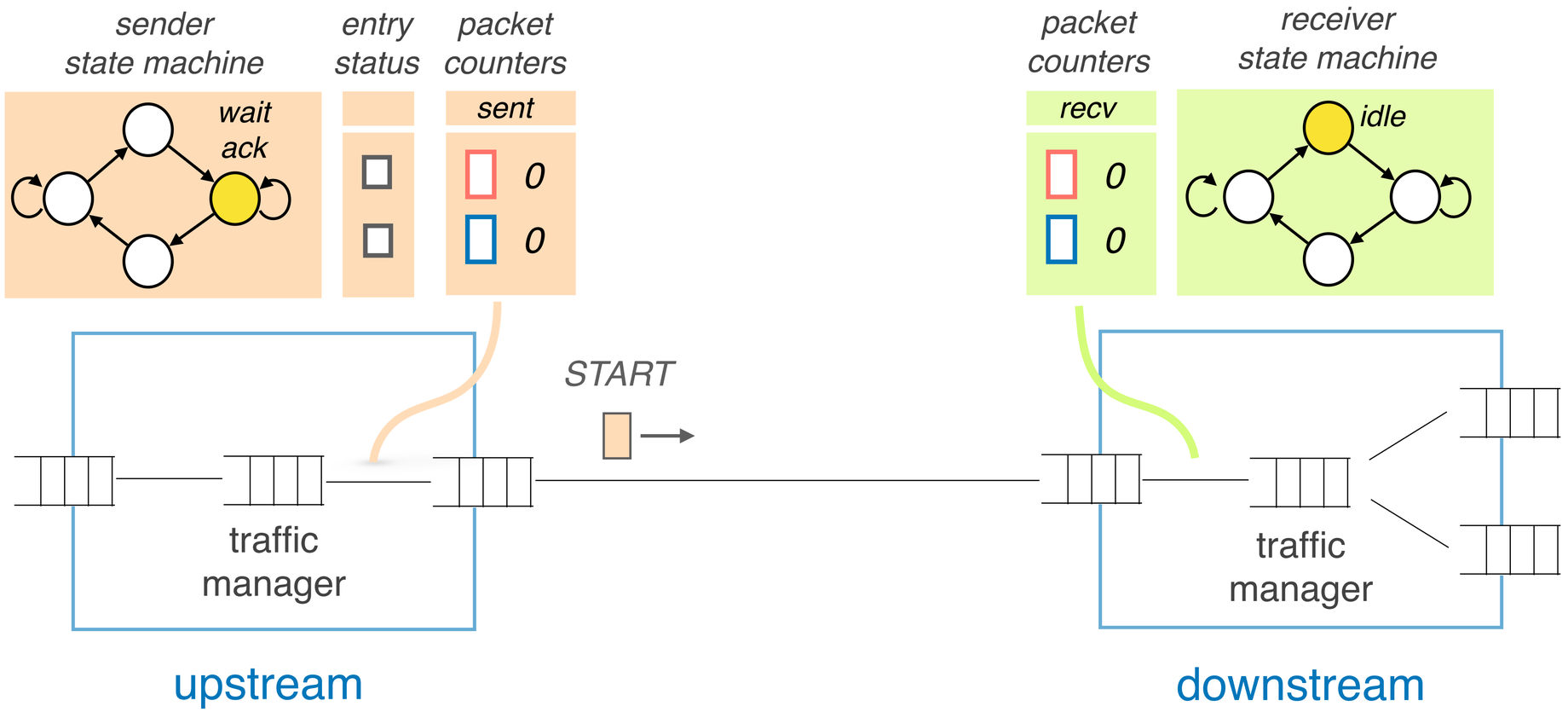
To achieve perfect **synchronization** and **reliability**

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



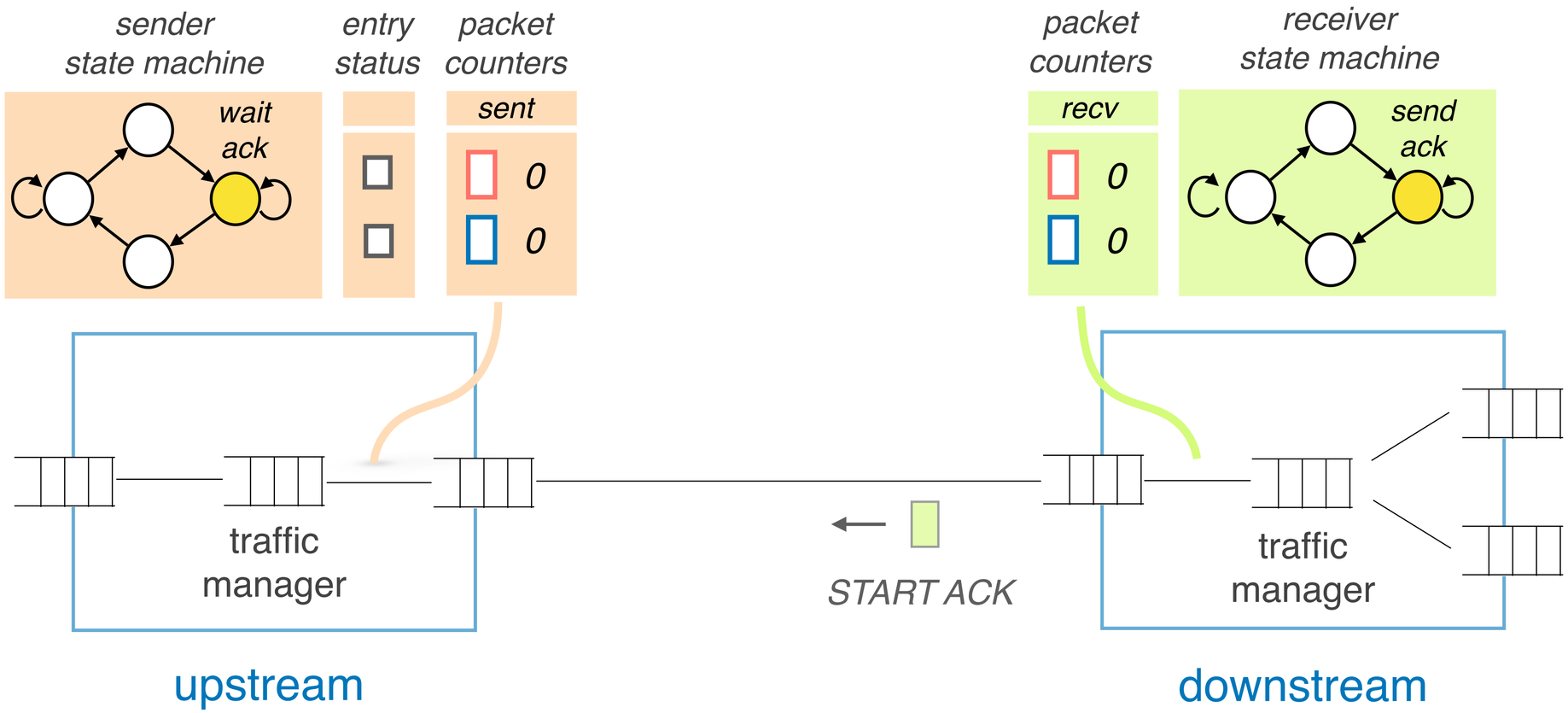
To achieve perfect **synchronization** and **reliability**

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



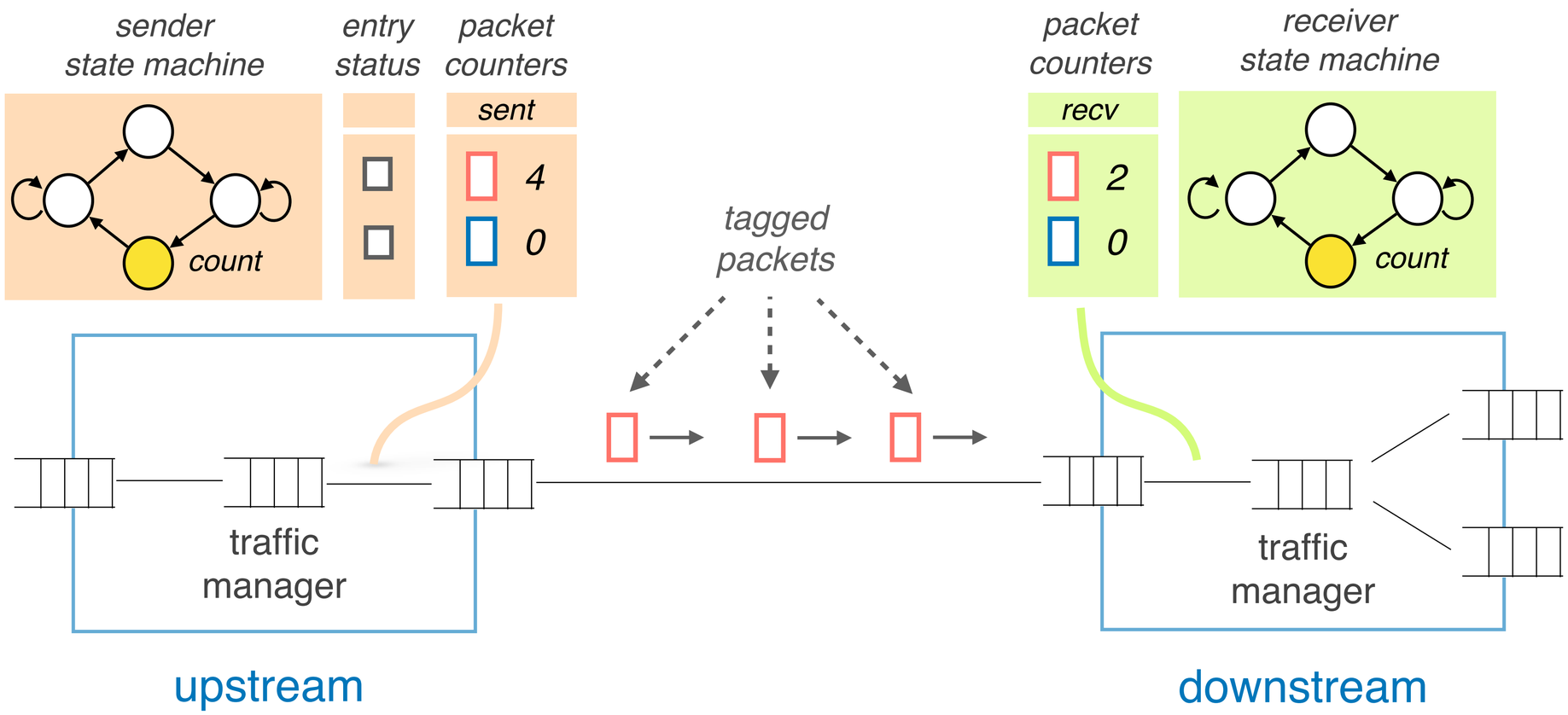
To achieve perfect **synchronization** and **reliability**

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



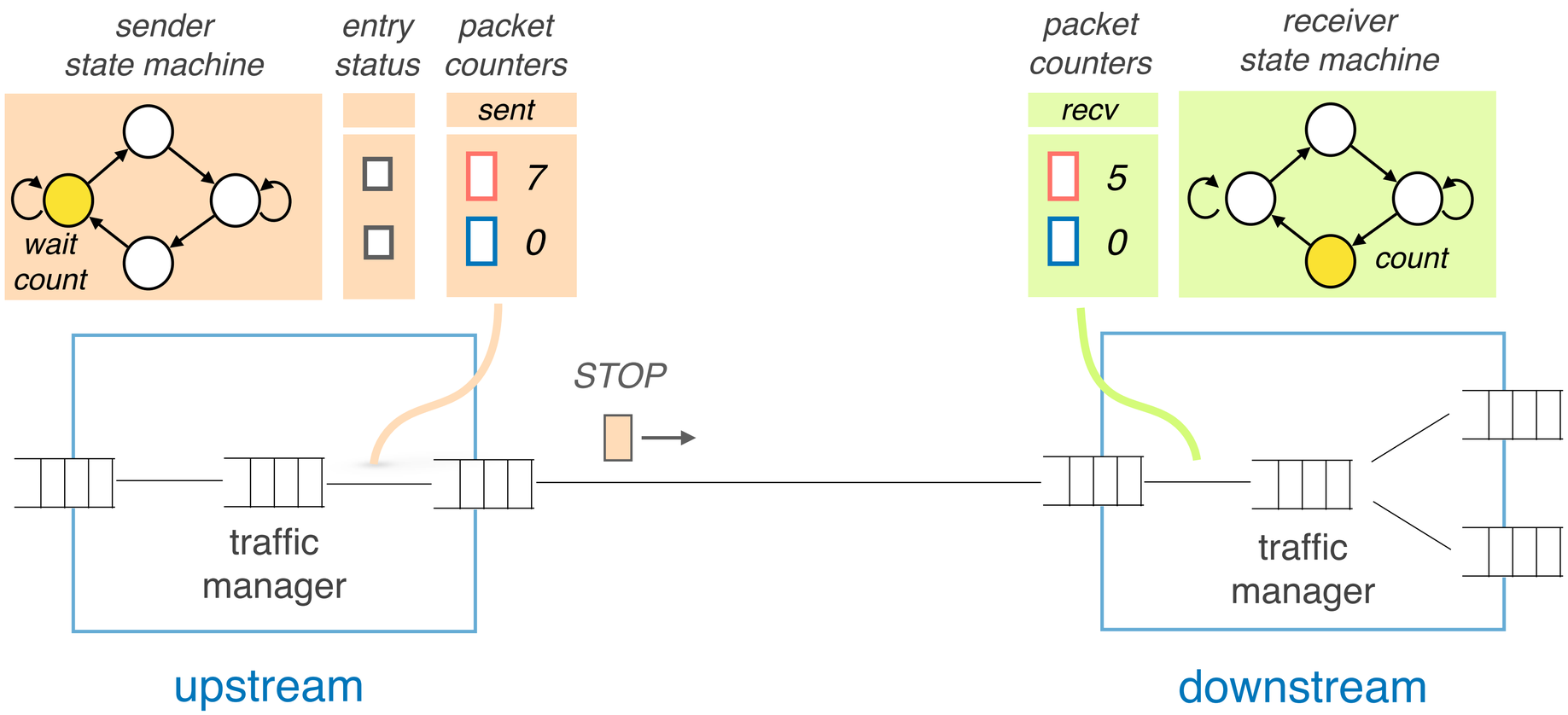
To achieve perfect **synchronization** and **reliability**

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



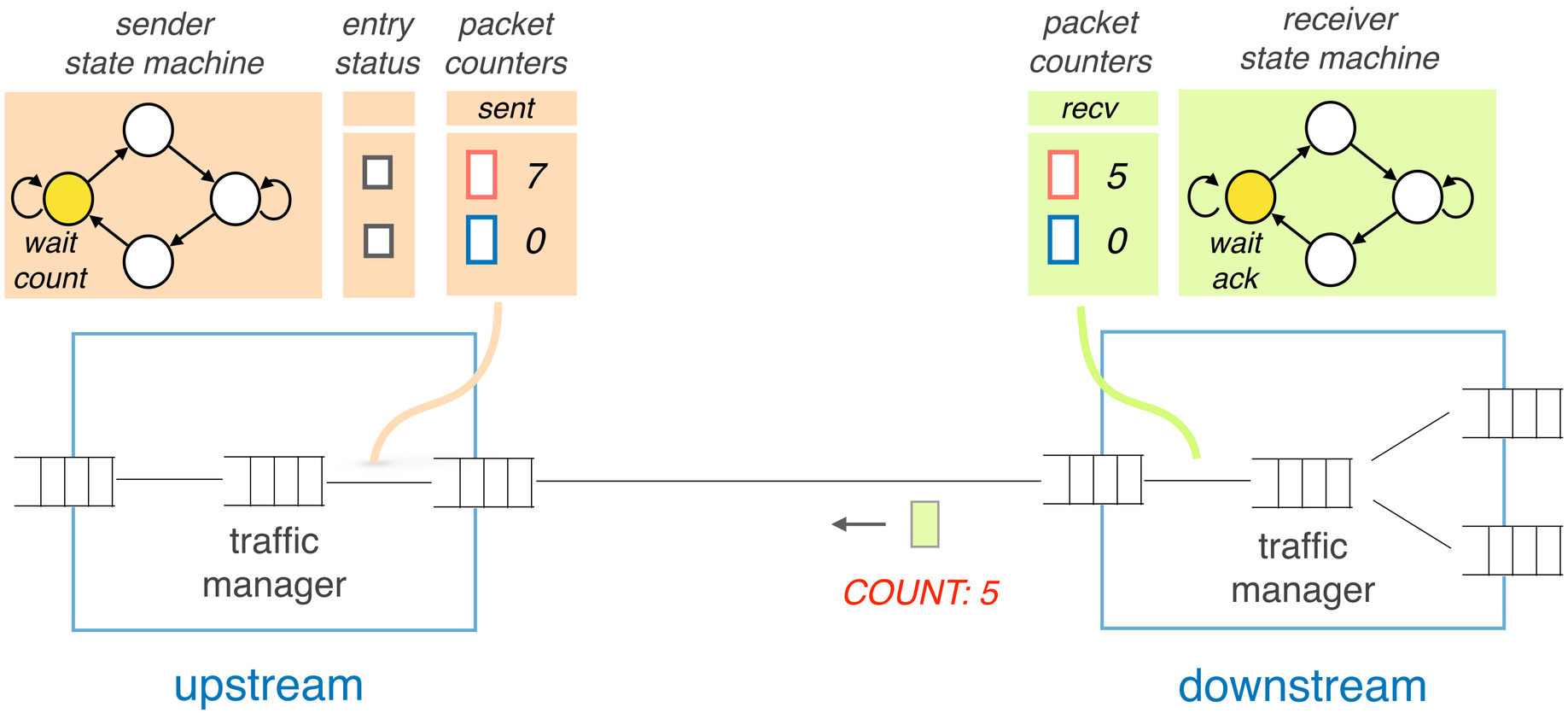
To achieve perfect **synchronization** and **reliability**

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



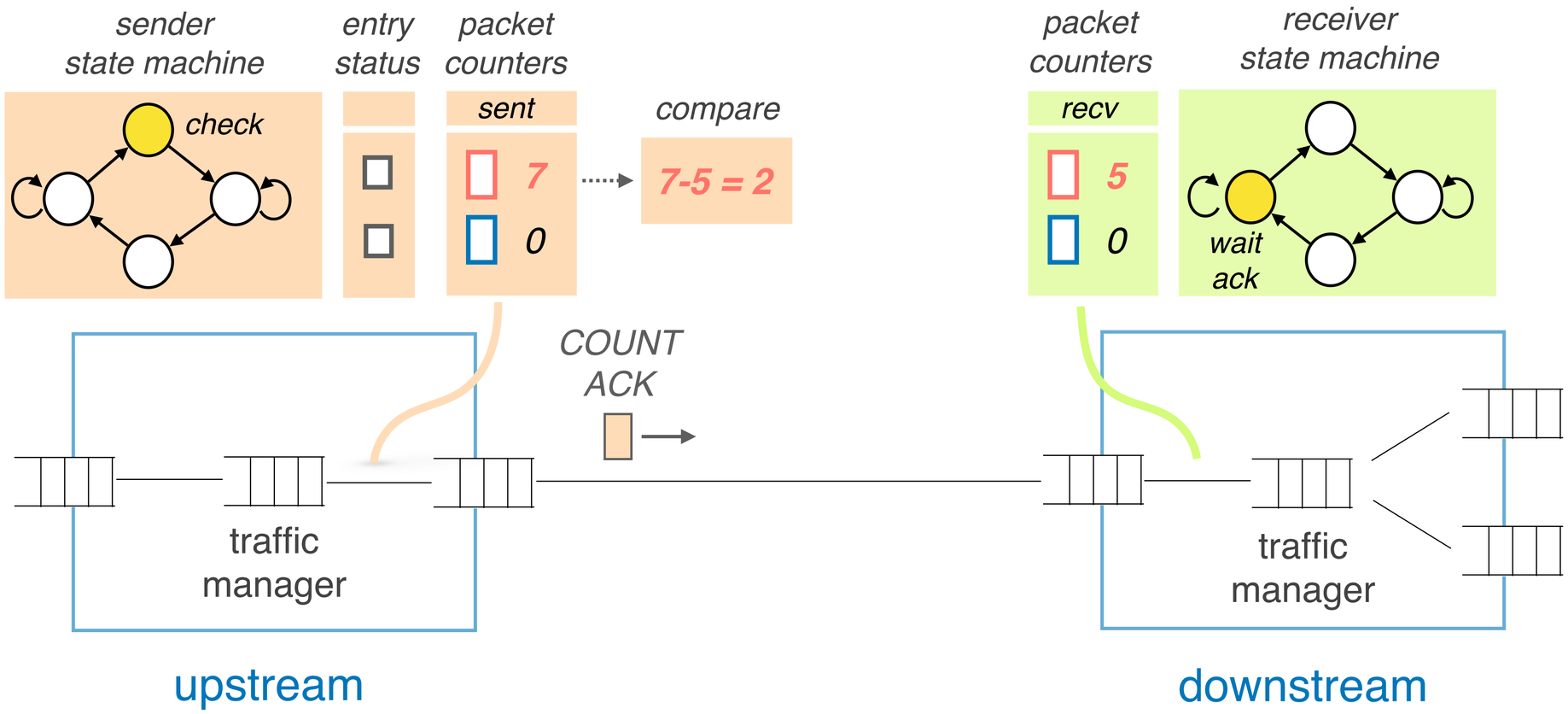
To achieve perfect **synchronization** and **reliability**

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



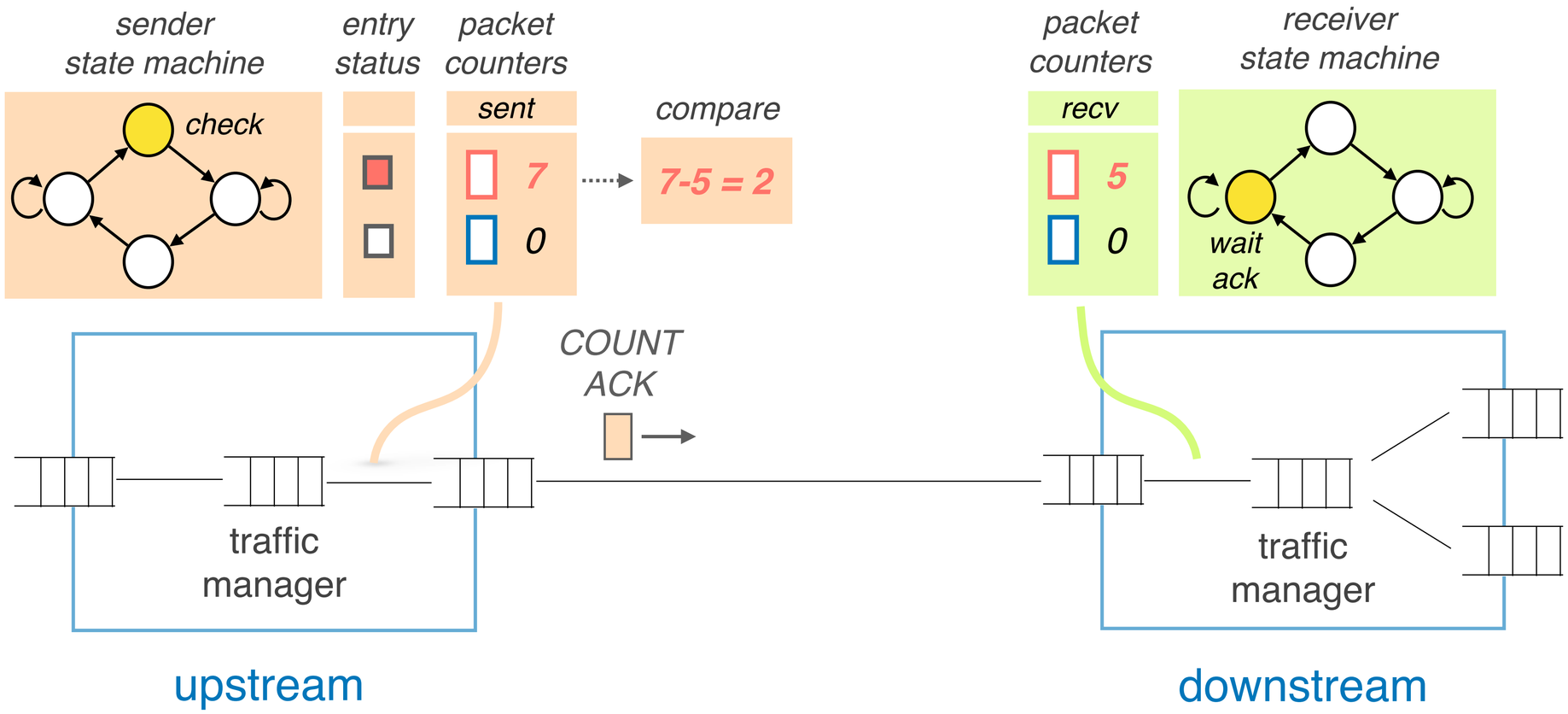
The upstream checks if there is

any ***discrepancy*** between counters



The upstream checks if there is any ***discrepancy***

between counters if so, it ***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

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

~1.25 GB for a 64 port switch!

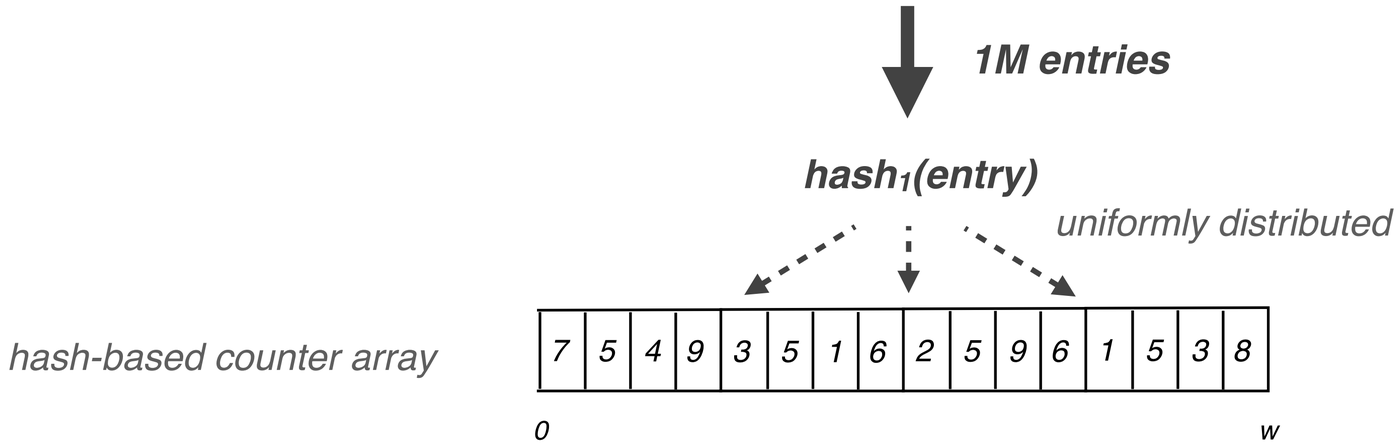
x20 today's
switches memory!

We can leverage the fact that ***gray*** failures tend to be ***sparse***

and ***aggregate*** multiple traffic entries into the ***same counter***

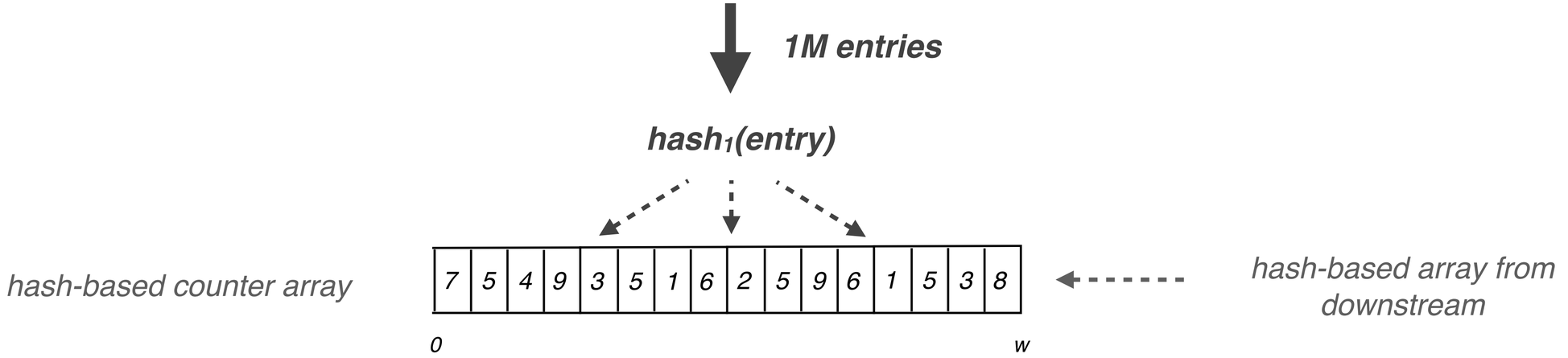
Instead of using one counter per entry,

we use a ***hash-based counter array*** for ***all*** the entries

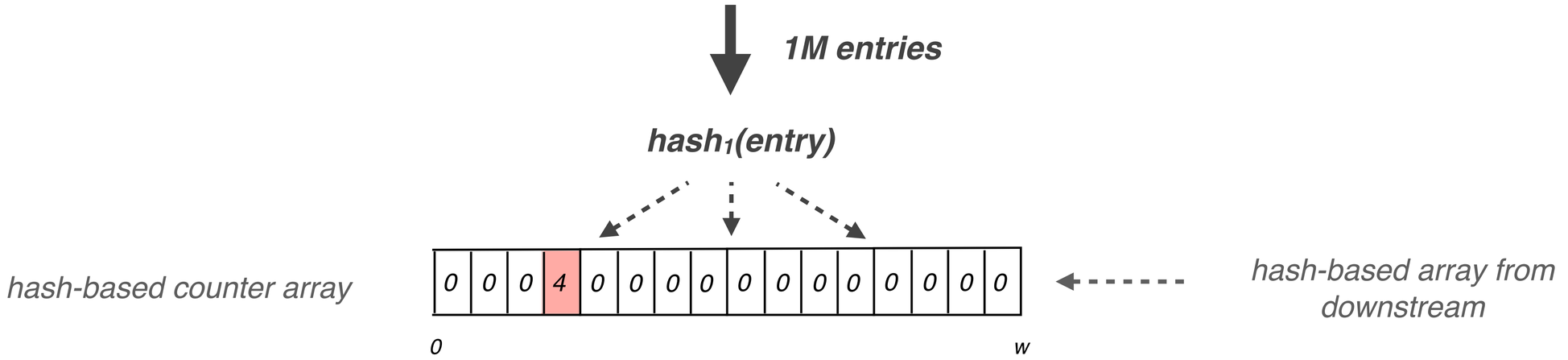


As with regular counters, at the end of a counting session,

the downstream sends the array of counters…

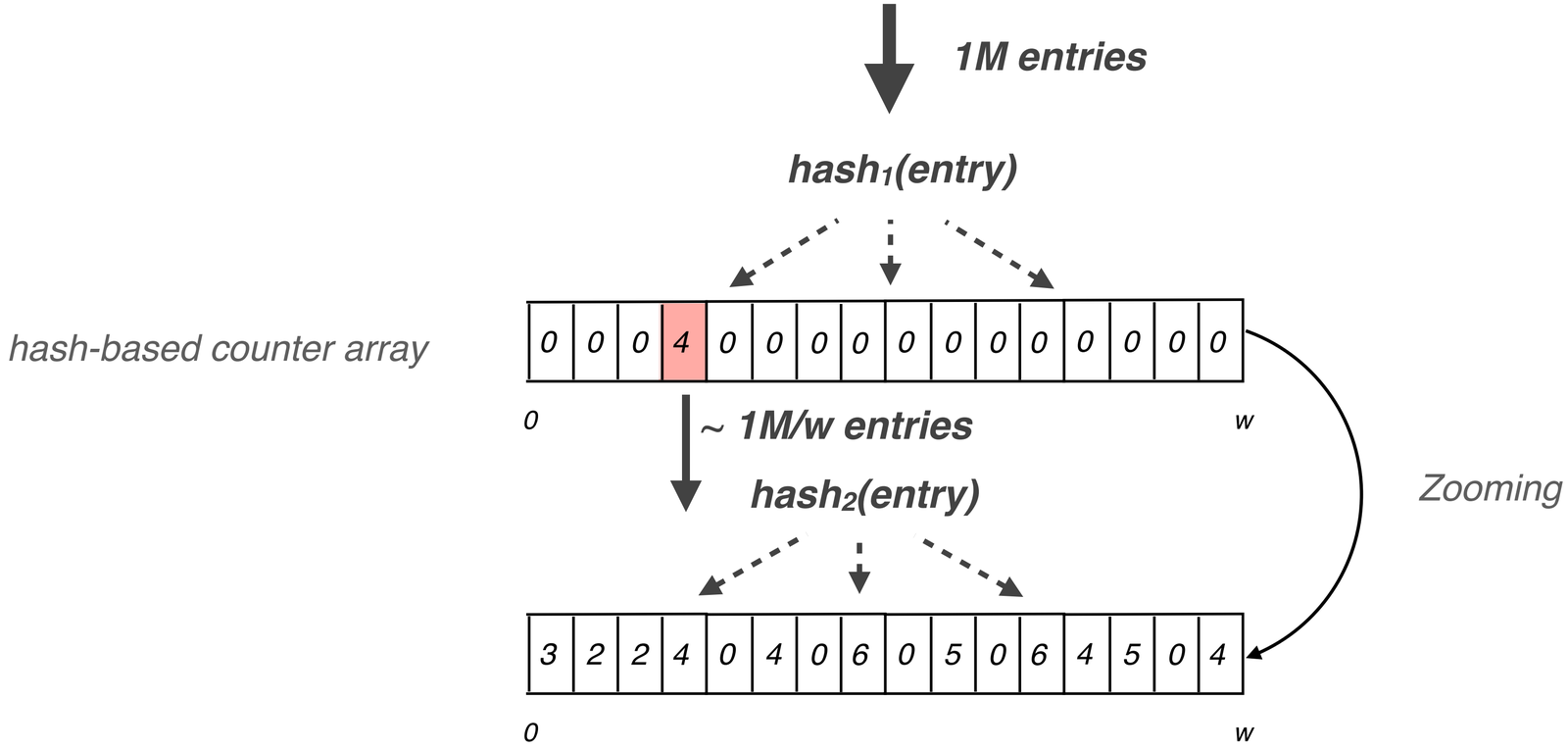


… and the upstream computes the difference



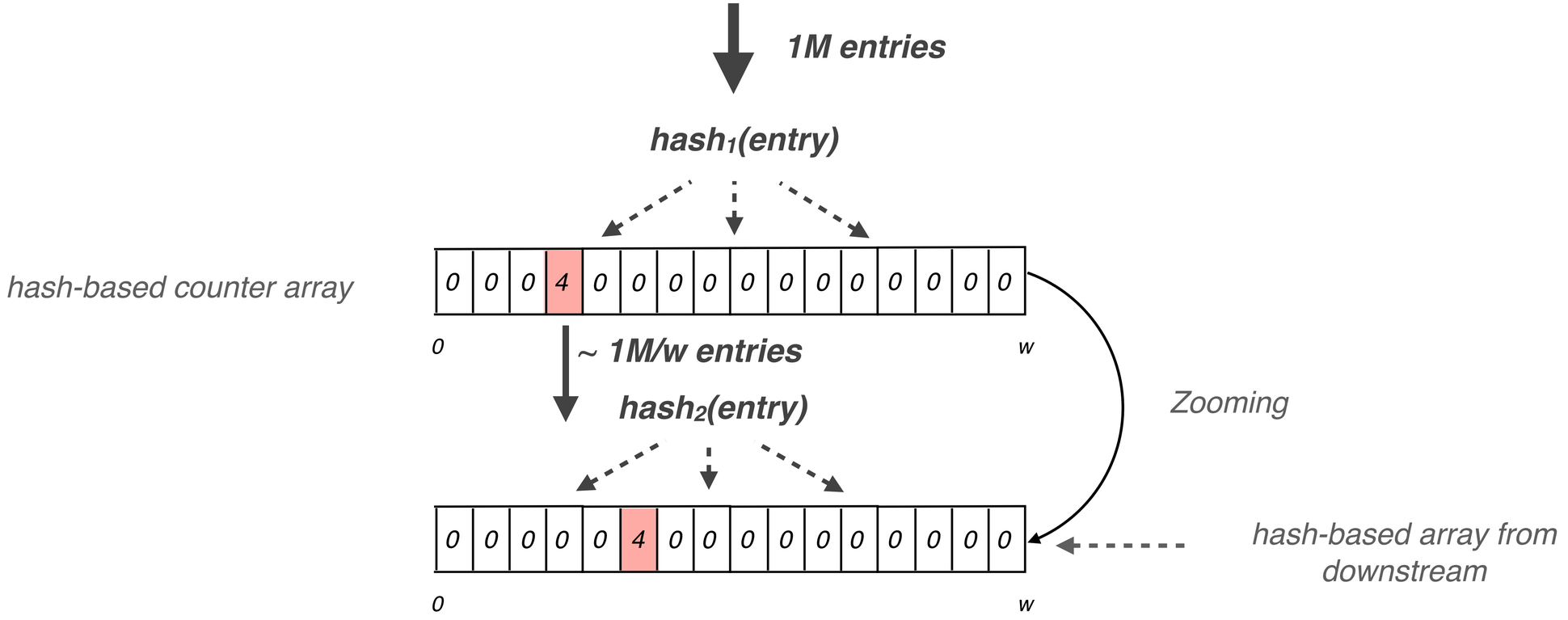
The process is repeated but only

for the traffic entries hashed in the faulty cell



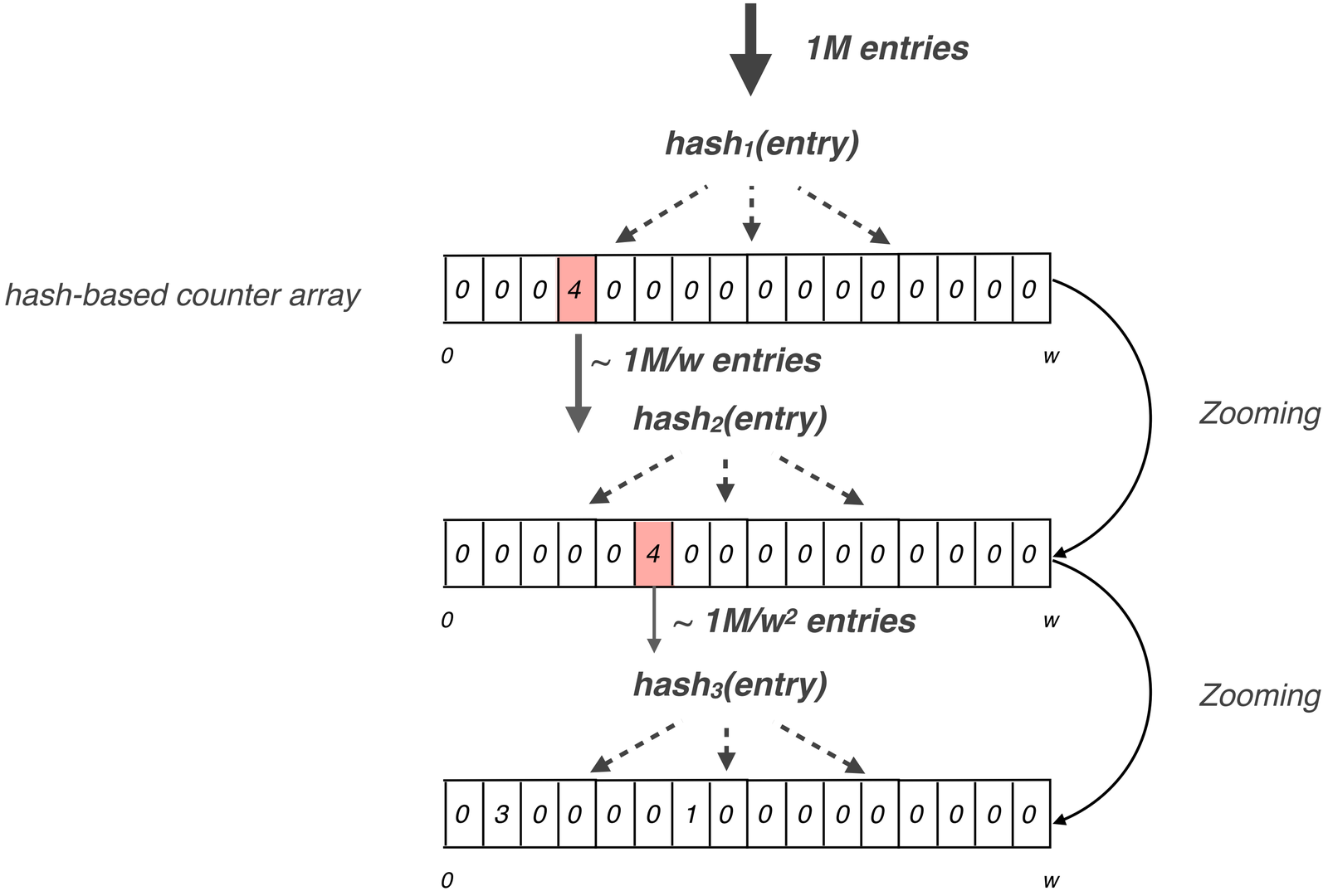
The process is repeated to decrease

the probability of collisions is low

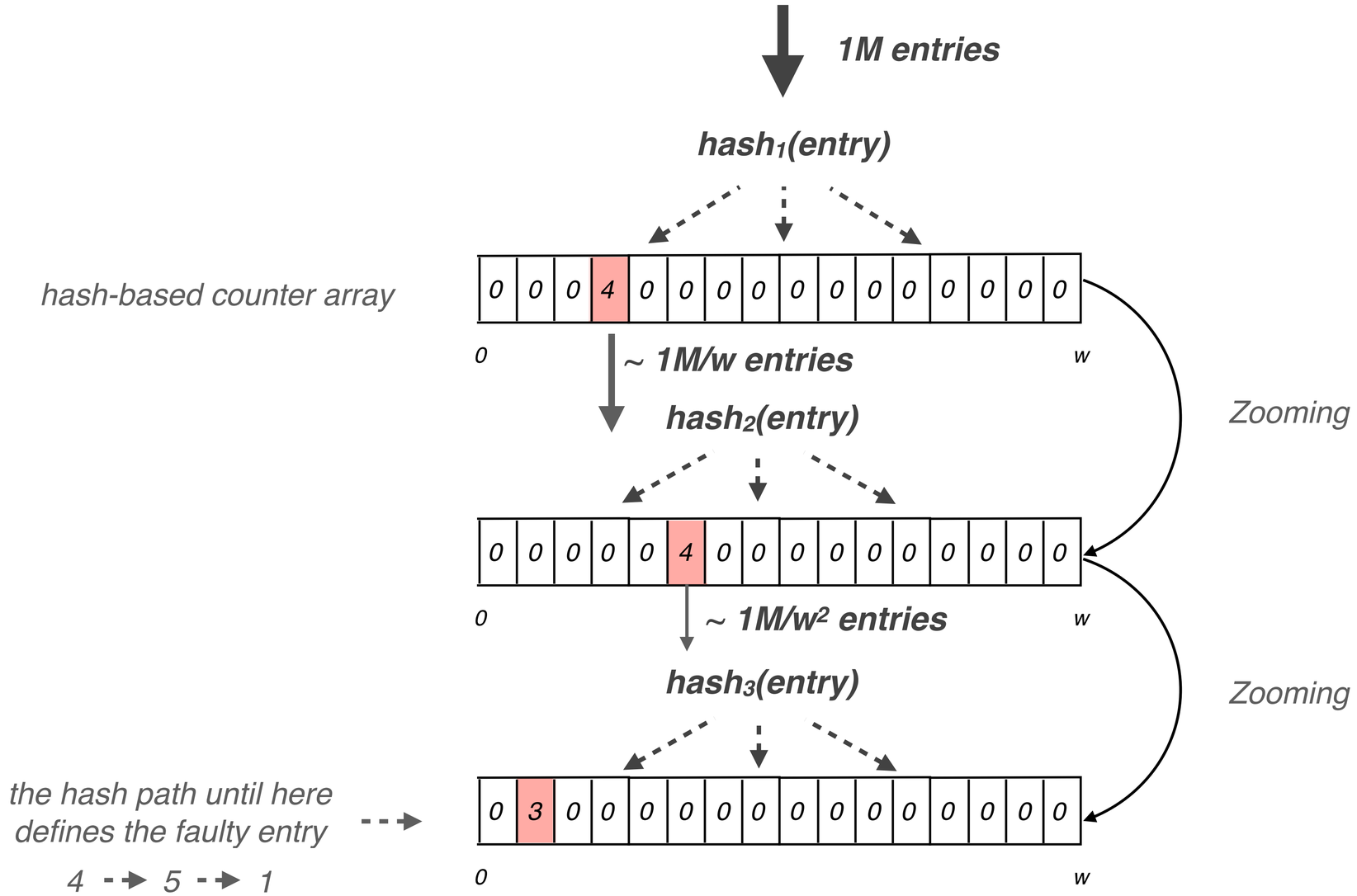


The process is repeated to decrease

the probability of collisions is low

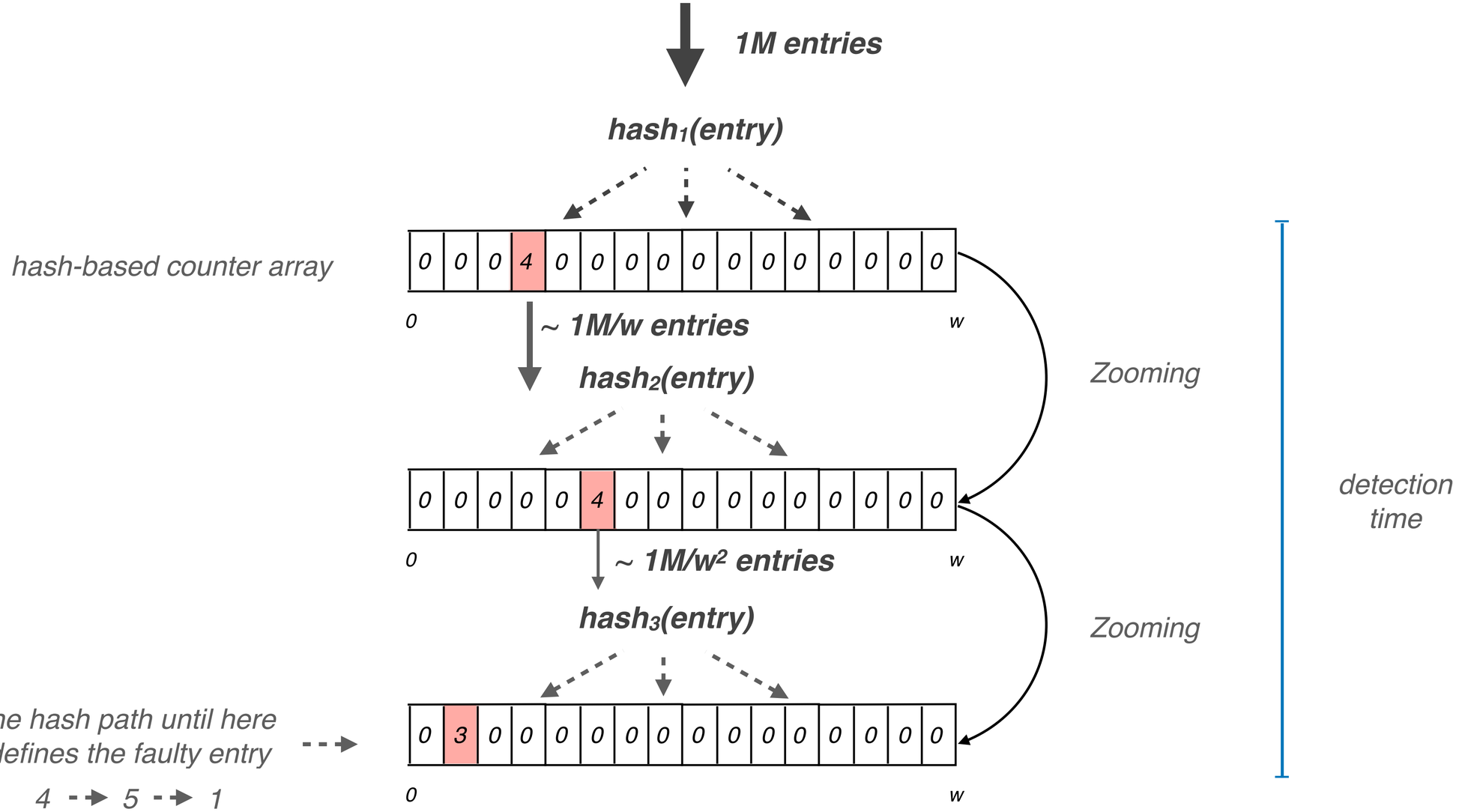


The switch identifies the faulty entry using the hash path from top to bottom



Hash-based counters allow ***FANcY*** to scale

at the cost of reducing the ***detection speed*** and ***accuracy***



We fully implemented ***FANcY***

and evaluated its accuracy and speed

- Software implementation

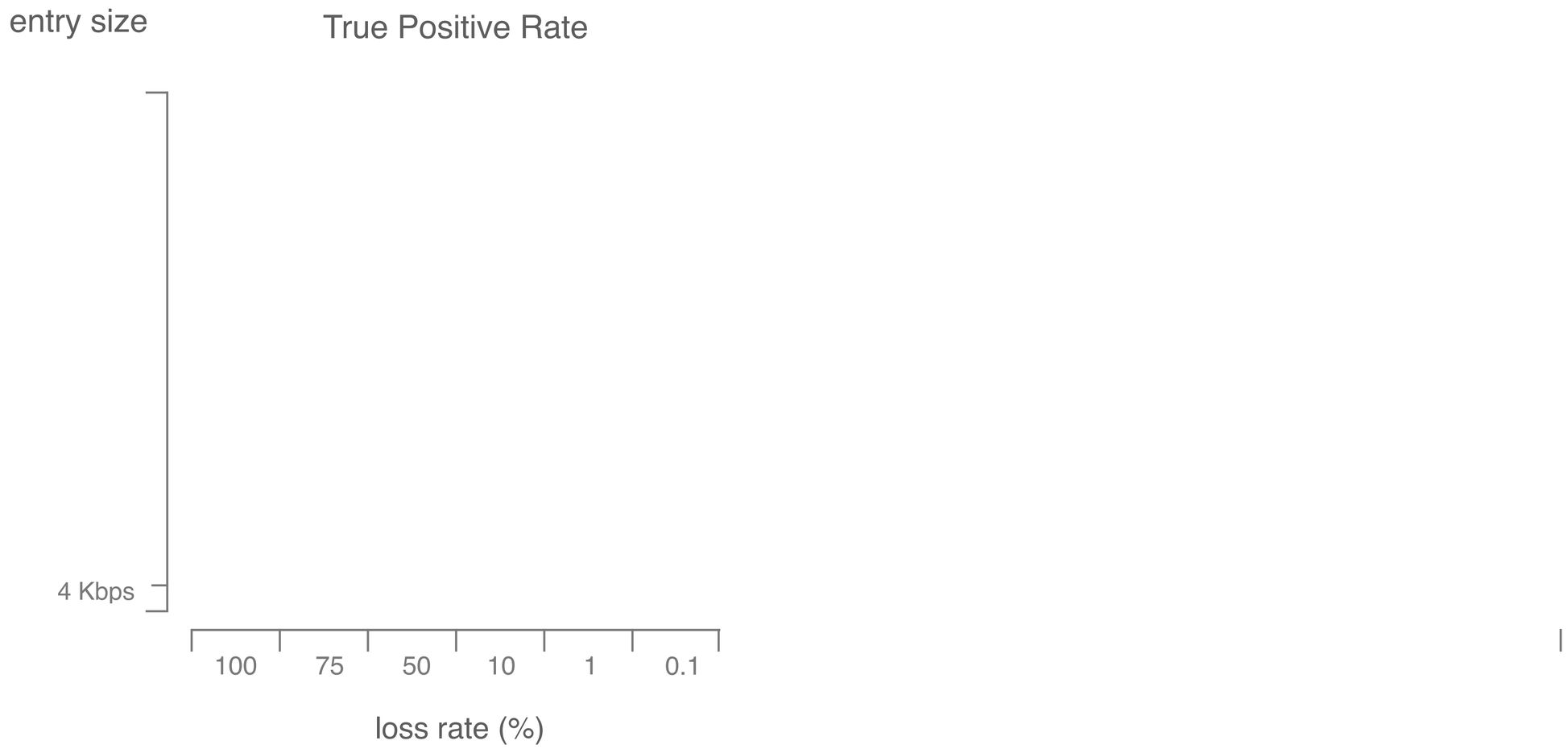
  ~9000 lines of C++ code
  extending ns-3

- Hardware implementation

  ~3000 lines of P4 code
  for Tofino switches

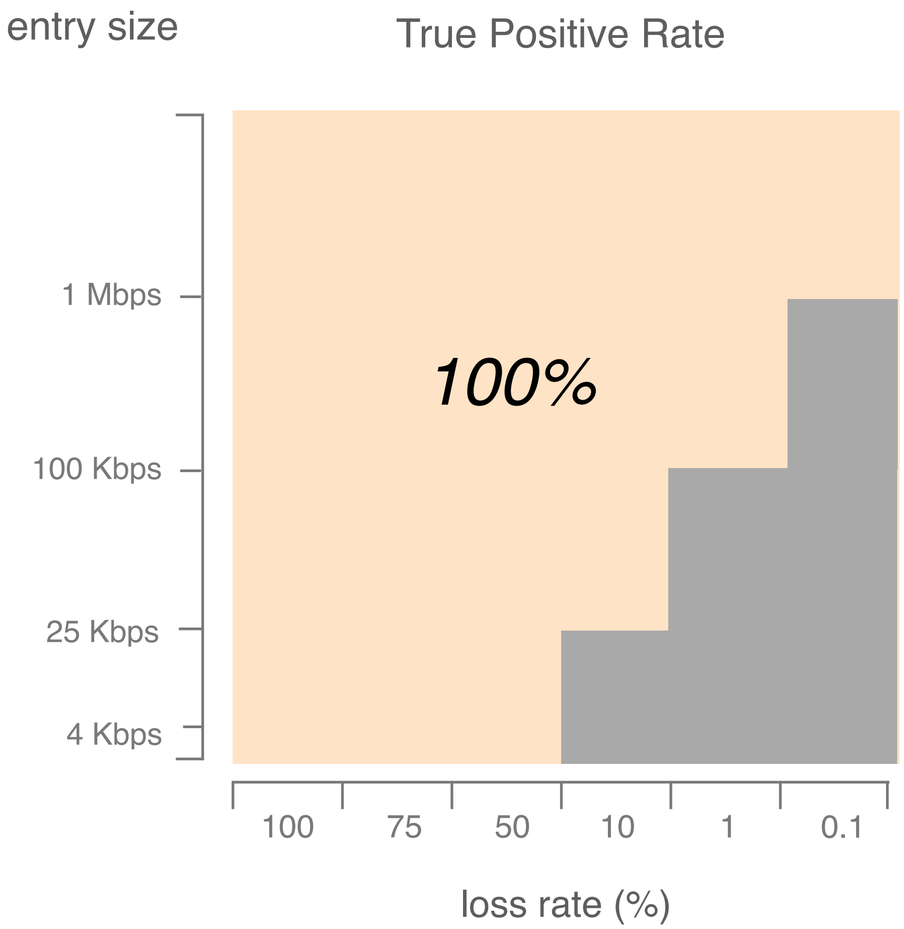
What is the accuracy and speed of ***hash-based counters*** for different entry sizes and loss rates?

Methodology	single-entry failures 10 ms of inter-switch delay 30 second runs
Hash-based counters	3-layer 200ms counting time

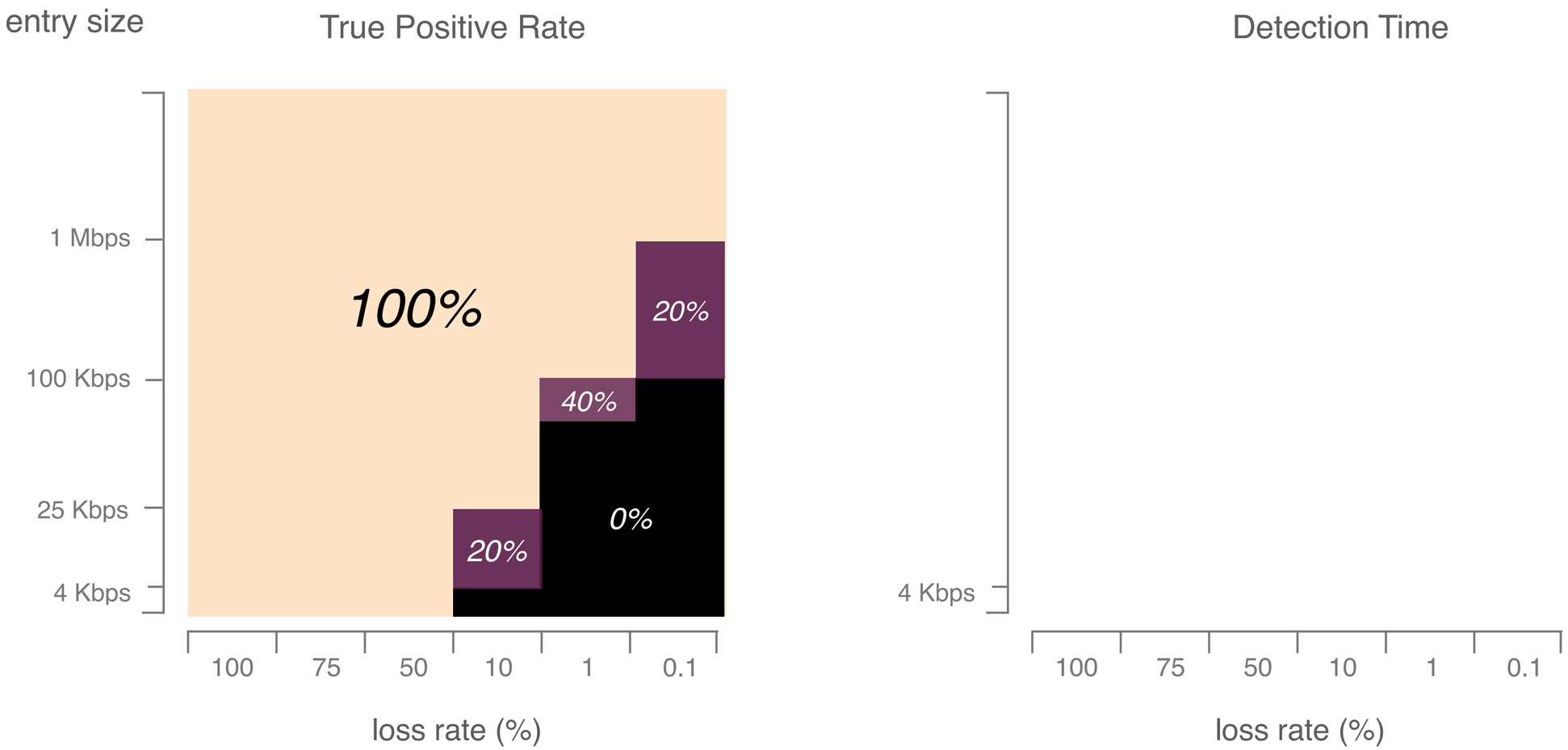

entry size
True Positive Rate
4 Kbps
100
75
50
10
1
0.1
loss rate (%)

Hash-based counters detect all the ***gray*** failures

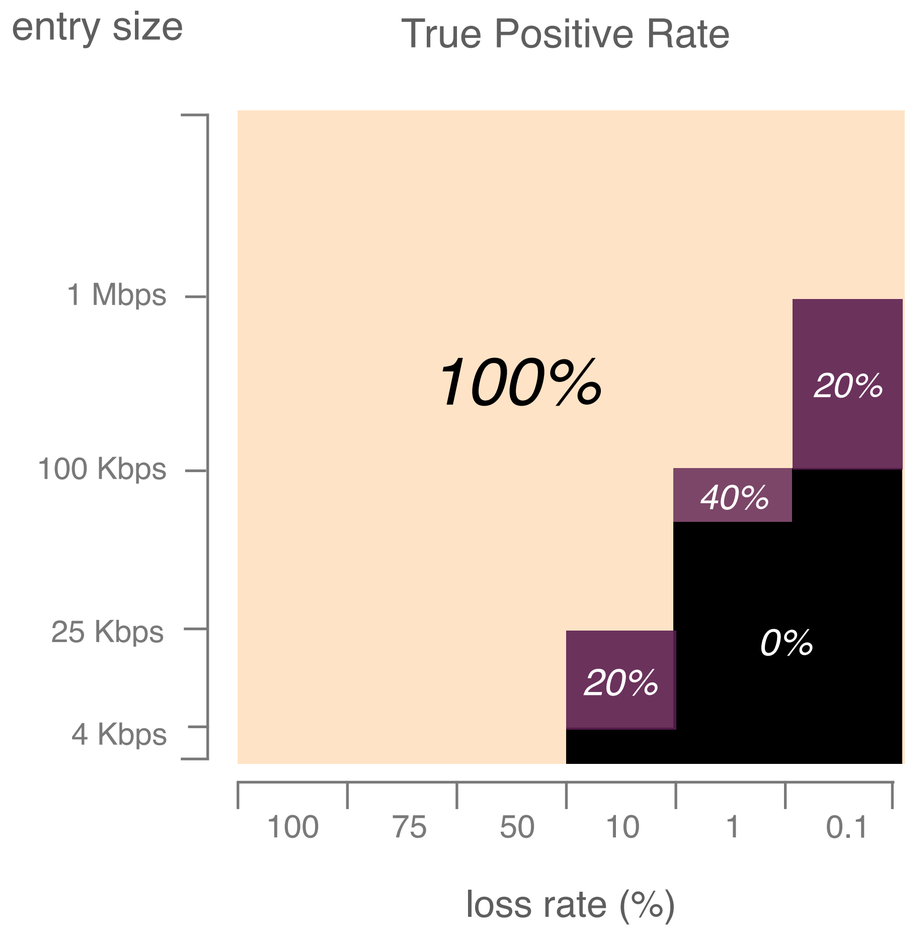
as long as there is enough traffic

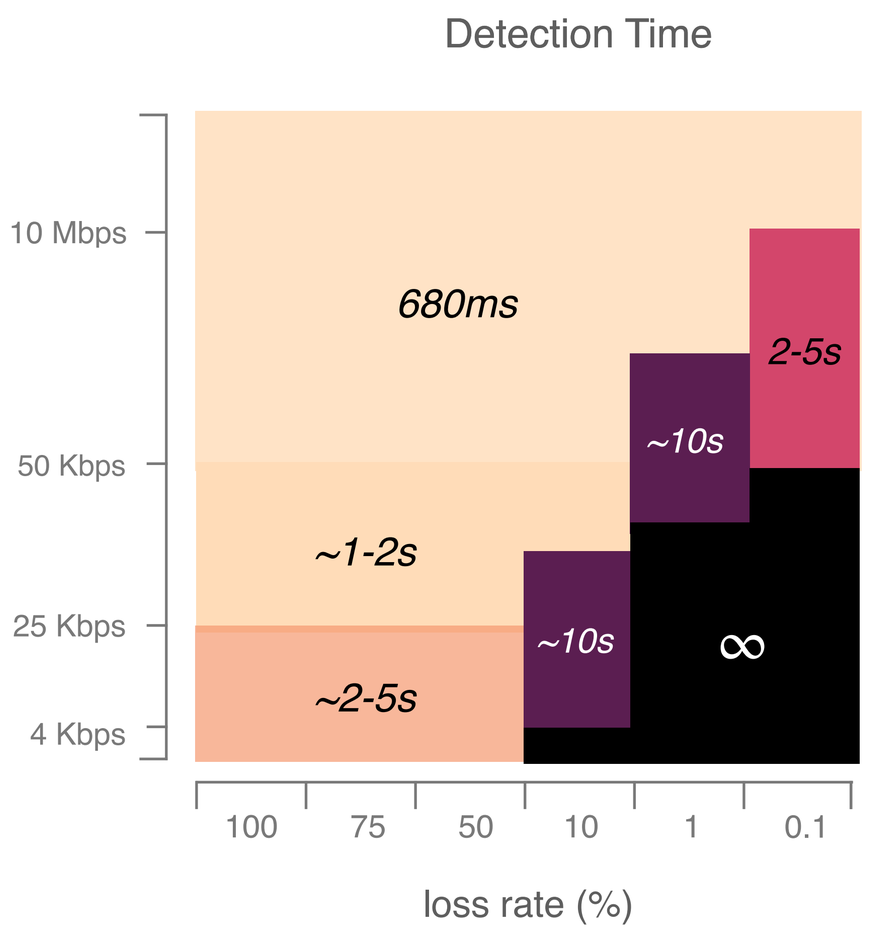


Accuracy decreases as the traffic and loss rate decrease


entry size
True Positive Rate
1 Mbps
100 Kbps
25 Kbps
4 Kbps
100%
20%
40%
0%
20%
100
75
50
10
1
0.1
loss rate (%)
Detection Time
4 Kbps
100
75
50
10
1
0.1
loss rate (%)

Detection times follow a similar pattern





FANcY
[SIGCOMM'22]

Local
hard and gray
failure detection

Blink
[NSDI'19]

Remote
failure detection

Hw-accelerated
control planes
[HotNets'18]

Hardware-based
control plane
implementation

1 detecting ***gray*** network failures

3 computing new paths for ***100,000s destinations***

4 updating the forwarding state for ***100,000s entries***

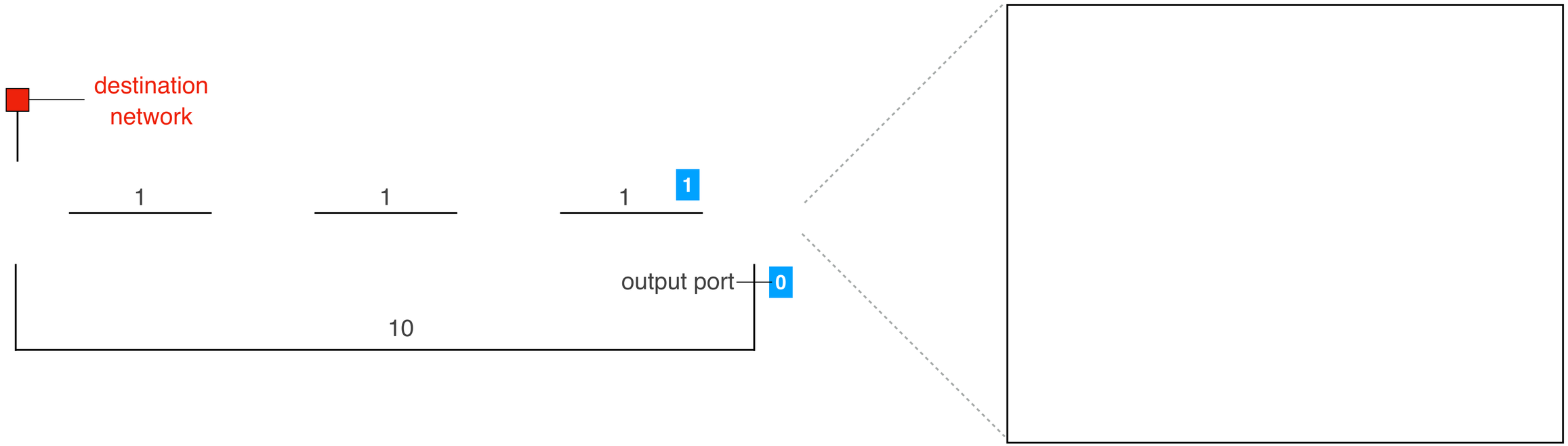
3 computing new paths for ***100,000s destinations***

4 updating the forwarding state for ***100,000s entries***

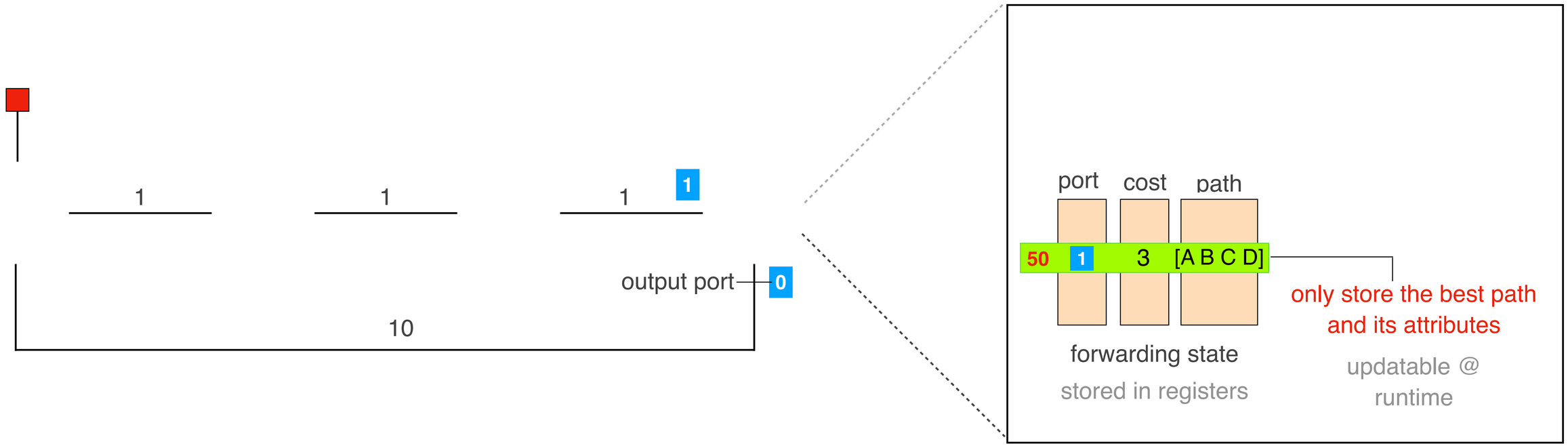
can programmable data planes
run these tasks?

Programmable data planes can process and perform computations on ***billions*** of packets per second

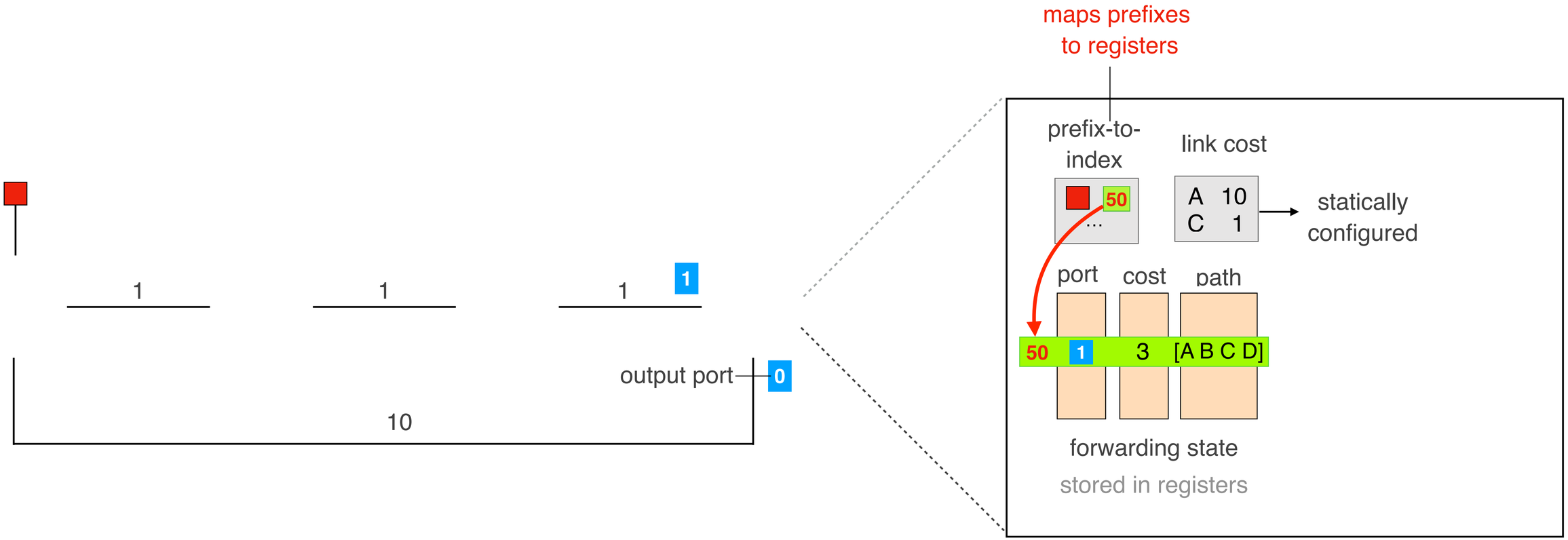
- Read & modify packet headers
  e.g. to update network state
- Basic operations
  e.g. +, -, >, <, min, max…
- Add or remove custom headers
  e.g. to carry routing information
- Keep and update state
  e.g. to save best paths

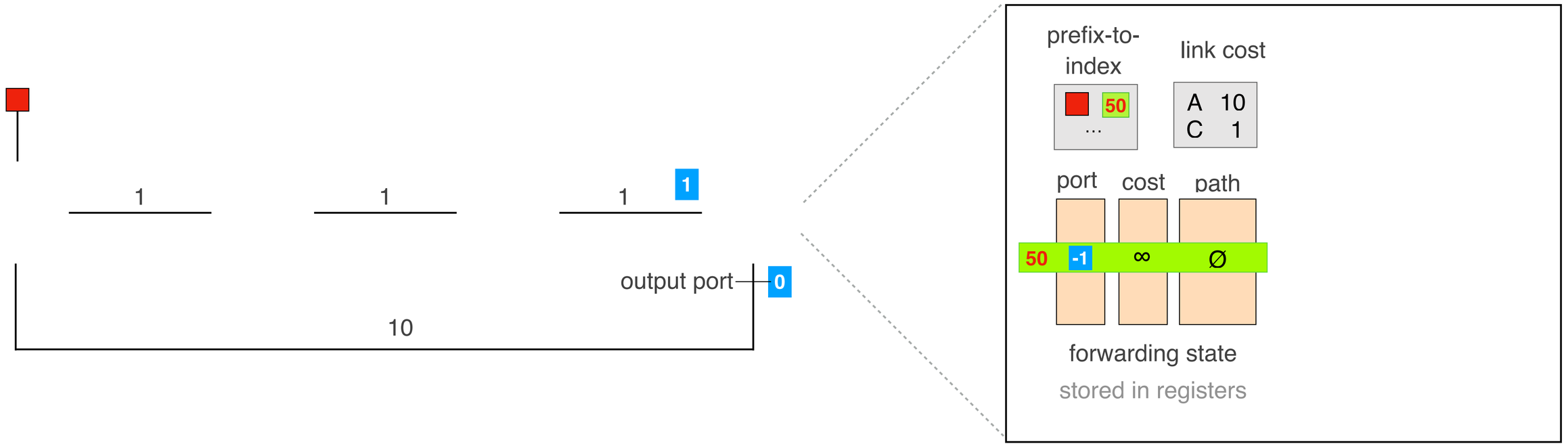

destination
network
1
1
1
1
output port
0
10

Use registers to store the best paths and their attributes



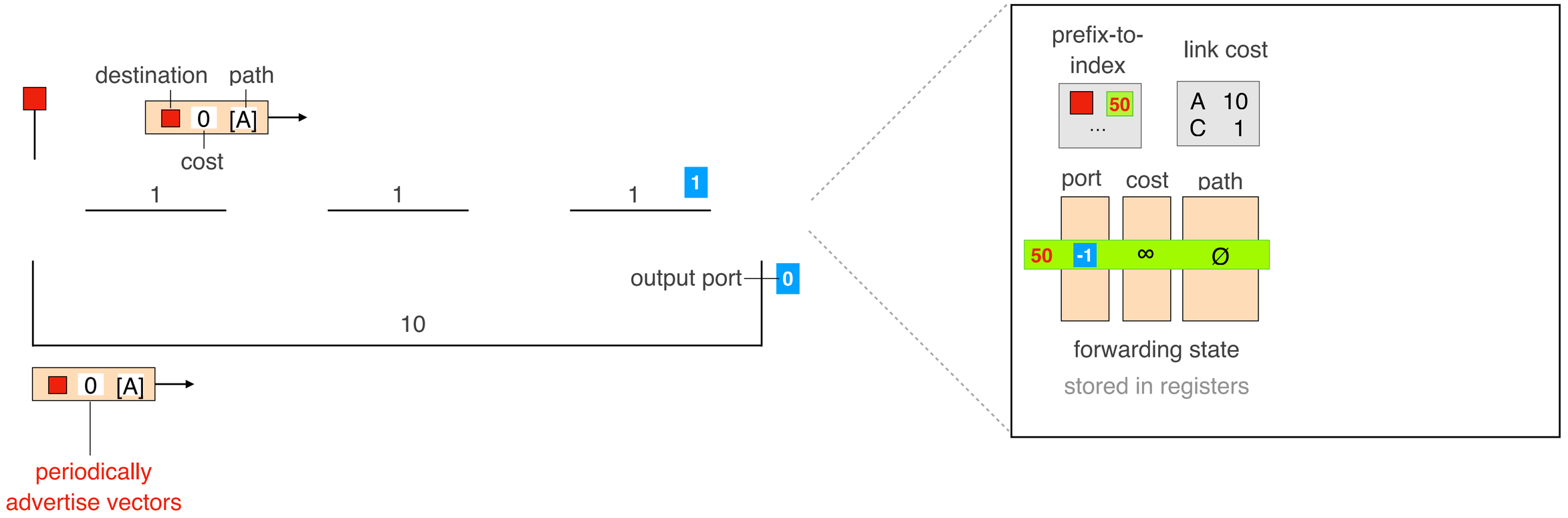
Use tables to store link costs and to map destinations to their register entries




1
1
1
1
output port
0
10
prefix-to-index
link cost
50
...
A 10
C 1
port
cost
path
50
-1
∞
Ø
forwarding state
stored in registers

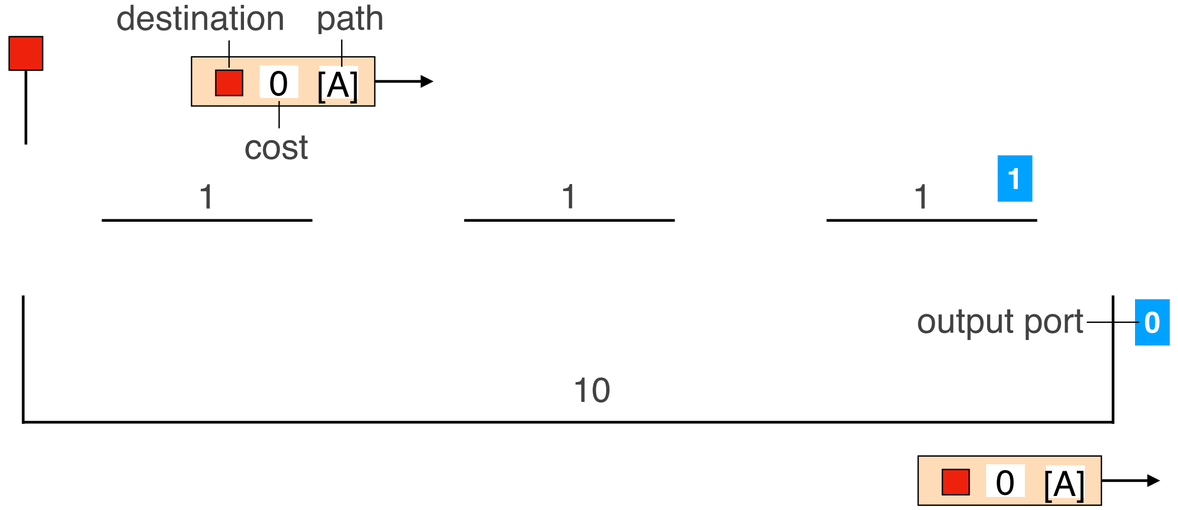
Switches periodically

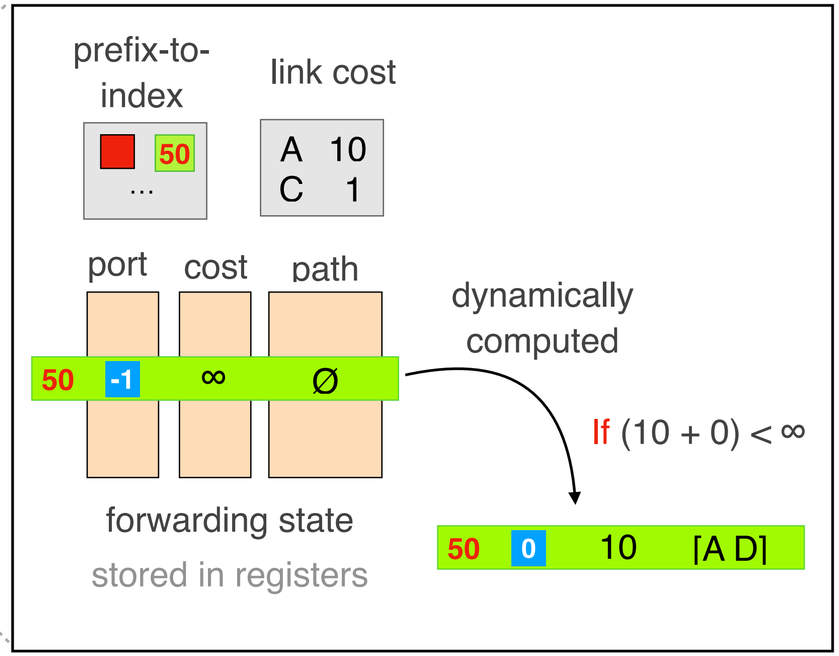
advertise vectors to neighbors

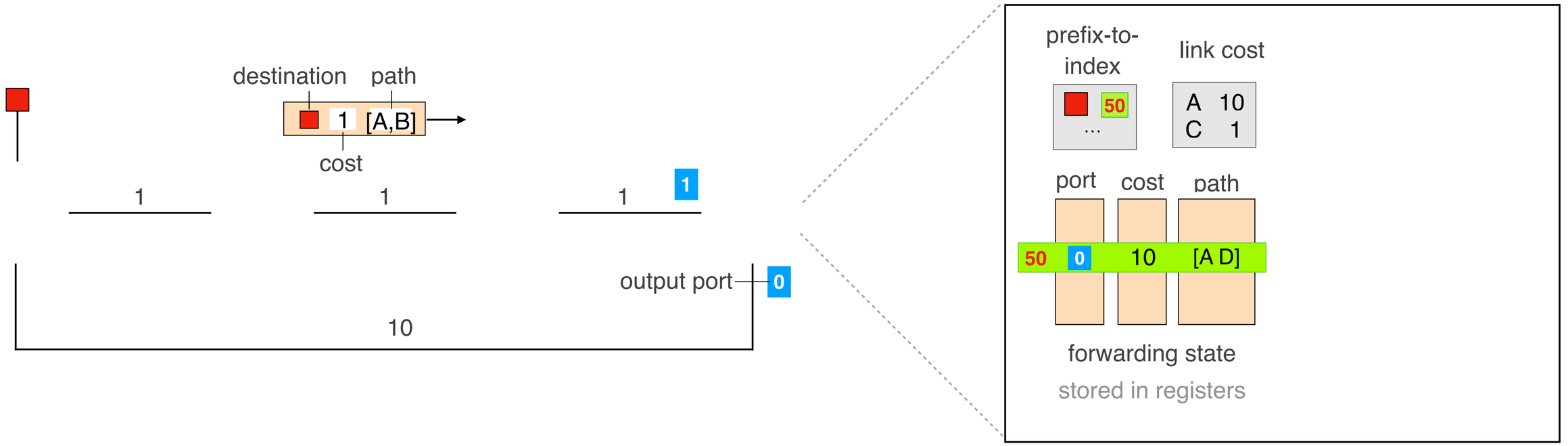


Upon receiving advertisements,

switches recompute their forwarding state

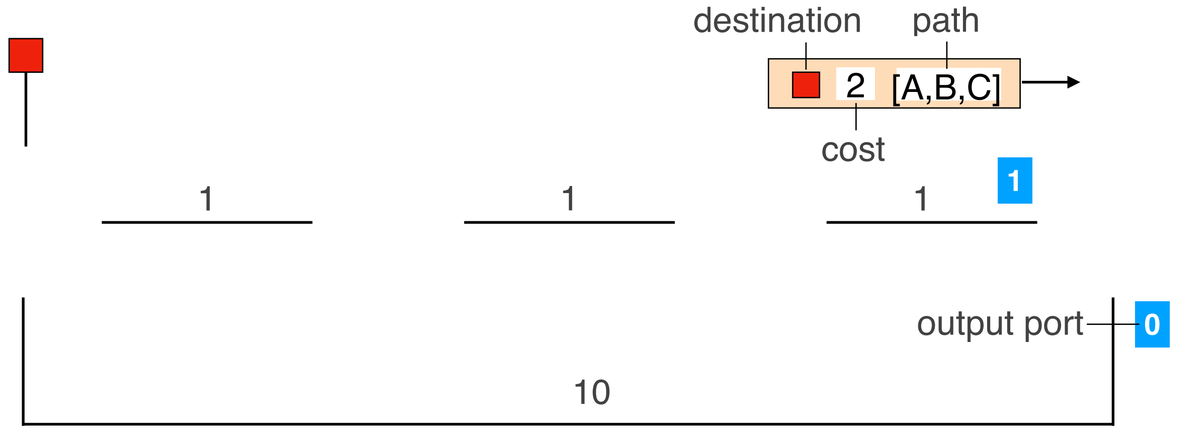


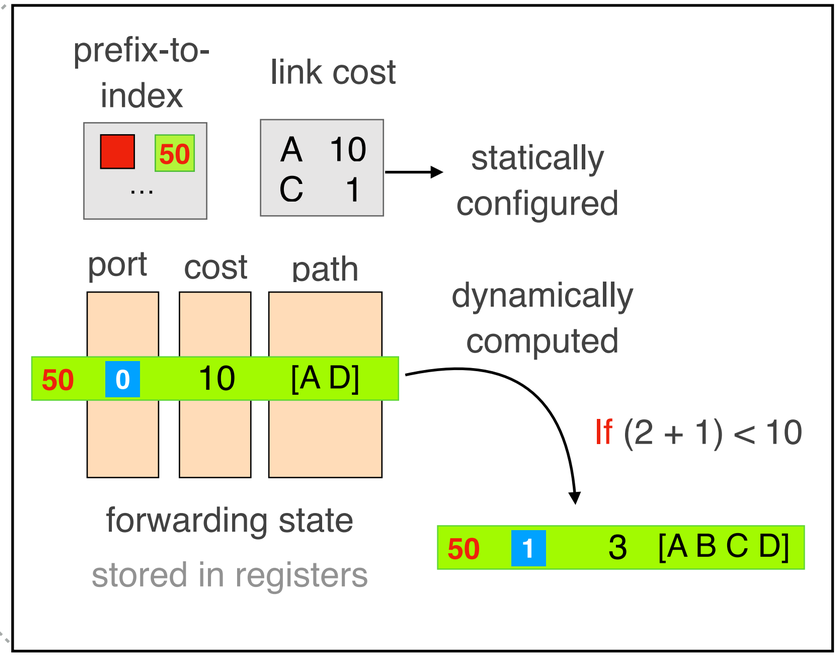



destination
path
1
[A,B]
cost
1
1
1
1
output port
0
10
prefix-to-index
50
...
link cost
A 10
C 1
port
cost
path
50
0
10
[A D]
forwarding state
stored in registers

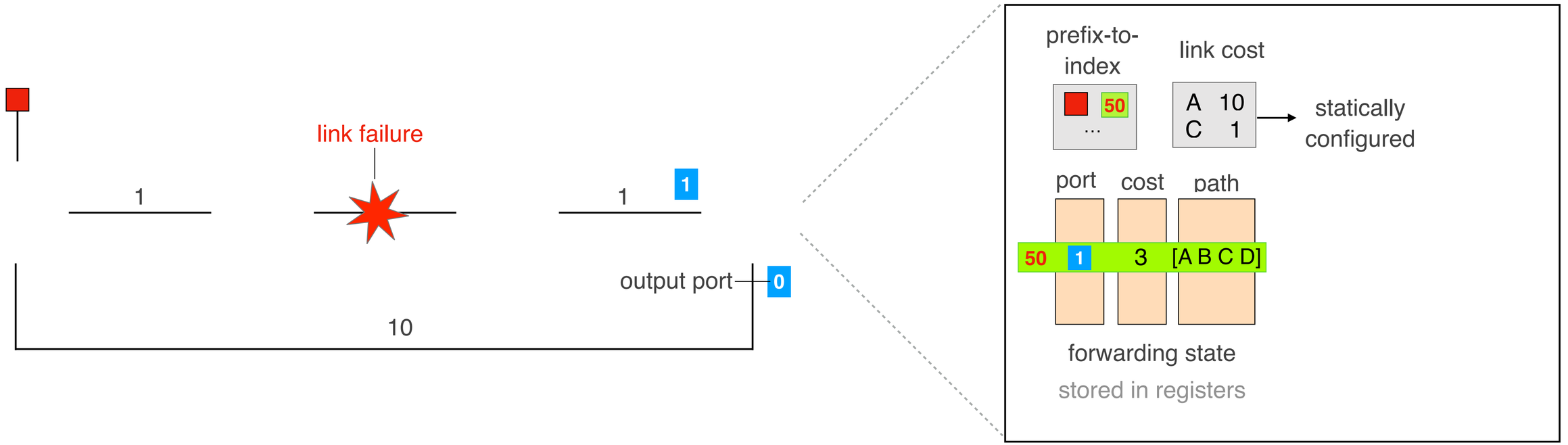
Upon receiving advertisements,

switches recompute their forwarding state



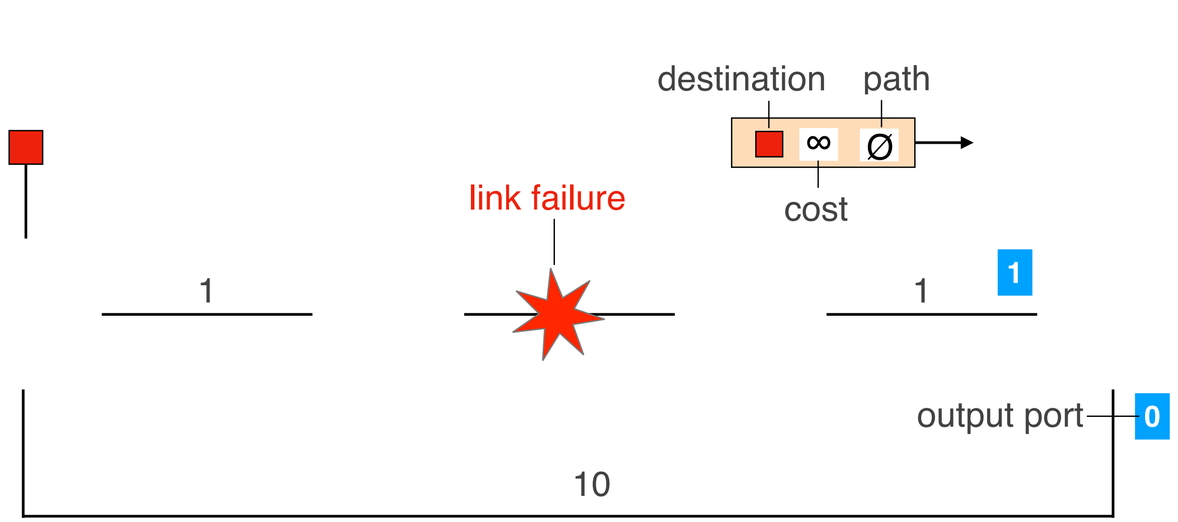


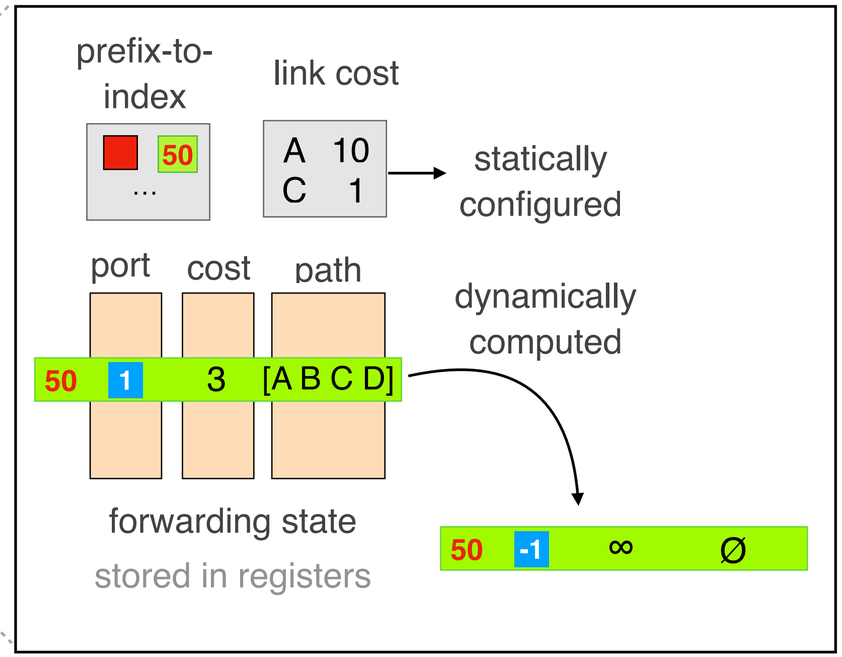
Computes new forwarding state after a a link failure



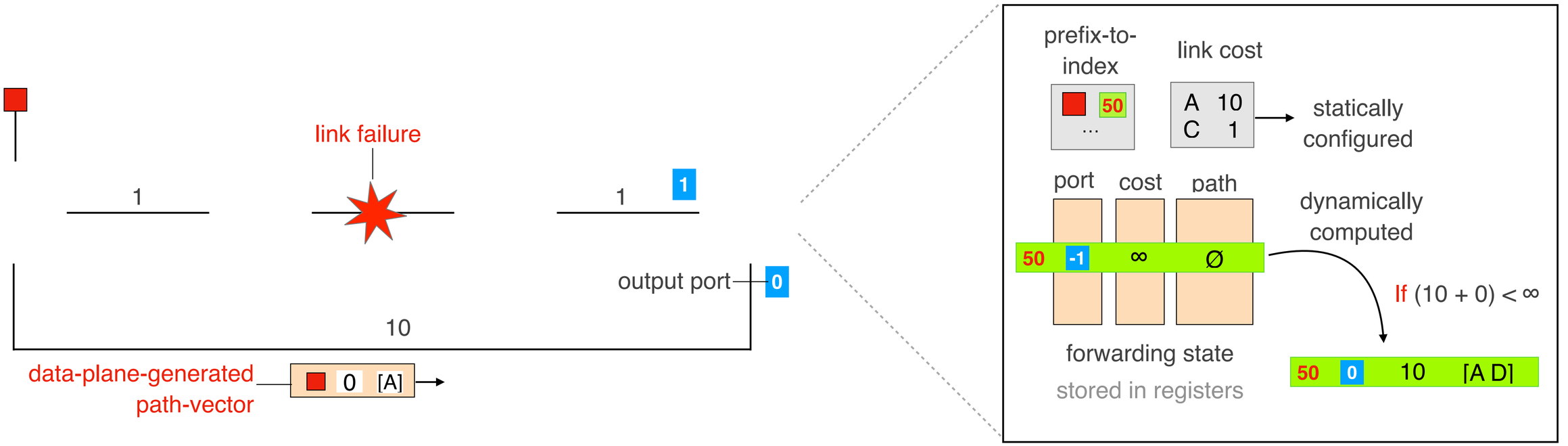
Computes new forwarding state

after a a link failure





Computes new forwarding state after a a link failure



We have a working prototype implemented in $P4_{16}$

Implementation

>2000 lines of P4 code

run on software model (bmv2)

Capabilities

Intra-domain destinations

path-vector routing

Inter-domain destinations

BGP-like route selection
& BGP export policies

Offloading control plane tasks

it is not ***always*** a good idea

Some tasks cannot be offloaded

e.g. crypto operations

Some tasks should not be offloaded

e.g. implementing the entire BGP/TCP protocol

Offloading routing tasks consumes hardware ressources

and those cannot be reused for other applications…

Improving network ***failure detection***

and ***recovery*** *with programmable data planes*

Improving network ***failure detection***

and ***recovery*** *with programmable data planes*

FANcY

[SIGCOMM'22]

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

Improving network ***failure detection***

and ***recovery*** *with programmable data planes*

explores running control plane tasks in the data plane

using programmable data planes capabilities

Hw-accelerated control planes

[HotNets'18]

computes and updates forwarding state upon network changes

running a path-vector algorithm and a simple BGP-like route selection

Improving network ***failure detection***

and ***recovery*** *with programmable data planes*

FANcY
[SIGCOMM'22]

Local
gray and hard
failure detection

Blink
[NSDI'19]

Remote
failure detection

Hw-accelerated
control planes
[HotNets'18]

Hardware-based
control plane
implementation