FAst In-Network Gray Failure Detection for ISPs

Edgar Costa Molero(1), Stefano Vissicchio(2), Laurent Vanbever(1)

SIGCOMM’22
August, 25 2022

(1)

(2)
ISP network
For me it does work!

it does not load!
it does not load!

ISP network

let me check
it does not load!

all seems fine!

network monitoring

ISP network
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**
- **✗** Heartbeat protocols (e.g., BFD)
  - only the heartbeat packets
- **✗** Sending traffic probes
  - only selected probes

**Passive**
- **✗** Packet counters (e.g., SNMP)
  - only available switch counters
- **✗** 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 *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*.
Introducing

*FANcY*: Fast In-network *Gray* Failure Detection for ISPs
We designed FANcY to work with **ISP network characteristics**

---

required memory to operate

---

<table>
<thead>
<tr>
<th>Collection rate</th>
<th>×</th>
<th>Collection complexity</th>
<th>×</th>
<th>Min time to compare</th>
<th>&lt;</th>
<th>Switch memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link bandwidth</td>
<td></td>
<td>Memory required per packet</td>
<td></td>
<td>Link delay</td>
<td></td>
<td>Fixed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Control plane reading speed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#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 failures 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 \textit{upstream} flags the entry as \textit{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, \textit{FANcY} uses state machines for each counting session.
To achieve perfect **synchronization** and **reliability**, $\textit{FANcY}$ uses **state machines** for each counting session.
To achieve perfect **synchronization** and **reliability**, \textit{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!
We can leverage the fact that *gray* failures tend to be *sparse* and *aggregate* multiple traffic entries into the *same counter*. 
We can leverage the fact that *gray* failures tend to be *sparse* and *aggregate* multiple traffic entries into the *same counter*.

![Diagram](attachment://diagram.png)

**hash-based counter array**

<table>
<thead>
<tr>
<th>7</th>
<th>5</th>
<th>4</th>
<th>9</th>
<th>3</th>
<th>5</th>
<th>1</th>
<th>6</th>
<th>2</th>
<th>5</th>
<th>9</th>
<th>6</th>
<th>1</th>
<th>5</th>
<th>3</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1M entries uniformly distributed
We can leverage the fact that *gray* failures tend to be *sparse* and *aggregate* multiple traffic entries into the *same counter*.
We can leverage the fact that gray failures tend to be sparse and aggregate multiple traffic entries into the same counter.
We can leverage the fact that gray failures tend to be sparse and aggregate multiple traffic entries into the same counter.
We can leverage the fact that *gray* failures tend to be *sparse* and *aggregate* multiple traffic entries into the *same counter*.
We can leverage the fact that *gray* failures tend to be *sparse* and *aggregate* multiple traffic entries into the *same counter*.
We can leverage the fact that gray failures tend to be sparse and aggregate multiple traffic entries into the same counter.
**Hash-based counters** allow **FANcY** to scale at the cost of reducing the **detection speed** and **accuracy**.

---

**Hash-based counter array**

![Diagram showing hash-based counter array with zooming and detection time metrics.](image-url)
FANcY can combine *dedicated counter entries* with the *hash-based counters*.
We evaluated *FANCY* accuracy and speed

- **Software simulations:** \( \sim 9000 \) lines of C++ code extending ns-3
  
  *#1* How does FANCY perform depending on the gray failure type and the volume of traffic affected

- **Hardware implementation:** \( \sim 3000 \) lines of P4 code
  
  *#2* Does FANCY work on Intel Tofino programmable switches?
We evaluated **FANcY** accuracy and speed

- Software simulations: ~9000 lines of C++ code extending ns-3
  - What is the minimum amount of traffic required for FANcY to detect different types of gray failures?
- Hardware implementation: ~3000 lines of P4 code
- Multi-entry failures, uniform random drops, eval with CAIDA traces

*more in the paper!*
How does FANcY perform depending on the gray failure type and the volume of traffic 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 *counting time of 200ms*
FANcY’s *hash-based counters* performance with 3 *layers* and *counting time of 200ms*
FANcY’s *hash-based counters* performance with *3 layers* and *counting time of 200ms*

![Diagram](image-url)
FANcY’s **hash-based counters** performance with **3 layers** and **counting time of 200ms**
FANcY’s **hash-based counters** performance with **3 layers** and **counting time of 200ms**

### True Positive Rate

- **Entry Size**
  - 1 Mbps
  - 100 Kbps
  - 25 Kbps
  - 4 Kbps

- **Loss Rate (%)**
  - 100%
  - 20%
  - 40%
  - 0%

### Detection Time

- **Loss Rate (%)**
  - 100%
  - 20%
  - 50 Kbps
  - 10 Mbps

- **Detection Time**
  - 680ms
  - ∞
  - ~1-2s
  - ~2-5s
  - ~10s
  - 2-5s
  - 10s
  - ∞
#2 Does \textit{FANcY} work on \textit{Intel Tofino} programmable switches?

\textbf{Dedicated counters} are exchanged every 200 ms

\textbf{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.
**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 in today’s hardware
implemented and tested on Intel Tofino Switches
F Ast In-Network \textit{Gray} Failure Detection for ISPs

\url{github.com/nsg-ethz/FANCY}

Thank you!

SIGCOMM’22
August, 25 2022

Edgar Costa Molero\textsuperscript{(1)},
Stefano Vissicchio\textsuperscript{(2)},
Laurent Vanbever\textsuperscript{(1)}