NetBouncer: Active Device and Link Failure Localization in Data Center Networks

Cheng Tan¹, Ze Jin², Chuanxiong Guo³, Tianrong Zhang⁴, Haitao Wu⁵, Karl Deng⁴, Dongming Bi⁴, and Dong Xiang⁴

Anna
Network operator
Anna
Network operator

Customers

Unstable network
High latency
Low throughput
Traditional monitoring system queries switches (e.g., SNMP)
This is a true story

• Root cause
  – A firmware bug on a switch link (bit flips of a fabric module)
  – It silently drops packets without any signal

• Gray failure*
  – Differential observability
  – Cause major cloud breakdowns
  – Localizing gray failures is essential for high availability

Why yet another monitoring system?

• Our response to network gray failures is NetBouncer

• Indeed, many monitoring systems
  – Academia: LossRadar, Trumpet, deTector, Netscope, ...
  – Industry: Pingmesh, NetNORAD, 007, Passive probing, ...

• In production, there are four requirements:
  1. Catch gray failures---from a server’s perspective
  2. Transparent to current software stack
  3. Pinpoint failures in links or devices
  4. Few false positives (i.e., misreporting) and false negatives
NetBouncer overview

NetBouncer is an active probing system which *infers failures* from path probing data.
Rest of the talk

1. How to achieve light-weight and explicit probing?

2. Which paths should be probed?

3. How to infer failures from path probing data?
Rest of the talk

1. How to achieve light-weight and explicit probing?

2. How to design an eligible probing plan?

3. How to infer failures from path probing data?
Active probing system requires explicit and efficient probing

- Server can choose which links to evaluate with explicit probing
- NetBouncer uses IP-in-IP to explicitly probe a path
  - IP-in-IP forwarding is implemented in hardware.
Active probing system requires explicit and efficient probing

- Server can choose which links to evaluate with explicit probing
- NetBouncer uses IP-in-IP to explicitly probe a path
  - IP-in-IP forwarding is implemented in hardware.

- A server asks a switch to “bounce back” probing packets
  - Simple model and simple fault tolerance
Which paths should be probed?

How to achieve light-weight and explicit probing?

How to infer failures from path probing data?
Observation vs. inference: from path probing to failures
Observation vs. inference: from path probing to failures

- Undirected graph (vertex=device, edge=link)
- Failures are probabilistic
Observation vs. inference: from path probing to failures

- Undirected graph (vertex=device, edge=link)
- Failures are probabilistic

<table>
<thead>
<tr>
<th>Observation</th>
<th>49 / 100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>49%</td>
</tr>
</tbody>
</table>

goal: is this link faulty?
Observation vs. inference: from path probing to failures

• Undirected graph (vertex=device, edge=link)
• Failures are probabilistic
Observation vs. inference: from path probing to failures

49% ?

possibility 1
49% ↓
49 / 100

possibility 2
49% ↑
49 / 100

possibility 3
70% ↑
49 / 100

...
Observation vs. inference: from path probing to failures

- Infer the link success probabilities from path probing observations
- Report links as faulty with success probability < threshold (e.g., 99%)
Observation vs. inference: from path probing to failures

- Infer the link success probabilities from path probing observations
- Report links as faulty with success probability < threshold

Which paths should be probed, s.t. all link success probabilities can be uniquely determined?
Real-world constraints complicate path selection

- Constraint 1: some switches may not bounce the probing
- Constraint 2: a probing path starts/ends at the same server
- Sometimes, it is impossible to uniquely identify all links
Real-world constraints complicate path selection

• Constraint 1: some switches may not bounce the probing
• Constraint 2: a probing path starts/ends at the same server
• Sometimes, it is impossible to uniquely identify all links
Real-world constraints complicate path selection

- Constraint 1: some switches may not bounce the probing
- Constraint 2: a probing path starts/ends at the same server
- Sometimes, it is impossible to uniquely identify all links

$y_j = x_j \times x_3$
Real-world constraints complicate path selection

• Constraint 1: some switches may not bounce the probing
• Constraint 2: a probing path starts/ends at the same server

• Sometimes, it is **impossible** to uniquely identify all links
Real-world constraints complicate path selection

- Constraint 1: some switches may not bounce the probing
- Constraint 2: a probing path starts/ends at the same server
- Sometimes, it is impossible to uniquely identify all links

$log(y_1) = \log(x_1) + \log(x_3)$
$log(y_2) = \log(x_1) + \log(x_4)$
$log(y_3) = \log(x_2) + \log(x_3)$
$log(y_4) = \log(x_2) + \log(x_4)$

Not full rank
Real-world constraints complicate path selection

- Constraint 1: some switches may not bounce the probing
- Constraint 2: a probing path starts/ends at the same server
- Sometimes, it is impossible to uniquely identify all links

\[
\begin{align*}
\log(y_1) &= \log(x_1) + \log(x_3) \\
\log(y_2) &= \log(x_1) + \log(x_4) \\
\log(y_3) &= \log(x_2) + \log(x_3) \\
\log(y_4) &= \log(x_2) + \log(x_4)
\end{align*}
\]

Links success probabilities \((x_1-x_4)\) can be arbitrary

Not full rank
A condition to uniquely identify link success probabilities

We proved a theorem (for Clos network), that provides
• a simple probing plan: each server probes all top-layer switches
• a necessary and sufficient condition for uniquely identifying P(link)
A condition to uniquely identify link success probabilities

We proved a theorem (for Clos network), that provides

• a simple probing plan: each server probes all top-layer switches
• a necessary and sufficient condition for uniquely identifying P(link)

each node has at least one good path through it
Original graph

Subgraph with unique solution

Unsolvable part

No good paths pass this switch
Device failure detection

Original graph

Subgraph with unique solution

Faulty devices

No good paths pass this switch
Device failure detection

Original graph

How to infer the link failures from this subgraph?

Faulty devices

No good paths pass this switch
How to infer the link failures from the solvable subgraph?

How to achieve light-weight and explicit probing?

Which paths should be probed?

How to infer the link failures from the solvable subgraph?
Link failure inference: an optimization problem

Assume packet drops are independent events.

Given the path probing data \( (y_j) \), how to infer the link success probabilities \( (x_i) \) that fits them the best?

\[
\begin{align*}
    y_1 &= \frac{50}{100} = x_1 \times x_2 \times x_3 \\

\end{align*}
\]

\[
\text{minimize } \sum_j (y_j - \prod_{i: \text{link}_i \in \text{path}_j} x_i)^2
\]

subject to \( 0 \leq x_i \leq 1, \forall i \)
Real-world data inconsistency induces false positives
Real-world data inconsistency induces false positives
Real-world data inconsistency induces false positives

![Diagram showing the percentage and number of false positives.

50/100 and 50/100 correspond to false positives with a 98% accuracy but a 2% loss.](image-url)
Real-world data inconsistency induces false positives

- Real-world data inconsistency
  - Measurements do not fully align
  - Inference results may overfit observations

\( 50/100 \quad 50/100 \quad 49/100 \)

\( 98\% \text{ (2\% loss)} \)
Real-world data inconsistency induces false positives

- Real-world data inconsistency
  - Measurements do not fully align
  - Inference results may overfit observations
- Solution: a specialized regularization

\[ \sum_{j} (y_j - \prod_{i: \text{link}_i \in \text{path}_j} x_i)^2 + \lambda \sum_{i} x_i (1 - x_i) \]
Evaluation questions

• In production, what failures have been detected by NetBouncer?
  – One real case, more in paper

• How accurate is NetBouncer compared with previous algorithms?

• What’s the performance of NetBouncer’s algorithm?
Real case: spine router gray failure

• Observations
  – Many customers experienced packet drops and latency increases
  – Traditional monitoring systems cannot pinpoint the failure

• NetBouncer detected this gray failure
  – One spine router silently dropped packets
  – Root cause was an issue in one of this switch’s linecard hardware
Accuracy comparison with previous algorithms

- Simulation setup:
  - 3-layer Clos network with 2.8K switches (48 ports), 27.6K servers and 82.9K links
  - 1% faulty links and 10 faulty devices

- Compare with two algorithms: deTector and NetScope

```
#false negative  #false positive
7.2k             10.8k
0.4              0
0                0
```

- deTector
- NetScope
- NetBouncer ($\lambda=1$)

Cannot guarantee zero-FP/FN; has FP/FN in other experiments
NetBouncer algorithm performance

- Xeon E5 2.4GHz CPU with 128GB memory
- One hour trace from 2016 (~130GB)
Related work

• Network tomography
  – Internet failure localization: NetScope, LIA, NetQuest
  – Heuristic algorithm: Tomo, detector
  – Require further investigation: Pingmesh, NetSonar, NetNorad

• Other troubleshooting systems
  – Panorama, Deepview, 007
  – Trumpet, LossRadar

• Explicit path probing
  – XPath and other source routing

• Probing plan design
  – Focus on minimizing number of paths
Conclusion

• A complete framework for data center network failure localization
  – An efficient path probing scheme
  – A necessary and sufficient condition for an eligible probing plan
  – A link failure inference algorithm

• NetBouncer has been deployed for three years and performs well