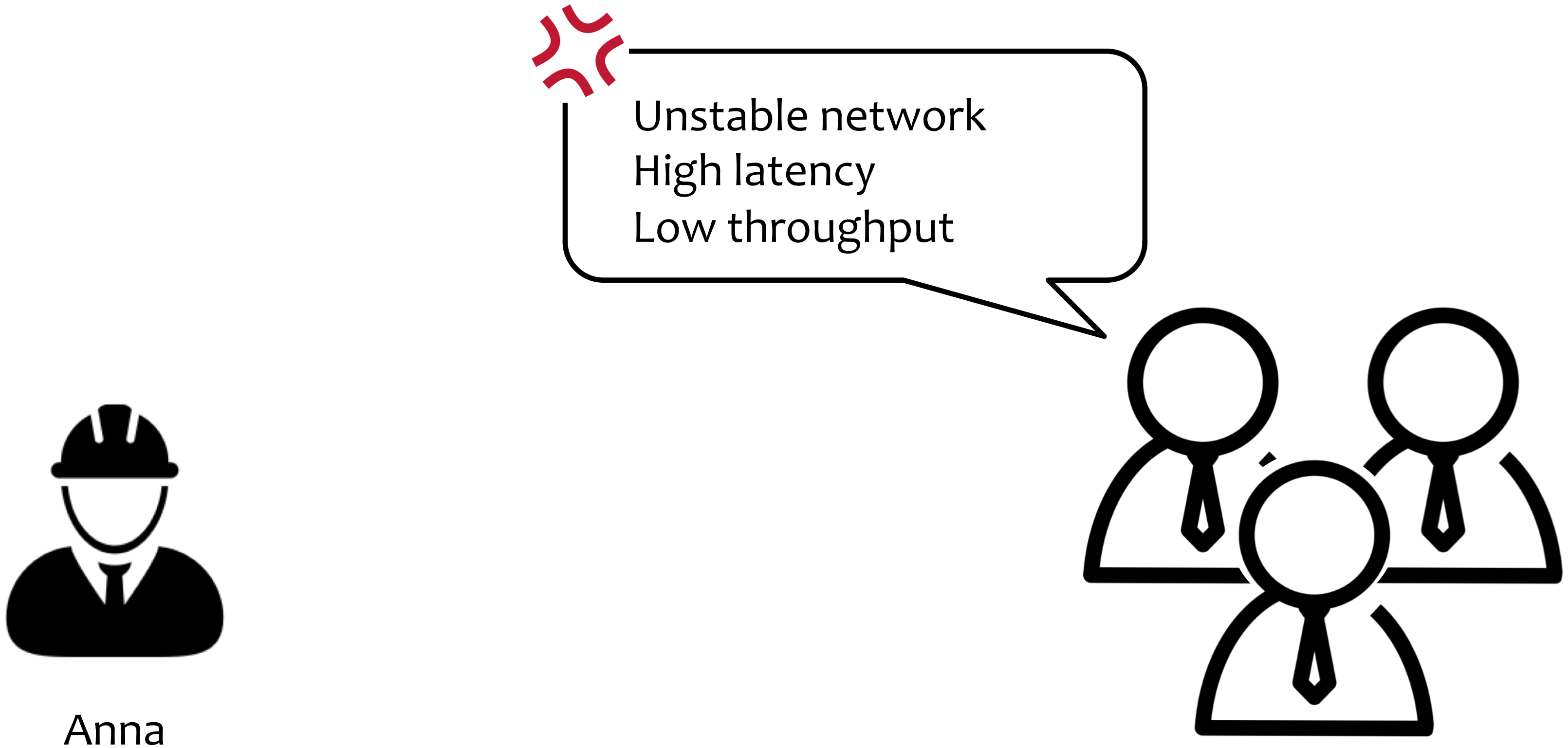
NetBouncer: Active Device and Link Failure Localization in Data Center Networks

Cheng Tan[1], Ze Jin[2], Chuanxiong Guo[3], Tianrong Zhang[4], Haitao Wu[5], Karl Deng[4], Dongming Bi[4], and Dong Xiang[4]

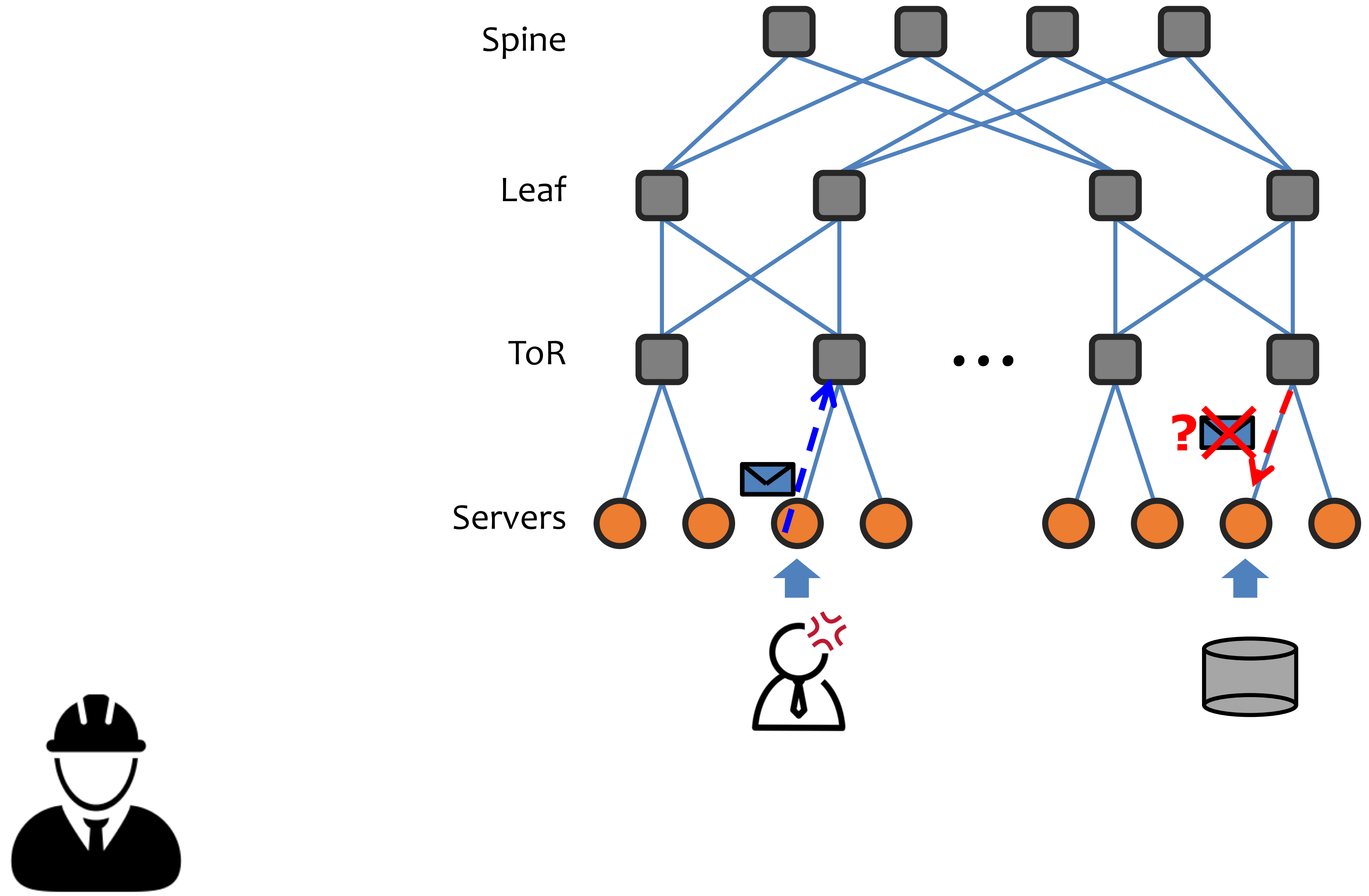
1. NYU 2. Cornell 3. Bytedance 4. Microsoft 5. Google

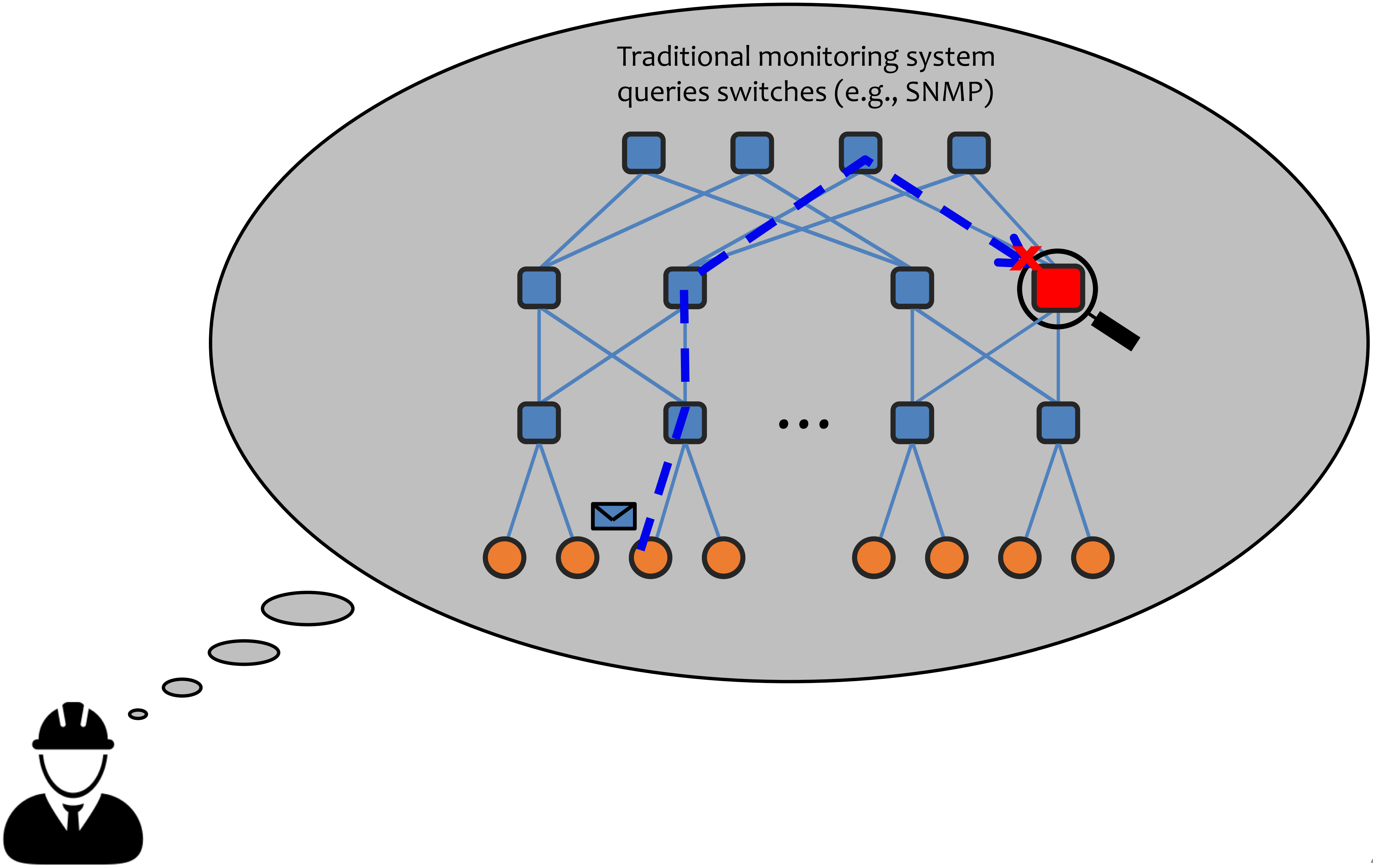
Anna
Network operator

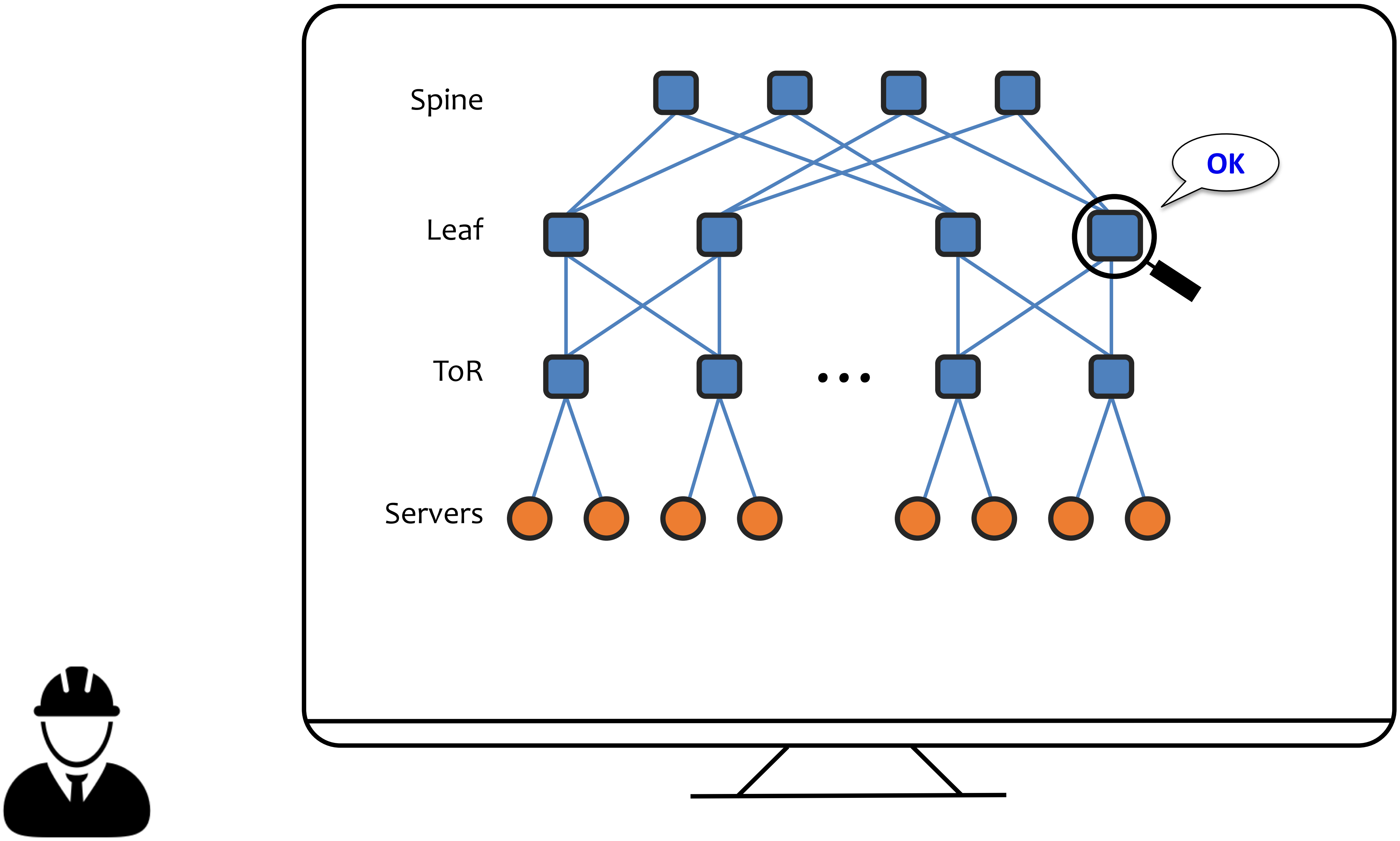


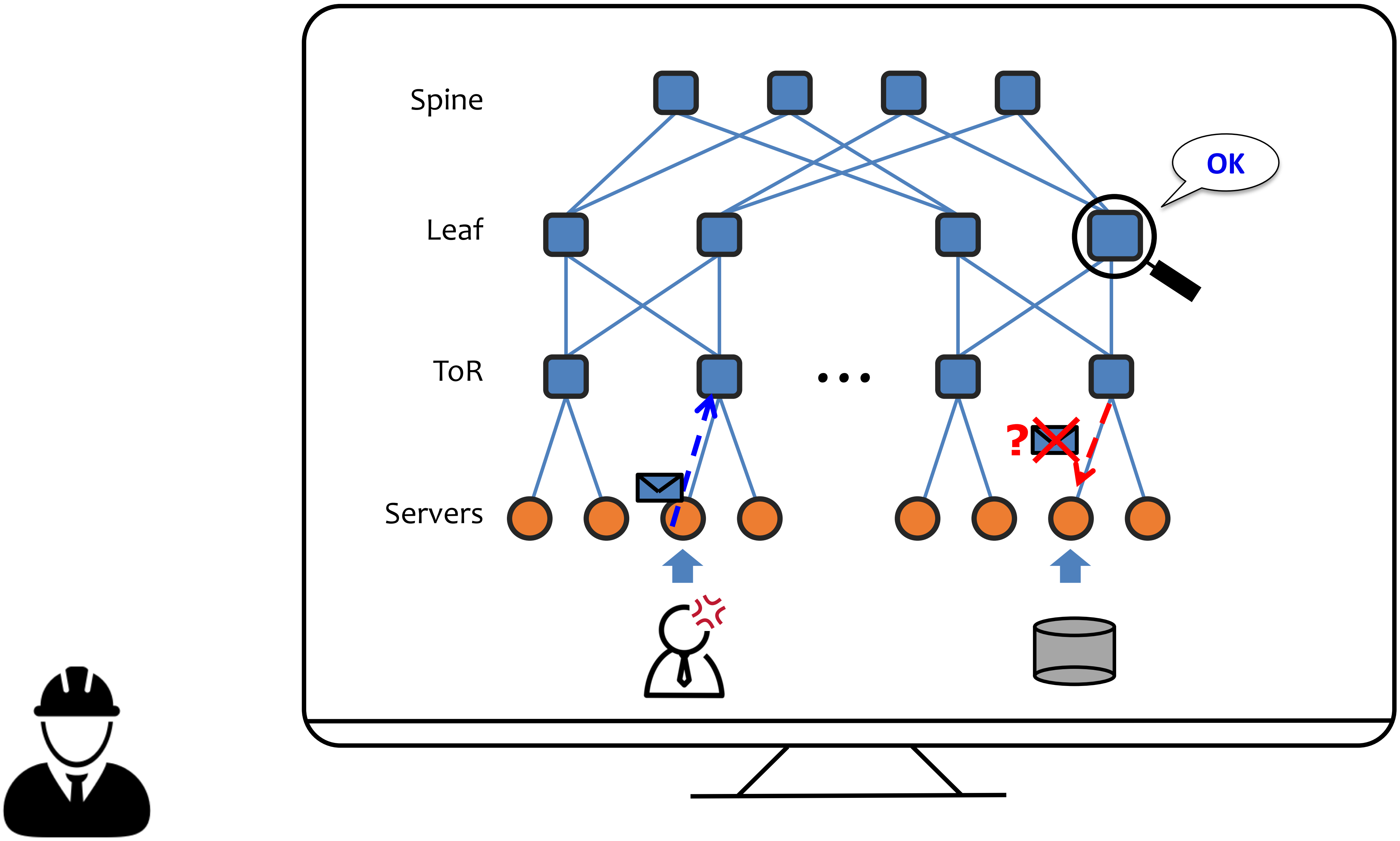
Anna
Network operator

Customers

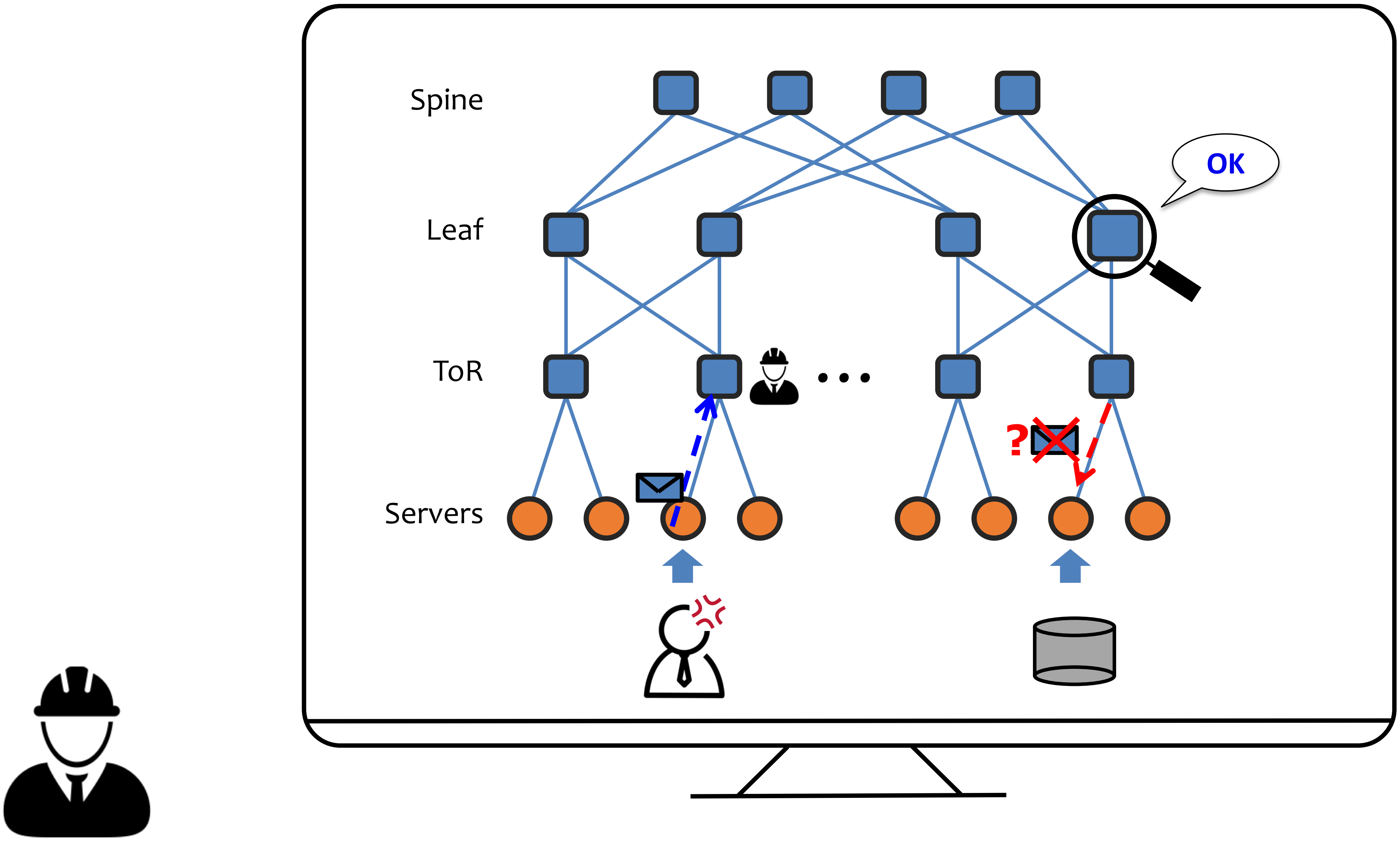

Spine
Leaf
ToR
Servers
?

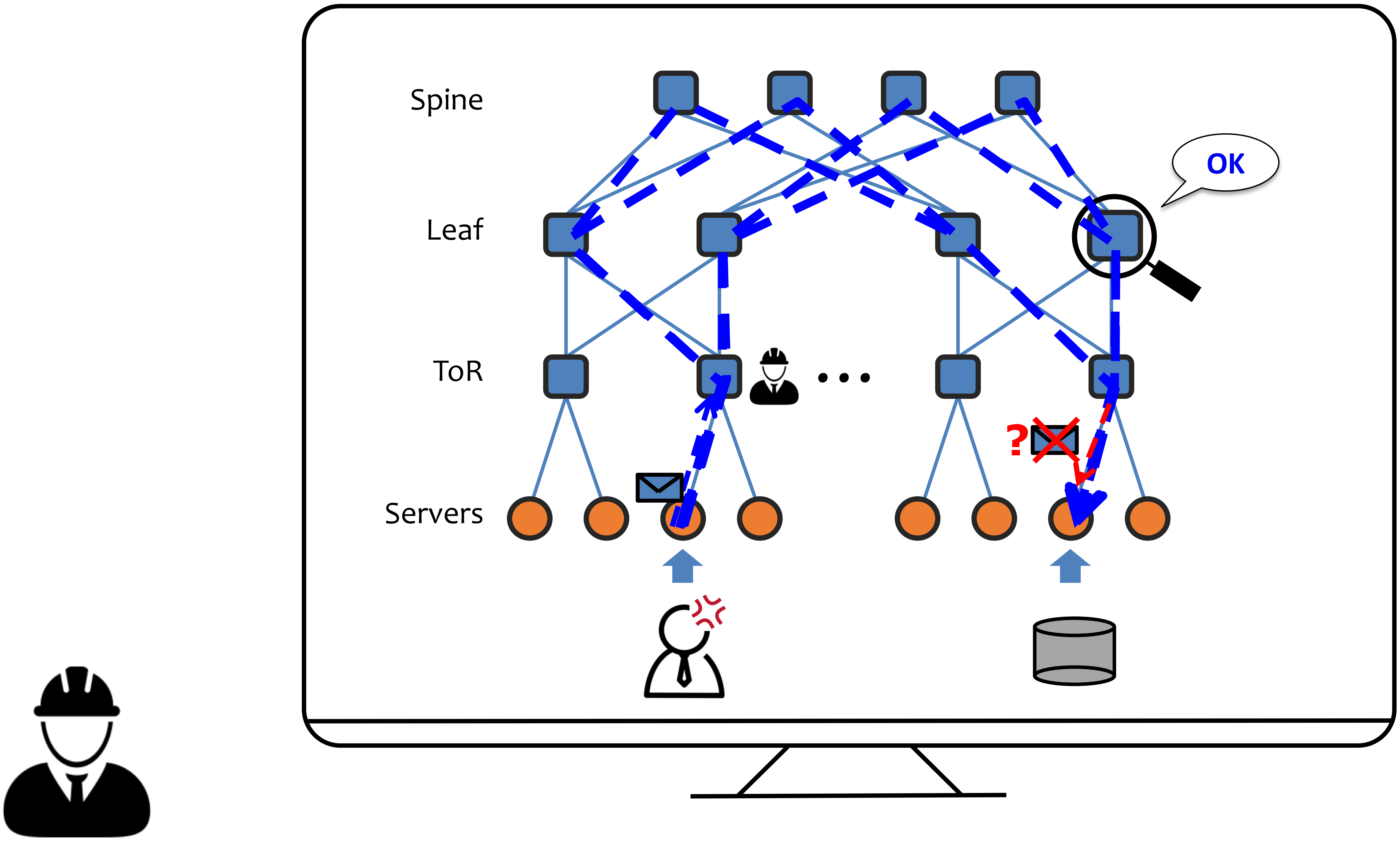

Traditional monitoring system
queries switches (e.g., SNMP)

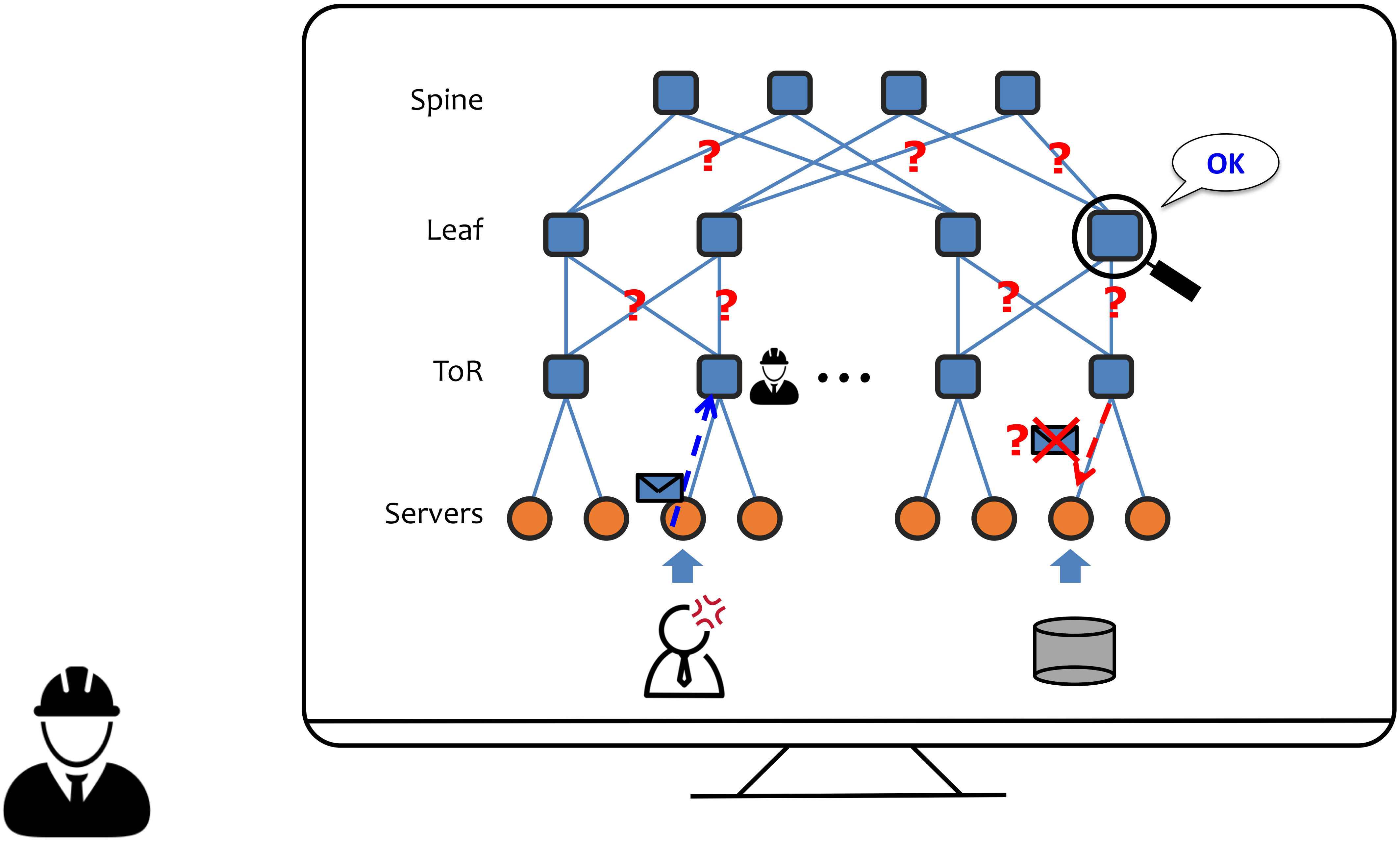

Spine
Leaf
ToR
Servers
OK

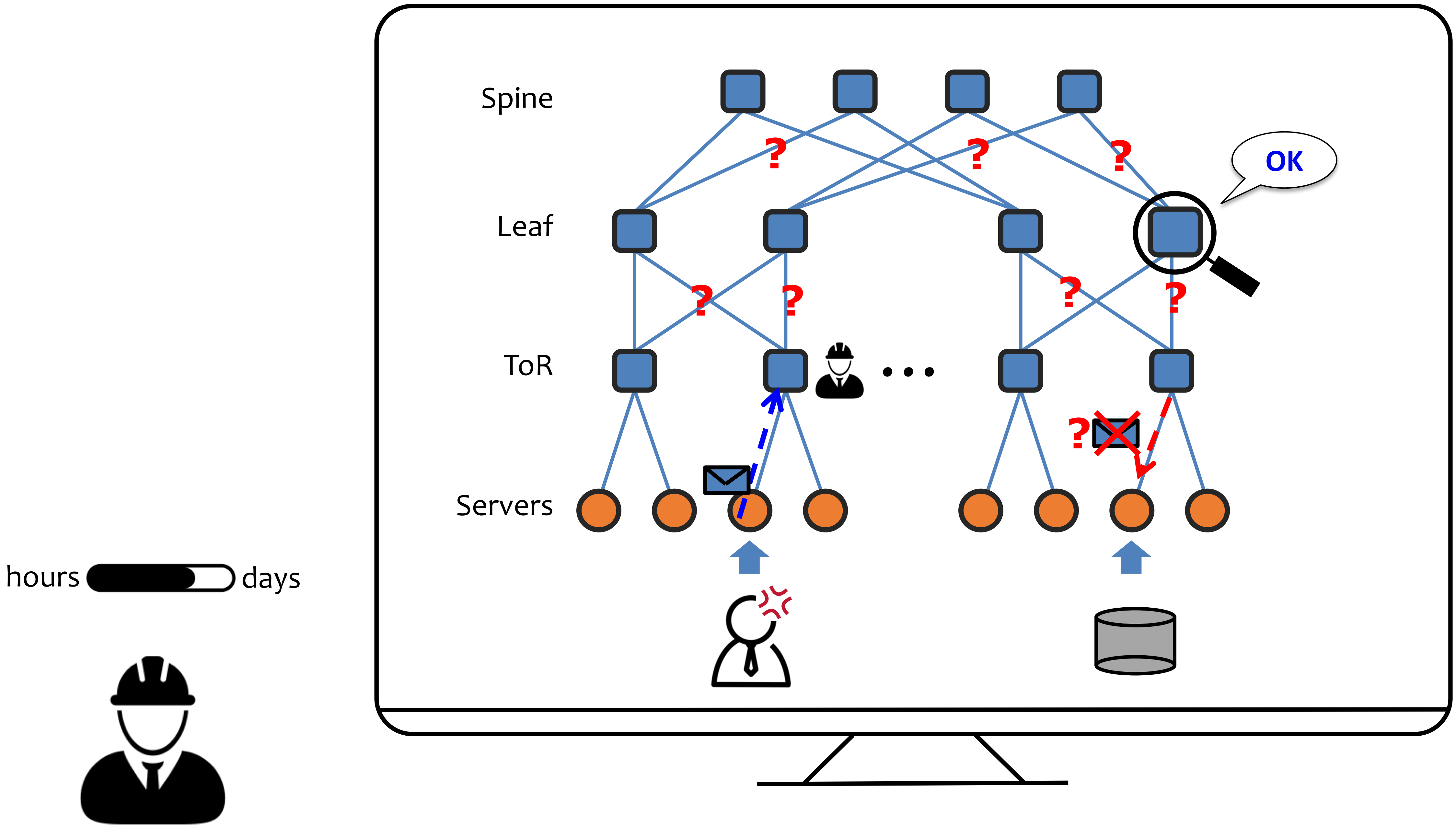


Spine
Leaf
ToR
Servers
OK
?


Spine
Leaf
ToR
Servers
OK
?


Spine
Leaf
ToR
Servers
OK
?

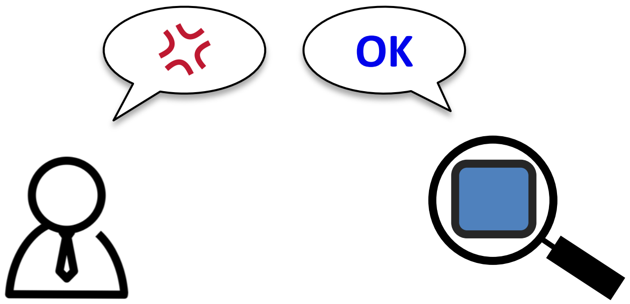

Spine
?
?
?
OK
Leaf
?
?
?
?
ToR
Servers
?


Spine
Leaf
ToR
Servers
OK
hours
days

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



*Huang et al. Gray Failure: The Achilles' Heel of Cloud-Scale Systems. HotOS'17

Why yet another monitoring system?

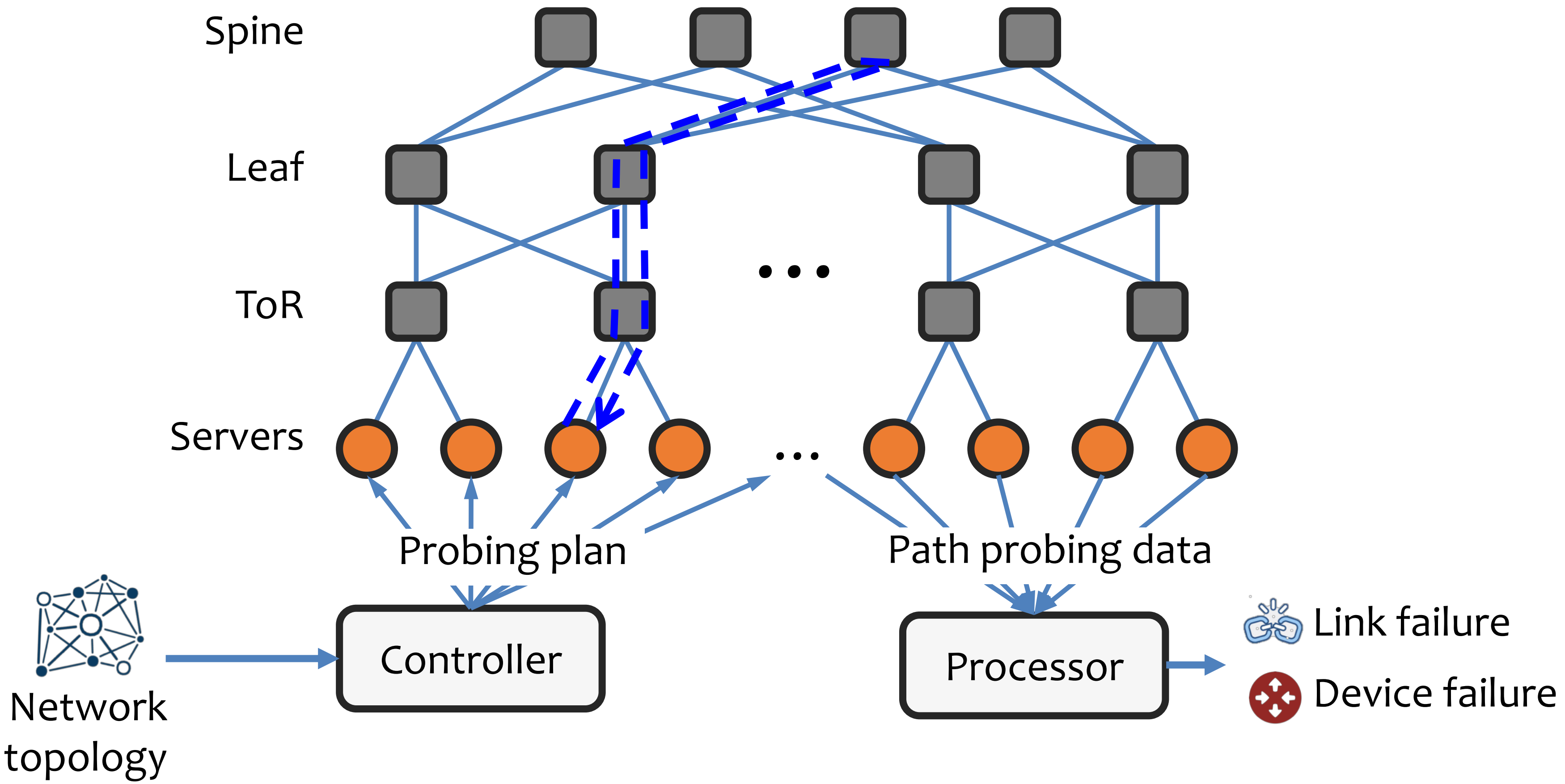
- Our response to *network gray failures* is *NetBouncer*

- Indeed, many monitoring systems
  - Academia: LossRadar, Trumpet, deTector, Netscope, …
  - Industry: Pingmesh, NetNORAD, 007, Passive probing, …

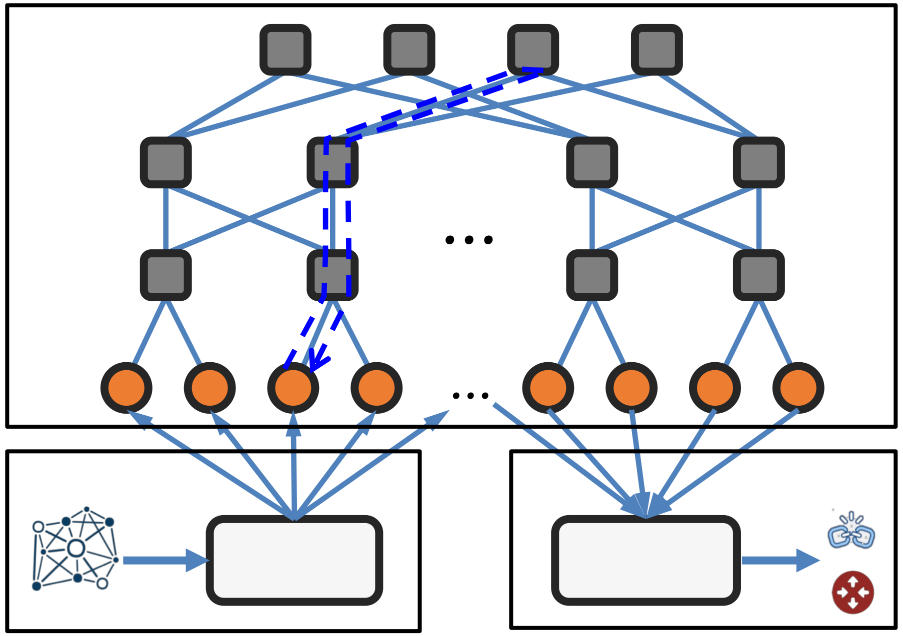
- In production, there are four requirements:
  1. Catch gray failures---from a server's perspective
  2. Transparent to current software stack
  3. Pinpoint failures in links or devices
  4. Few false positives (i.e., misreporting) and false negatives

NetBouncer overview

NetBouncer is an active probing system which **infers failures** from path probing data



Rest of the talk

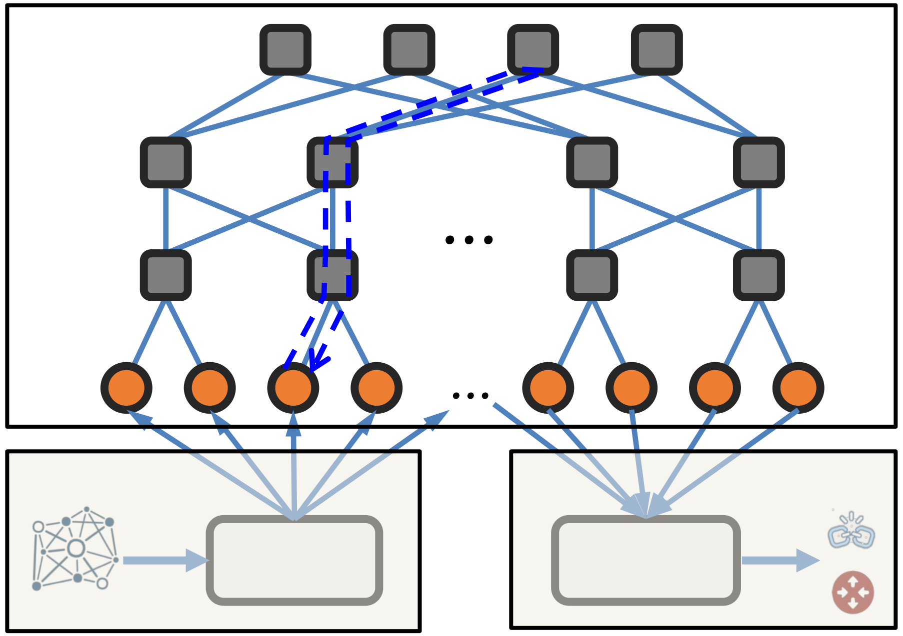


1 How to achieve light-weight and explicit probing?

2 Which paths should be probed?

3 How to infer failures from path probing data?

Rest of the talk



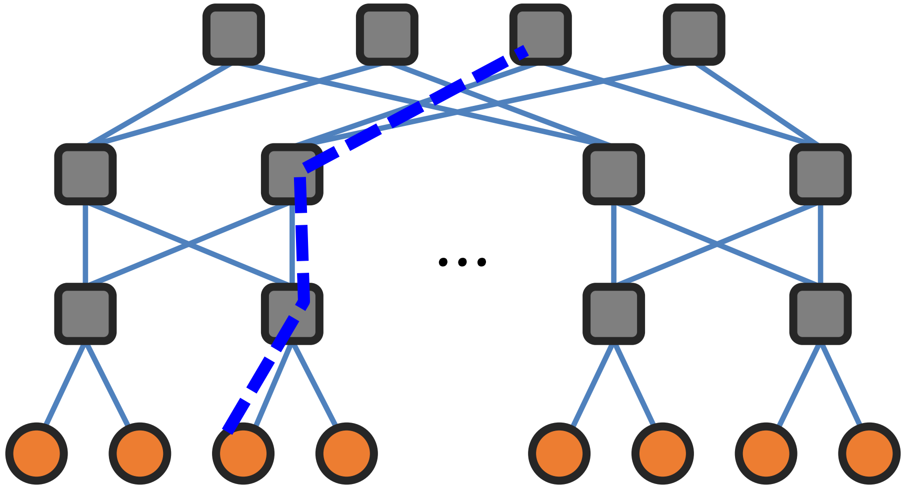
① How to achieve light-weight and explicit probing?

② How to design an eligible probing plan?

③ How to infer failures from path probing data?

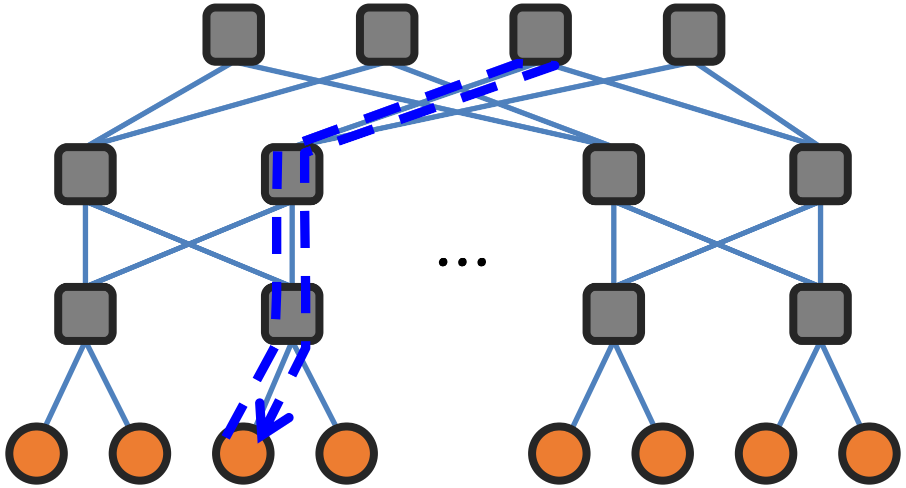
Active probing system requires explicit and efficient probing

- 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.

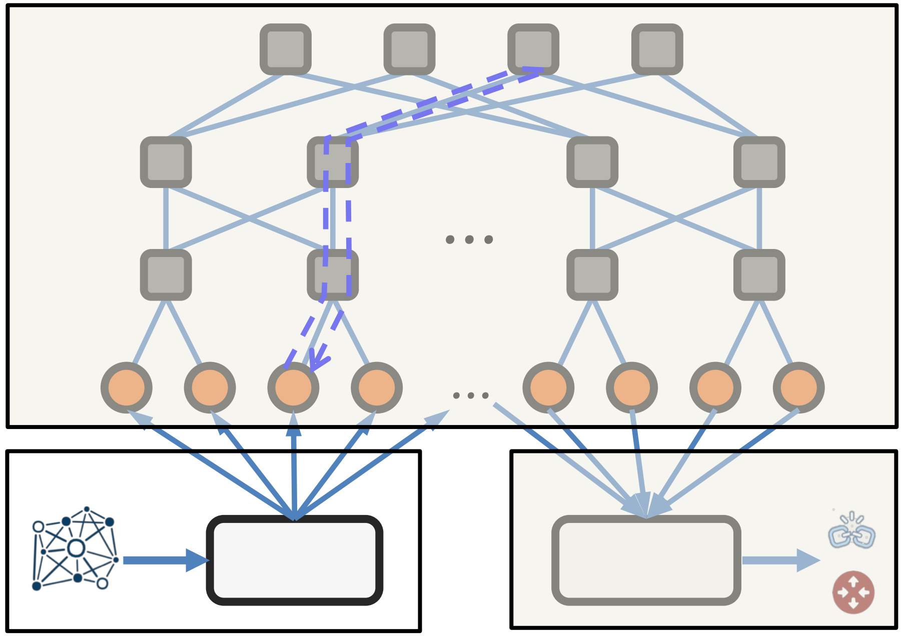


Active probing system requires explicit and efficient probing

- 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

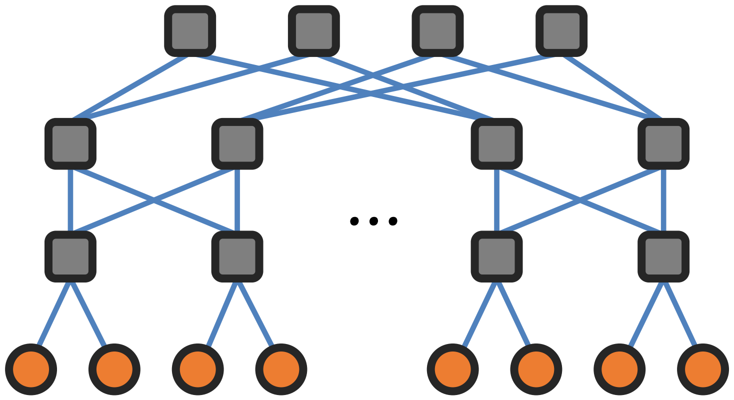


1 How to achieve light-weight and explicit probing?

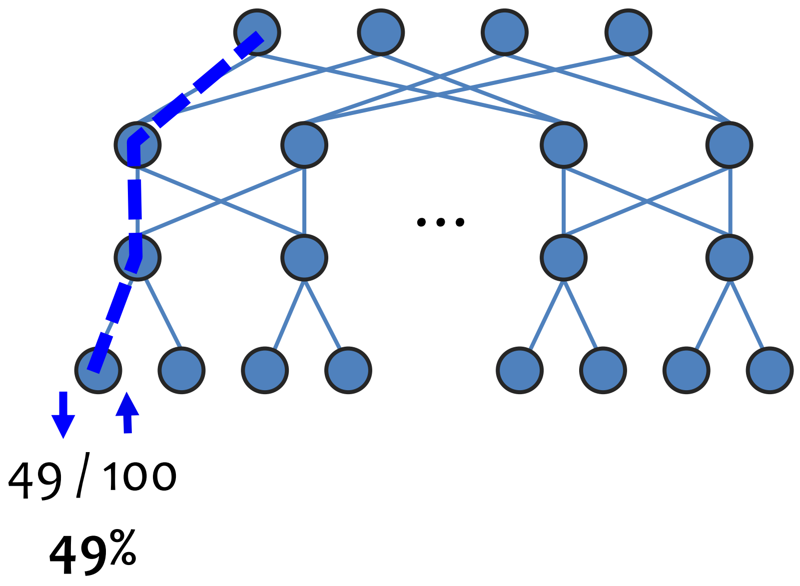
2 Which paths should be probed?

3 How to infer failures from path probing data?

Observation vs. inference: from path probing to failures

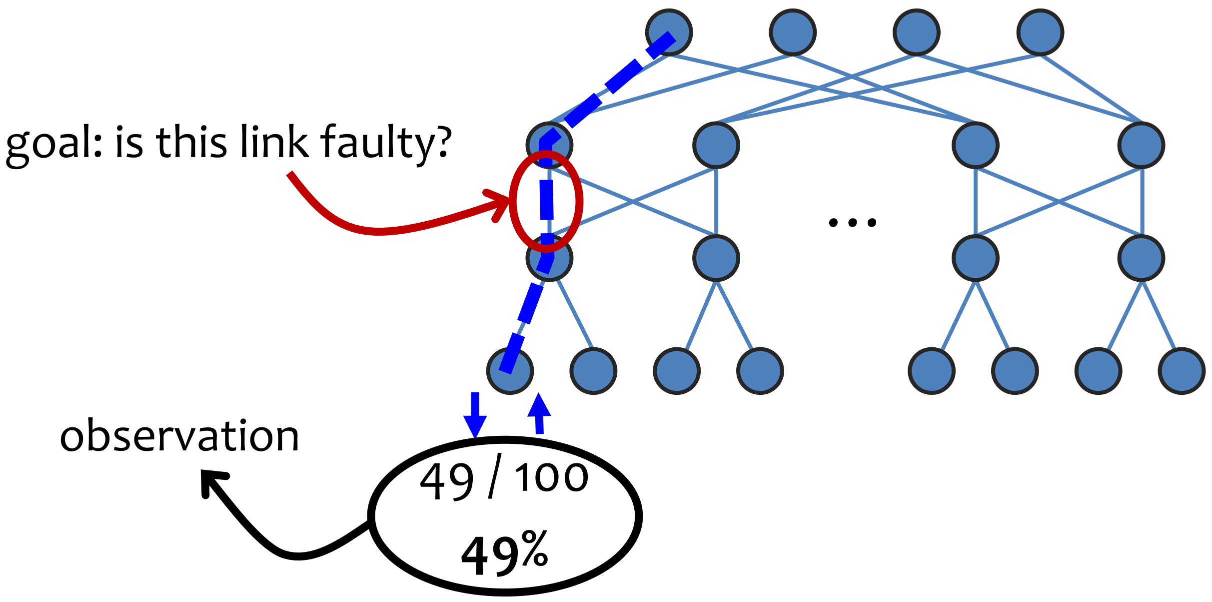


Observation vs. inference: from path probing to failures



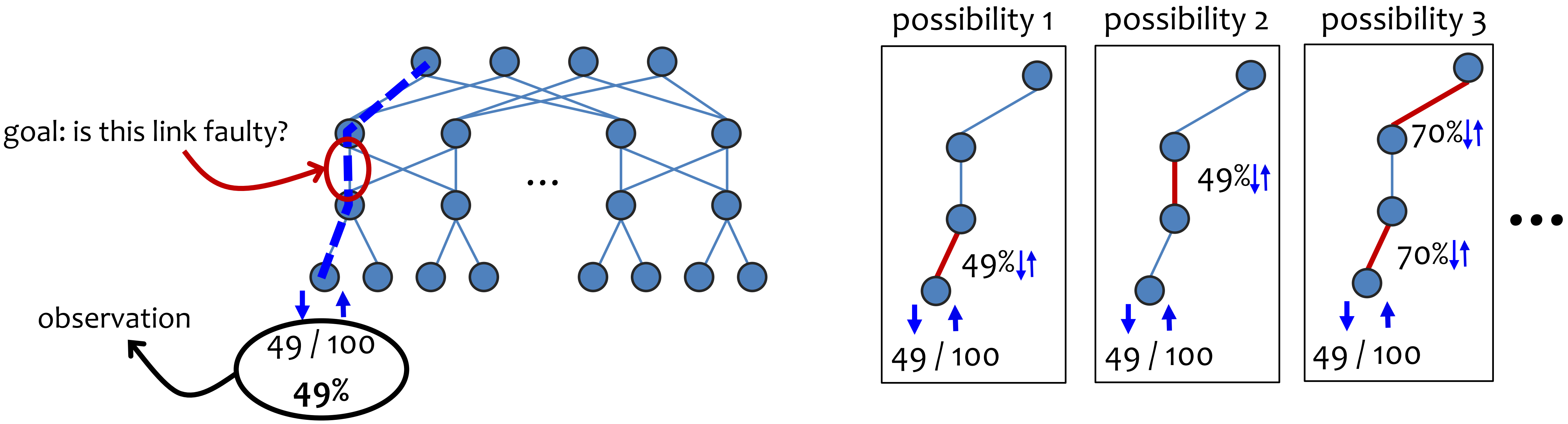
- Undirected graph (vertex=device, edge=link)
- Failures are probabilistic

Observation vs. inference: from path probing to failures



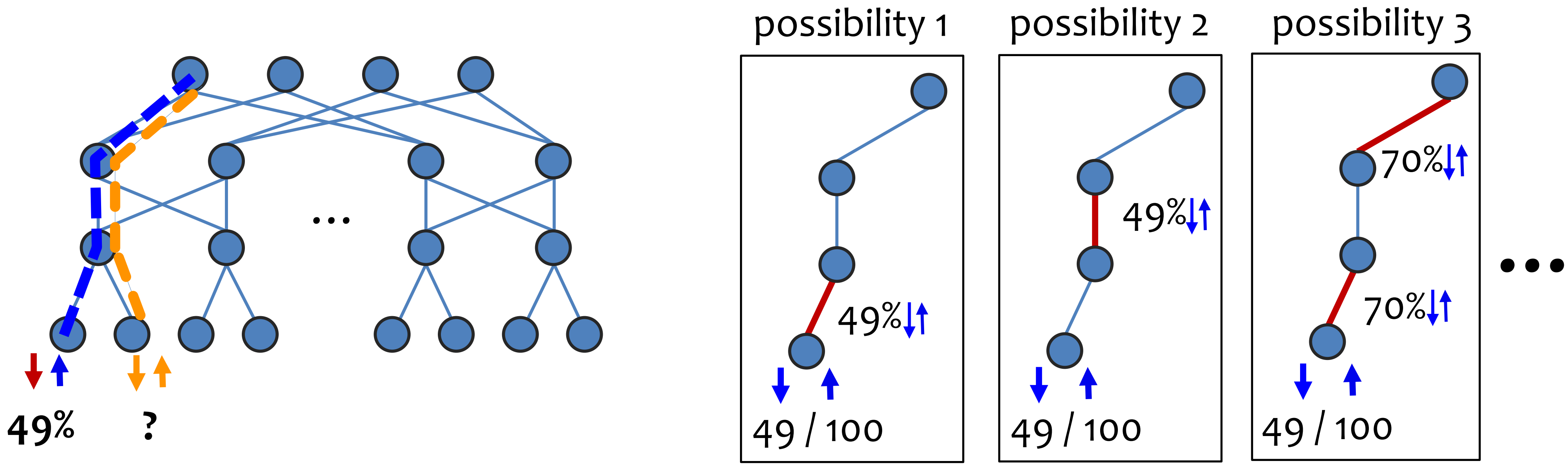
- Undirected graph (vertex=device, edge=link)
- Failures are probabilistic

Observation vs. inference: from path probing to failures

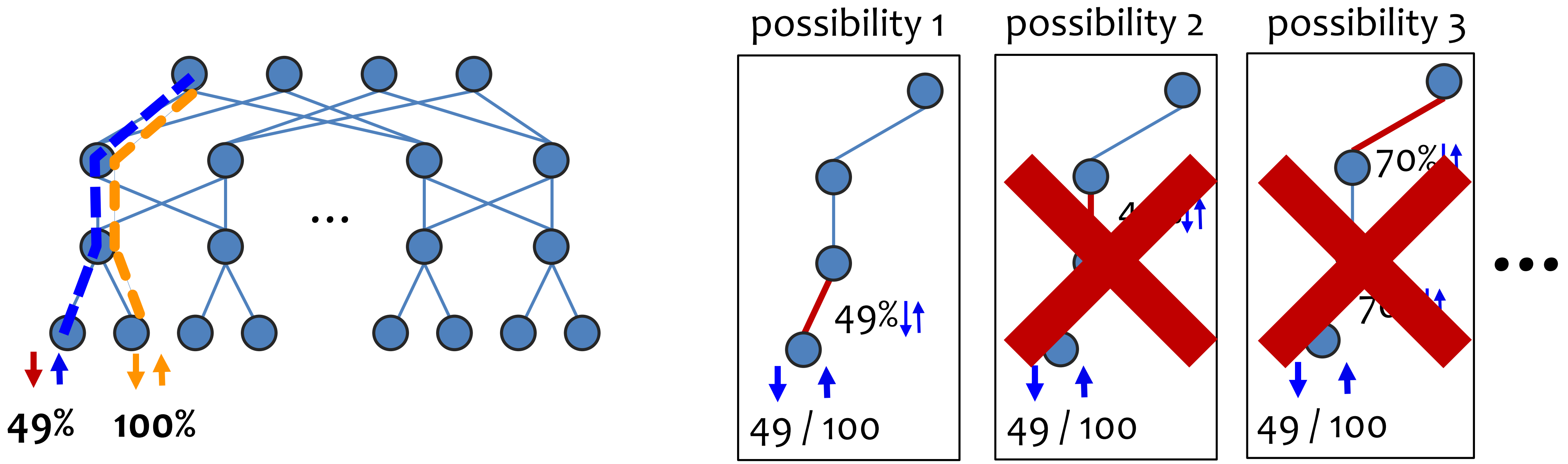


- Undirected graph (vertex=device, edge=link)
- Failures are probabilistic

Observation vs. inference: from path probing to failures

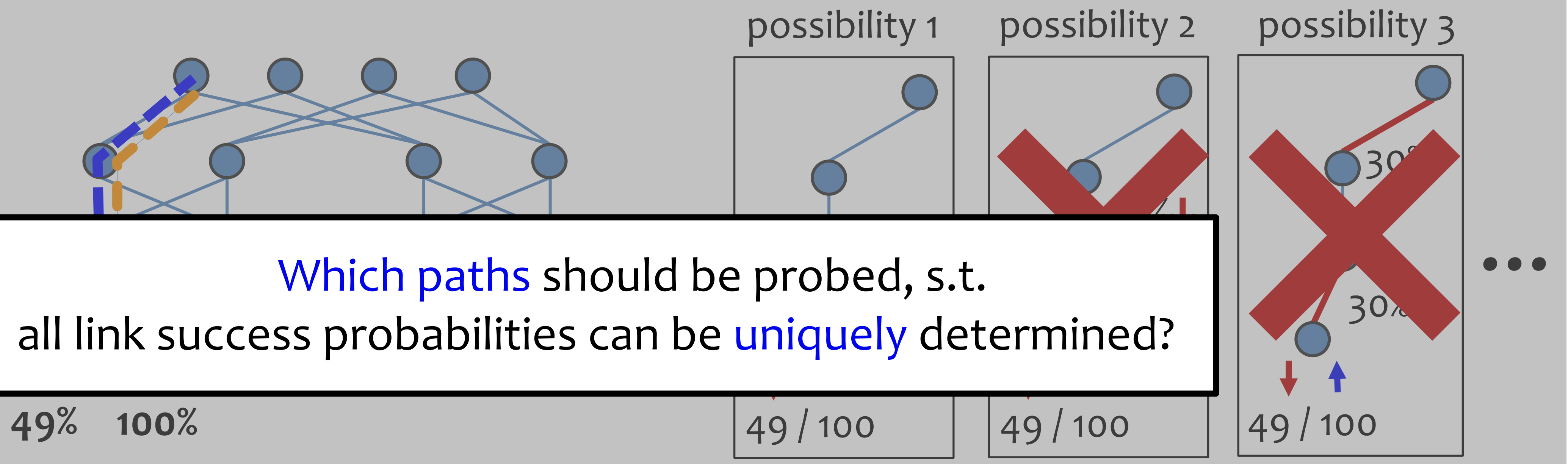


Observation vs. inference: from path probing to failures



- 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



Which paths should be probed, s.t.
all link success probabilities can be uniquely determined?

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

Real-world constraints complicate path selection

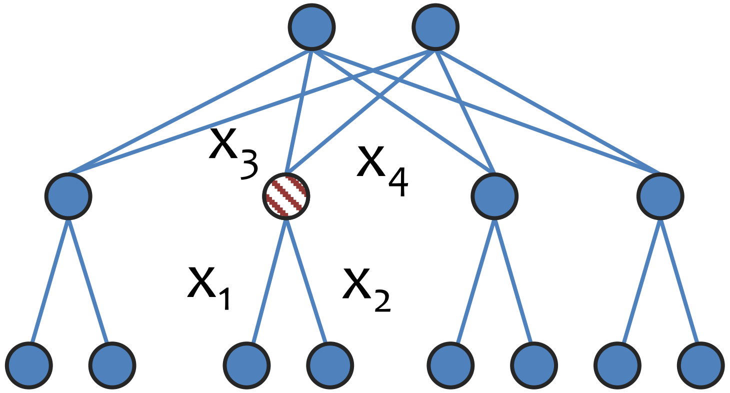
- Constraint 1: some switches may not bounce the probing
- Constraint 2: a probing path starts/ends at the same server

- 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

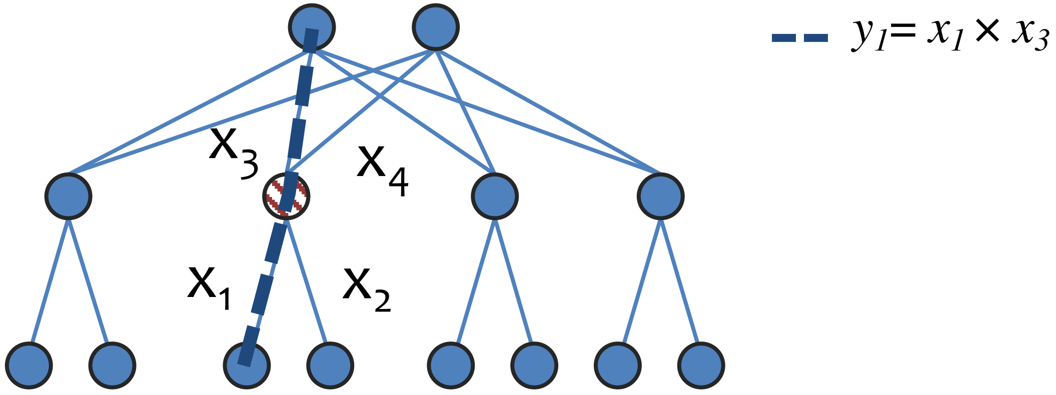
- Sometimes, it is impossible to uniquely identify all links



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

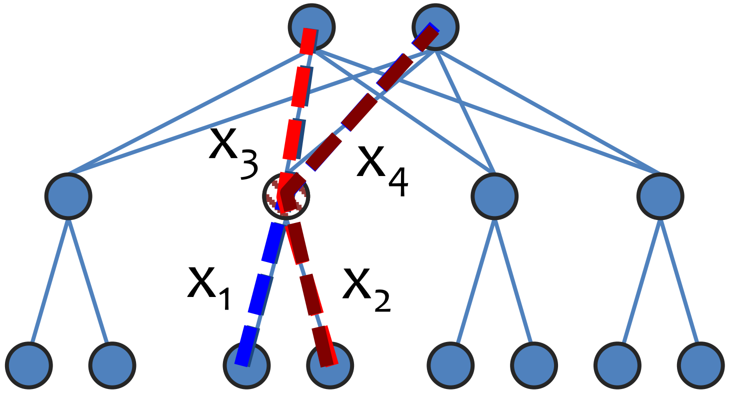
- Sometimes, it is impossible to uniquely identify all links



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



$y_1 = x_1 \times x_3$

$y_2 = x_1 \times x_4$

$y_3 = x_2 \times x_3$

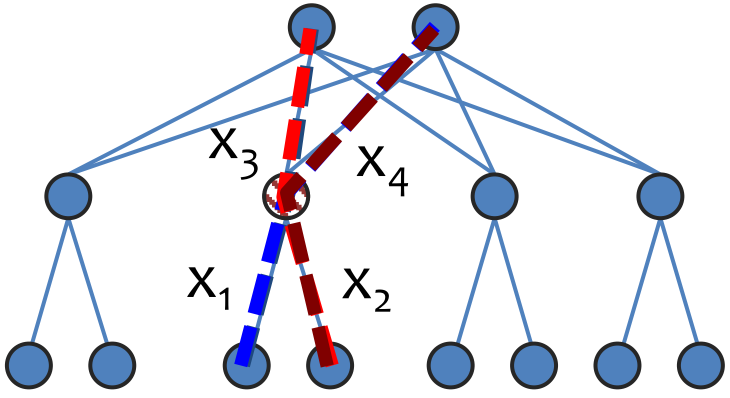
$y_4 = x_2 \times x_4$

$$y_1 \times y_4 = y_2 \times y_3$$

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



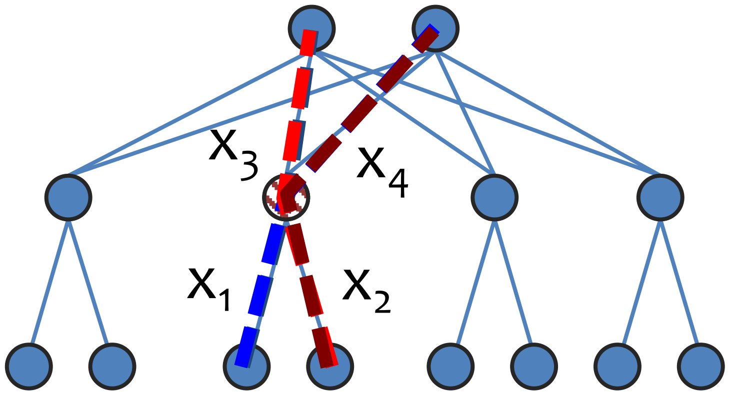
$$\log(y_1) = \log(x_1)+\log(x_3)$$
$$\log(y_2) = \log(x_1)+\log(x_4)$$
$$\log(y_3) = \log(x_2)+\log(x_3)$$
$$\log(y_4) = \log(x_2)+\log(x_4)$$

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



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

Not full rank

Links success probabilities (x_1-x_4) can be arbitrary

A condition to uniquely identify link success probabilities

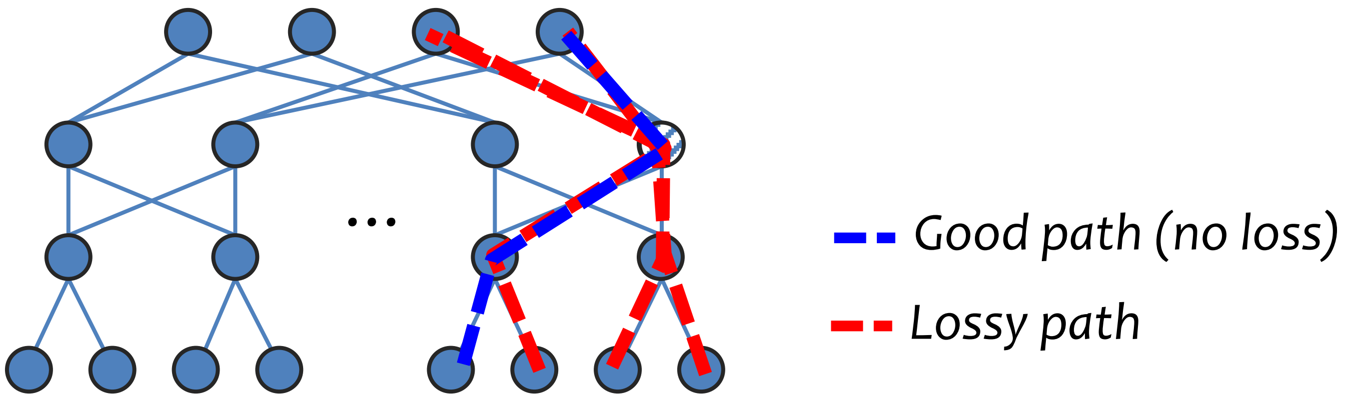
We proved a theorem (*for Clos network*), that provides

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

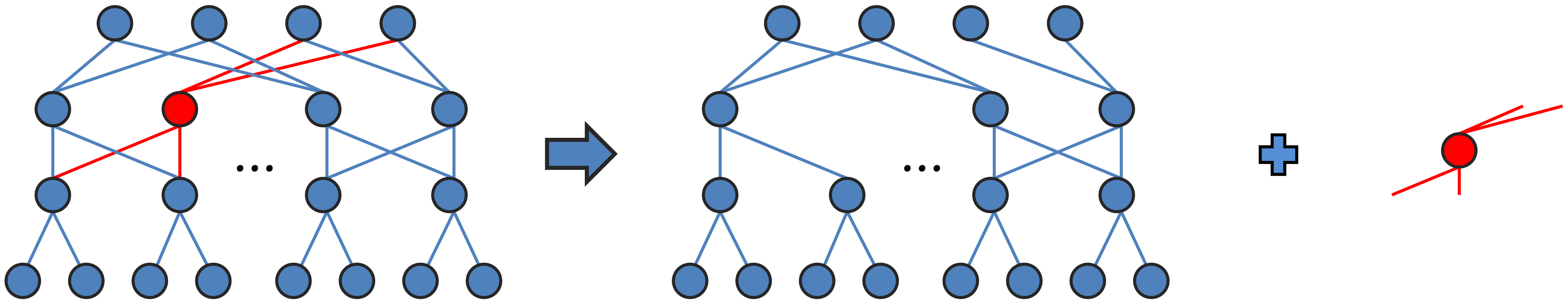
A condition to uniquely identify link success probabilities

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



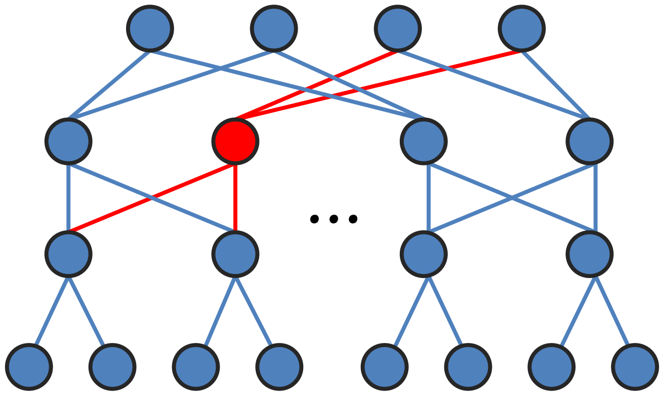
Original graph

Subgraph with unique solution

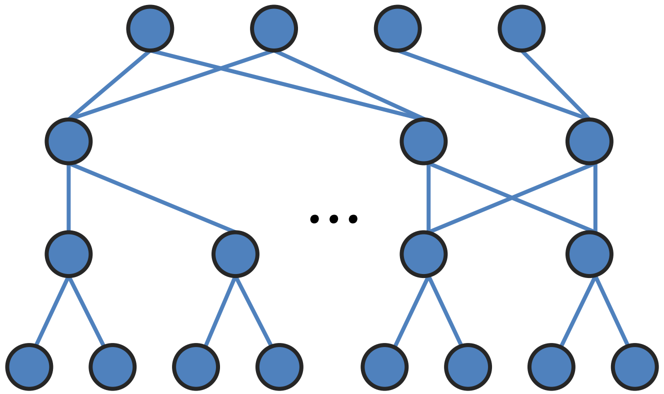
Unsolvable part

No good paths pass this switch

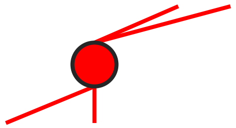
Device failure detection



Original graph



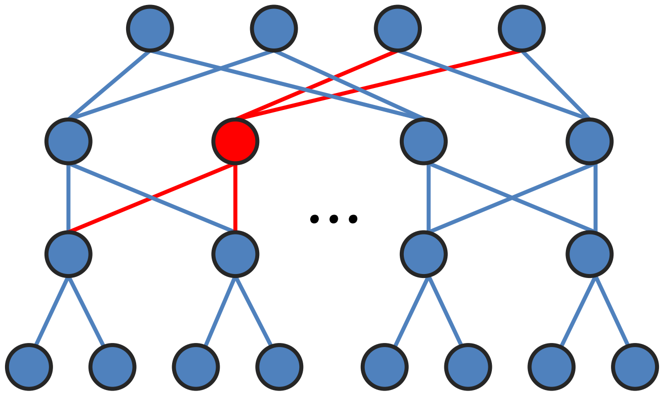
Subgraph with unique solution



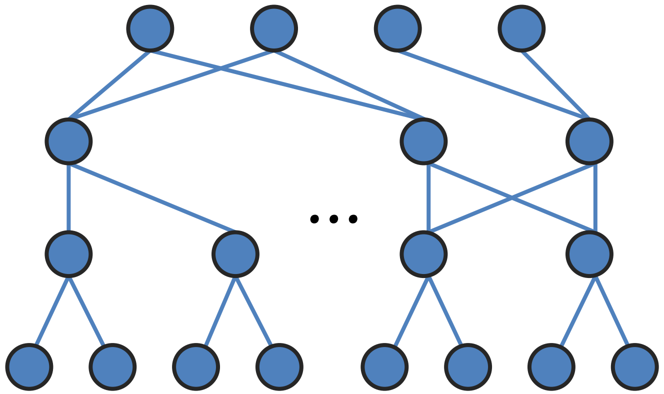
Faulty devices

No good paths pass this switch

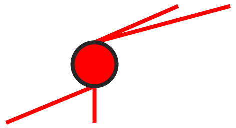
Device failure detection



Original graph

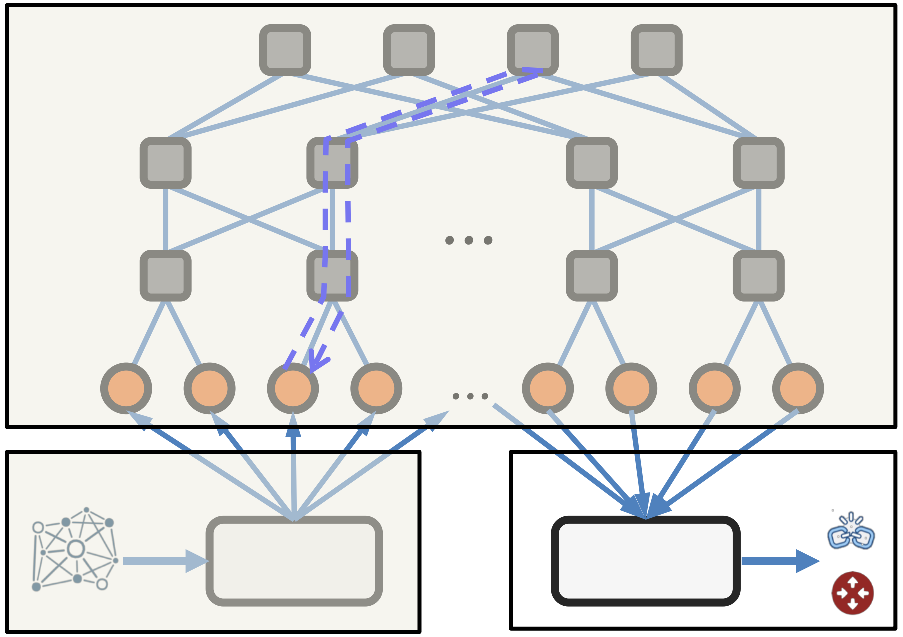


How to infer the link failures
from this subgraph?



Faulty devices

No good paths pass
this switch

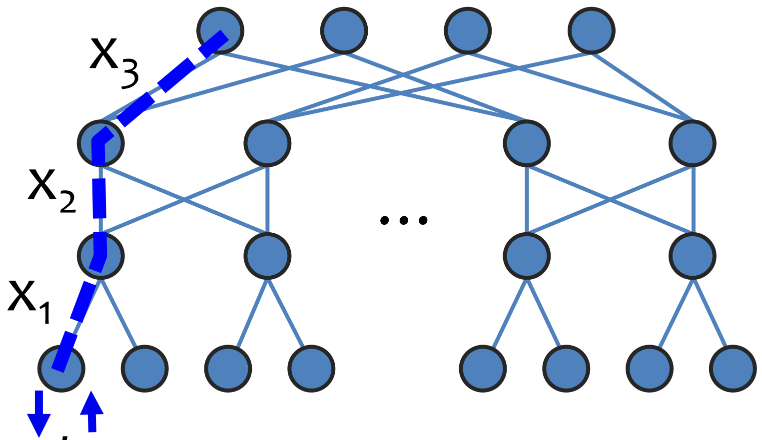


1 How to achieve light-weight and explicit probing?

2 Which paths should be probed?

3 How to infer the link failures from the solvable subgraph?

Link failure inference: an optimization problem



$y_1 = 50/100 = x_1 \times x_2 \times x_3$

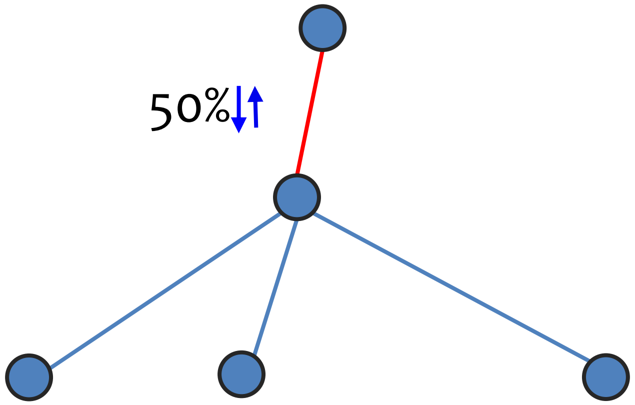
Assume packet drops are independent events.

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

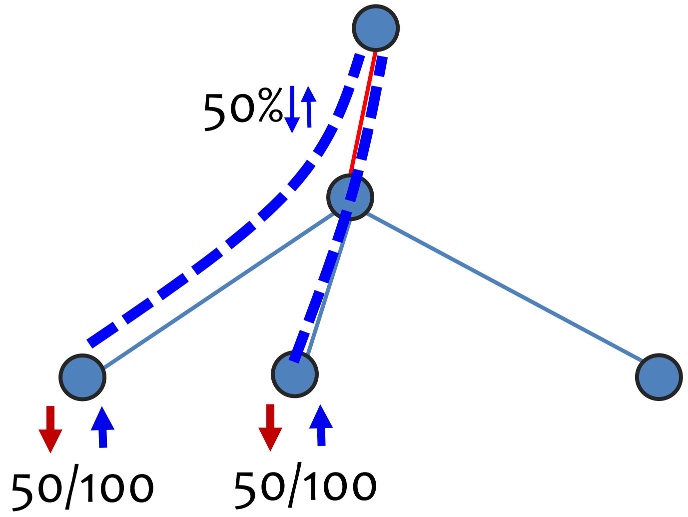
$$\text{minimize} \quad \sum_j \Big(y_j - \prod_{i:\text{link}_i \in \text{path}_j} x_i\Big)^2$$

$$\text{subject to} \quad 0 \le x_i \le 1, \forall i$$

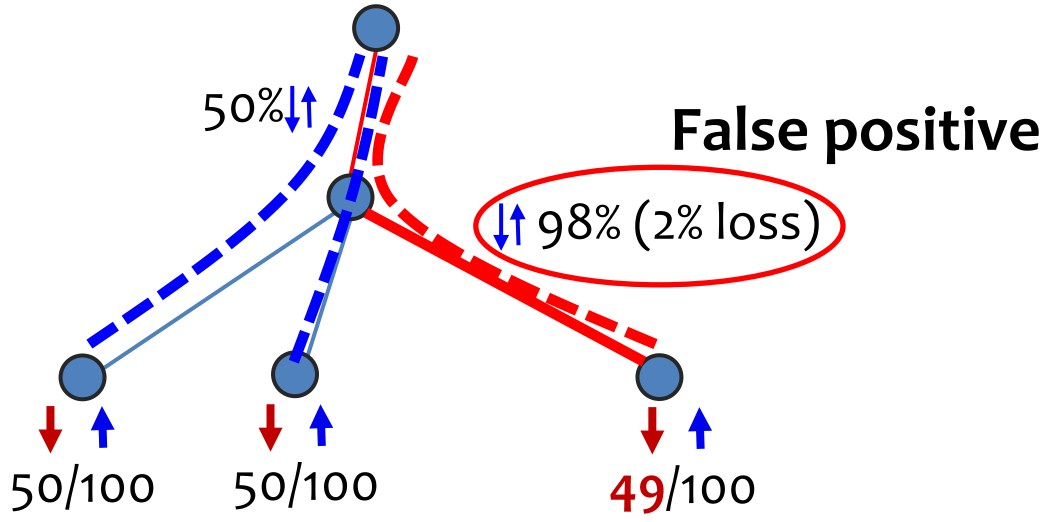
Real-world data inconsistency induces false positives



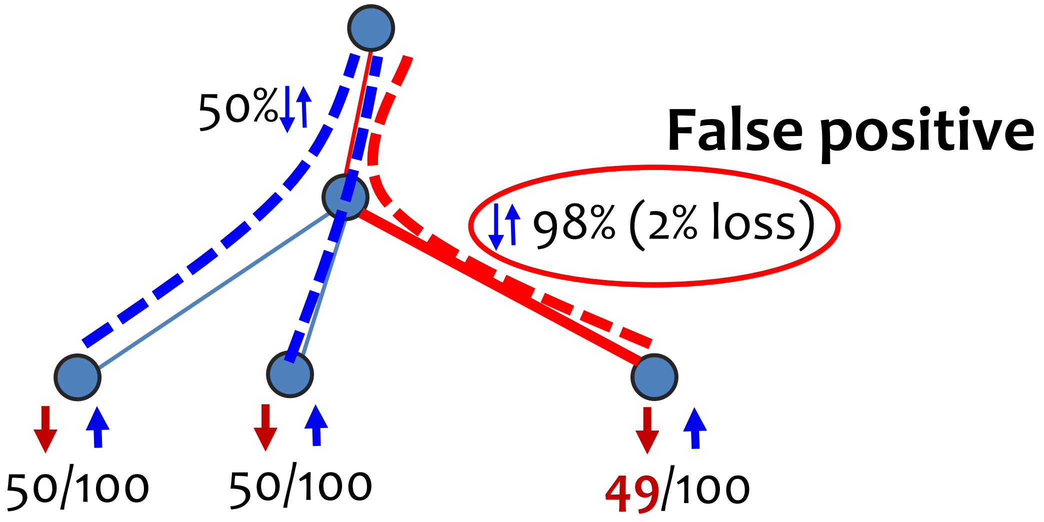
Real-world data inconsistency induces false positives



Real-world data inconsistency induces false positives

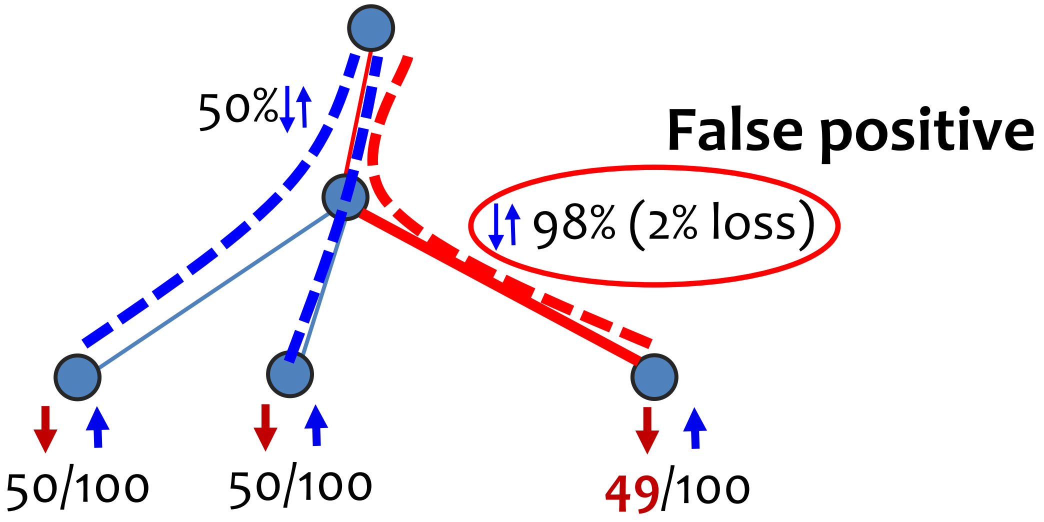


Real-world data inconsistency induces false positives



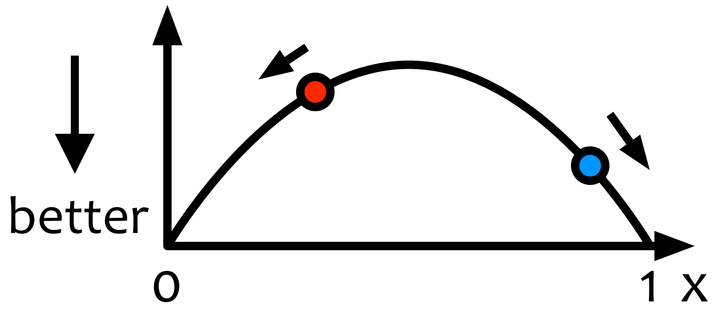
- Real-world data inconsistency
  - Measurements do not fully align
  - Inference results may overfit observations

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

$$\sum_j (y_j - \prod_{i:\text{link}_i \in \text{path}_j} x_i)^2 + \lambda \sum_i x_i(1-x_i)$$



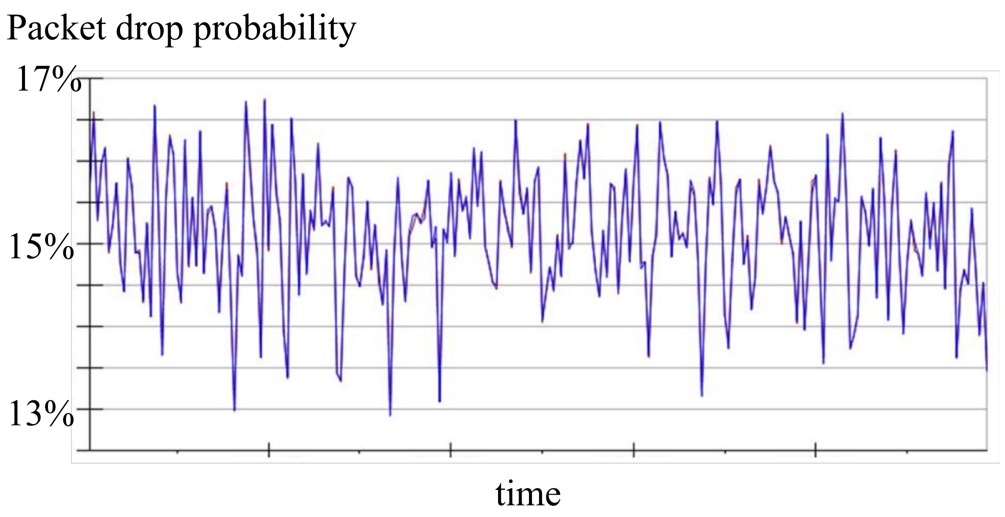
Evaluation questions

- 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

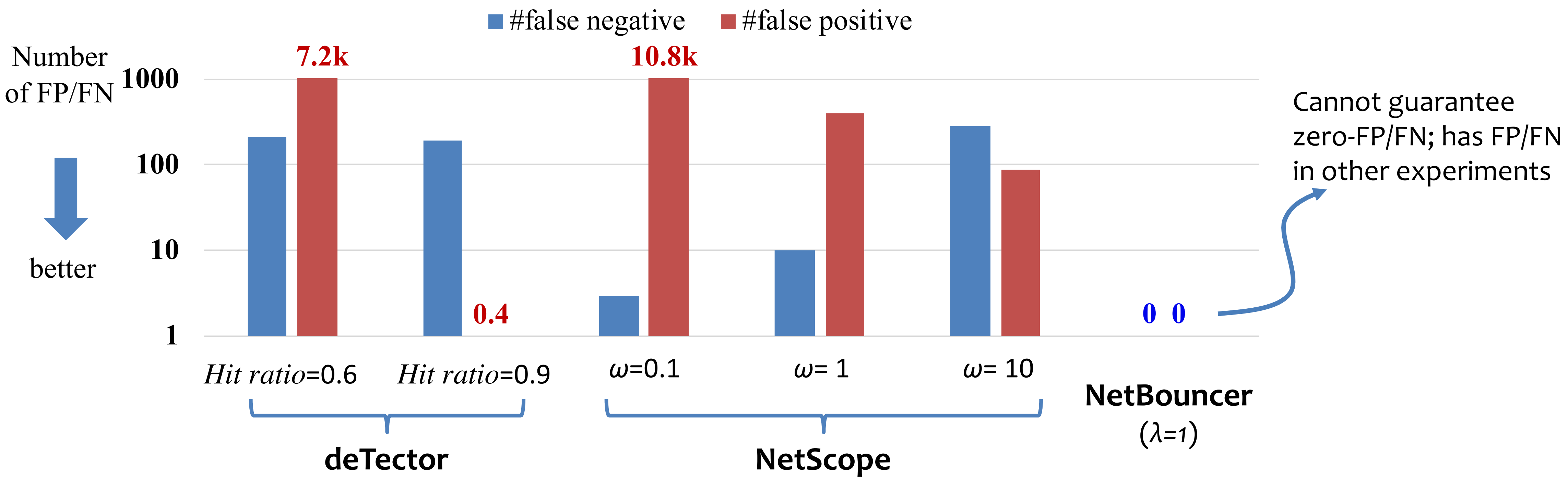
- Observations
  - Many customers experienced packet drops and latency increases
  - Traditional monitoring systems cannot pinpoint the failure

- NetBouncer detected this gray failure
  - One spine router silently dropped packets
  - Root cause was an issue in one of this switch's linecard hardware



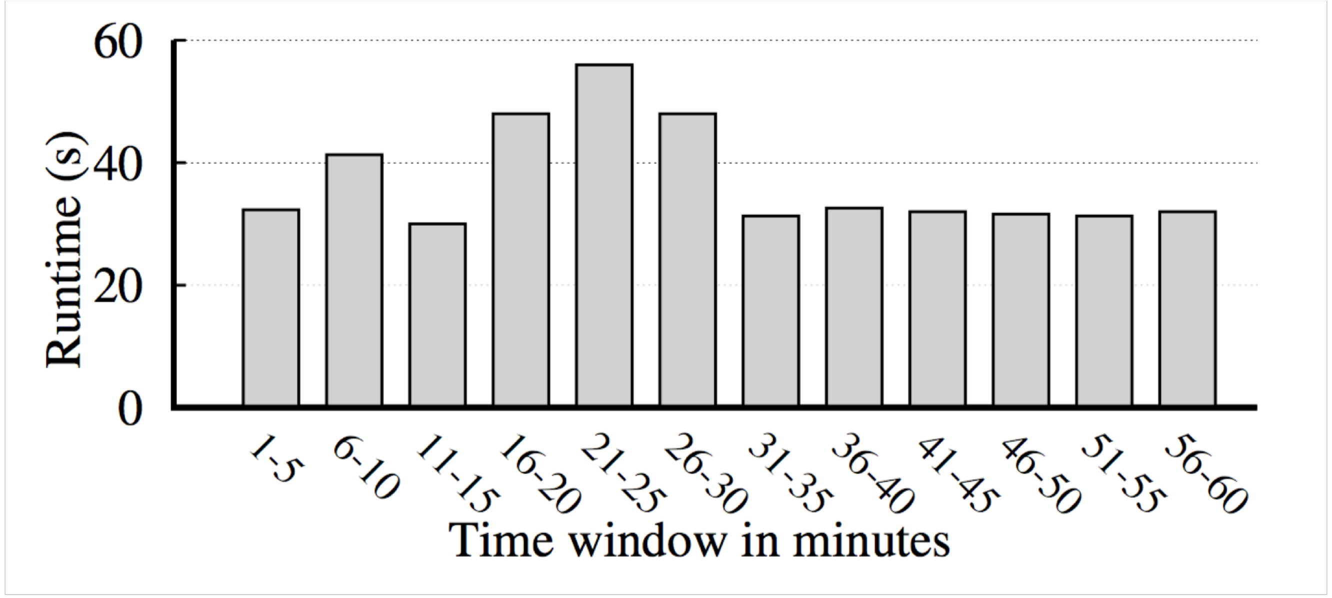
Accuracy comparison with previous algorithms

- 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*



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