Democratically Finding The Cause of Packet Drops

Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang (Harry) Liu, Jitu Padhye, Geoff Outhred, Boon Thau Loo

Marple- SigComm 2017

DeepView- NSDI 2018

Gestalt- ATC 2014

Sherlock- SigComm 2007

SNAP- NSDI 2011

In this talk I will show how to:
Find the cause of every *TCP* packet drop*

TRat- SigComm 2002

Roy et al- NSDI 2017

Netprofiler- P2Psys 2005

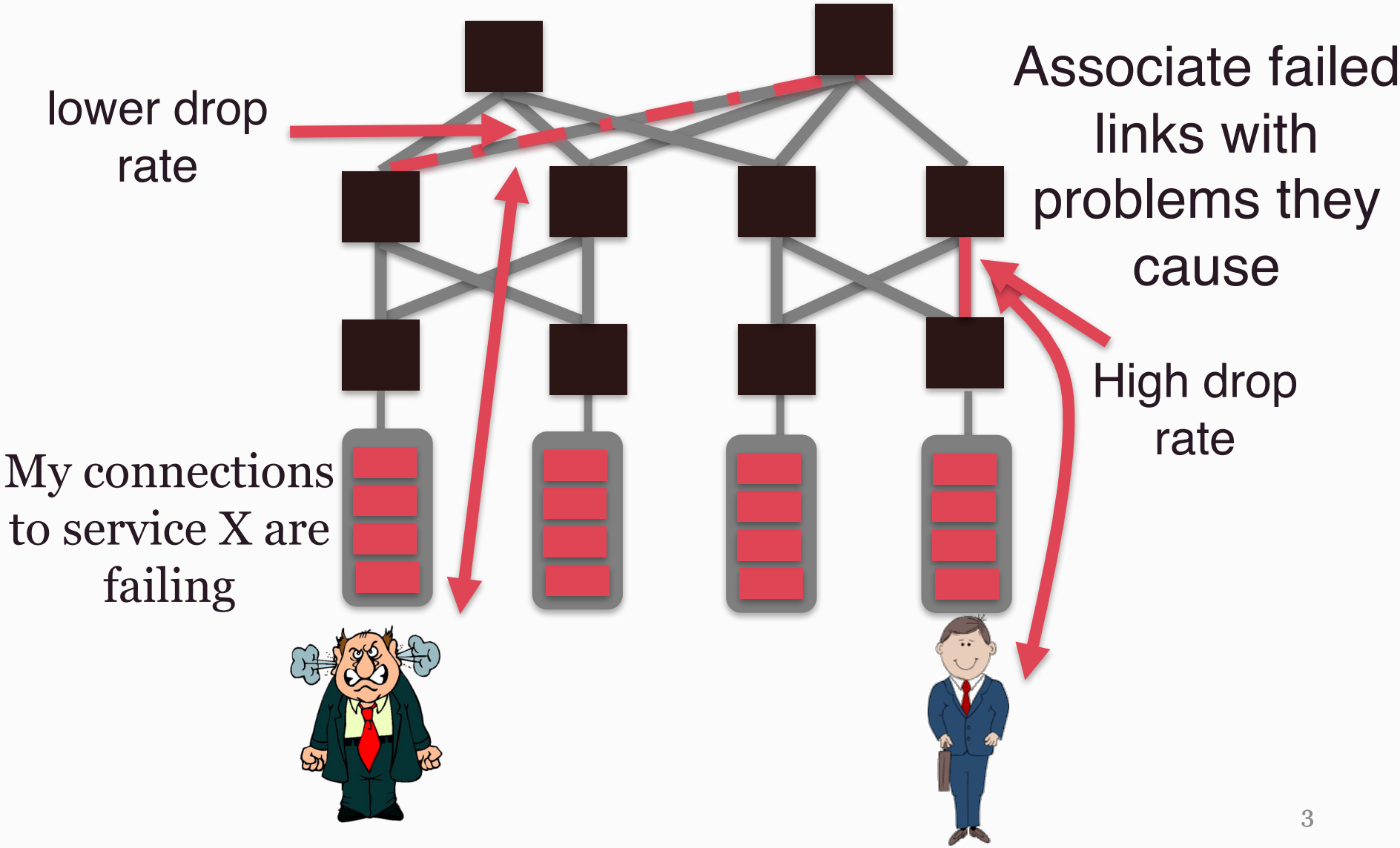
Pingmesh - SigC 2016

PathDump- OSDI 2015

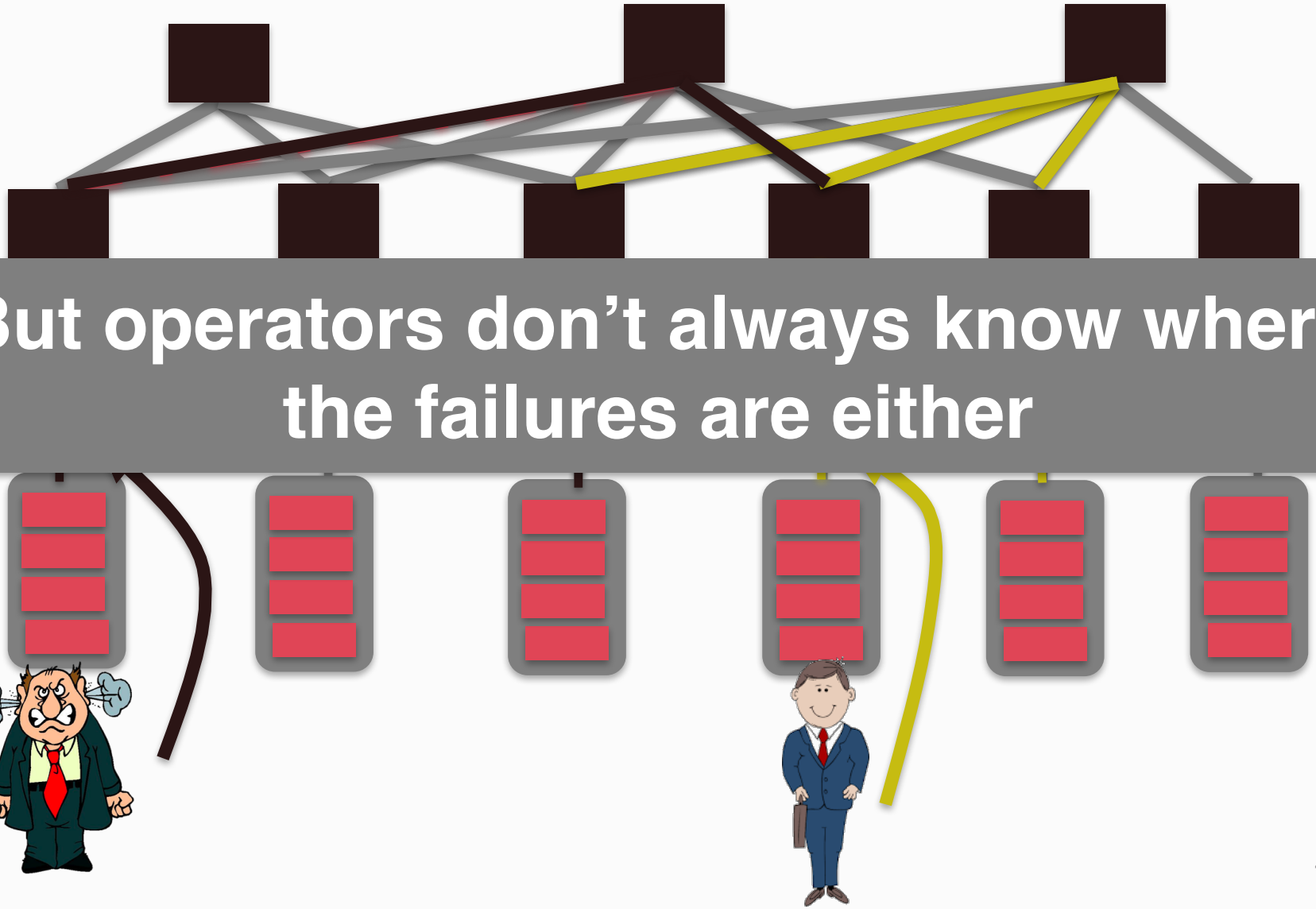
Netclinic- VAST 2010

*As long as it is not caused by noise

Not all faults are the same

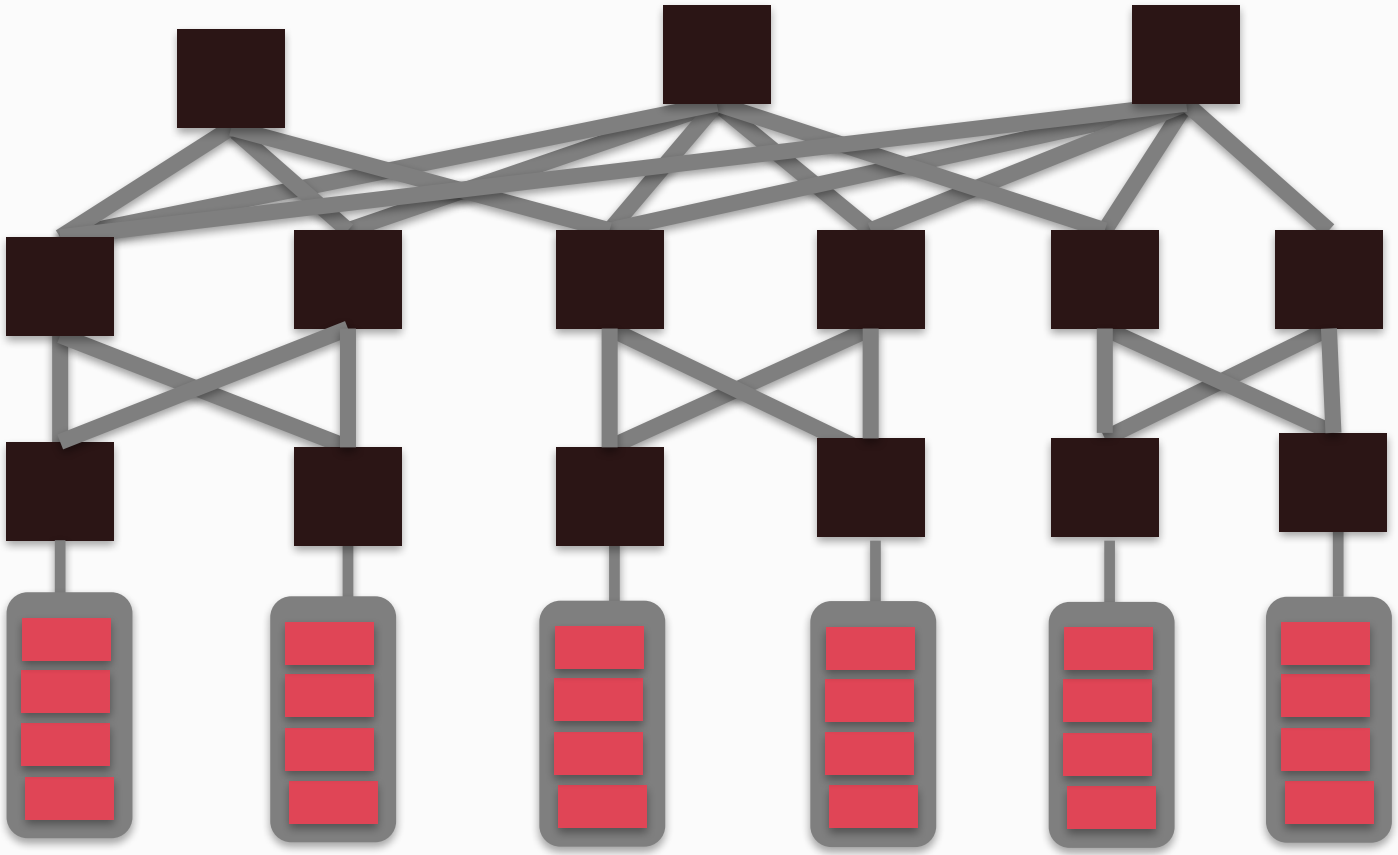


Mapping complaints to faulty links



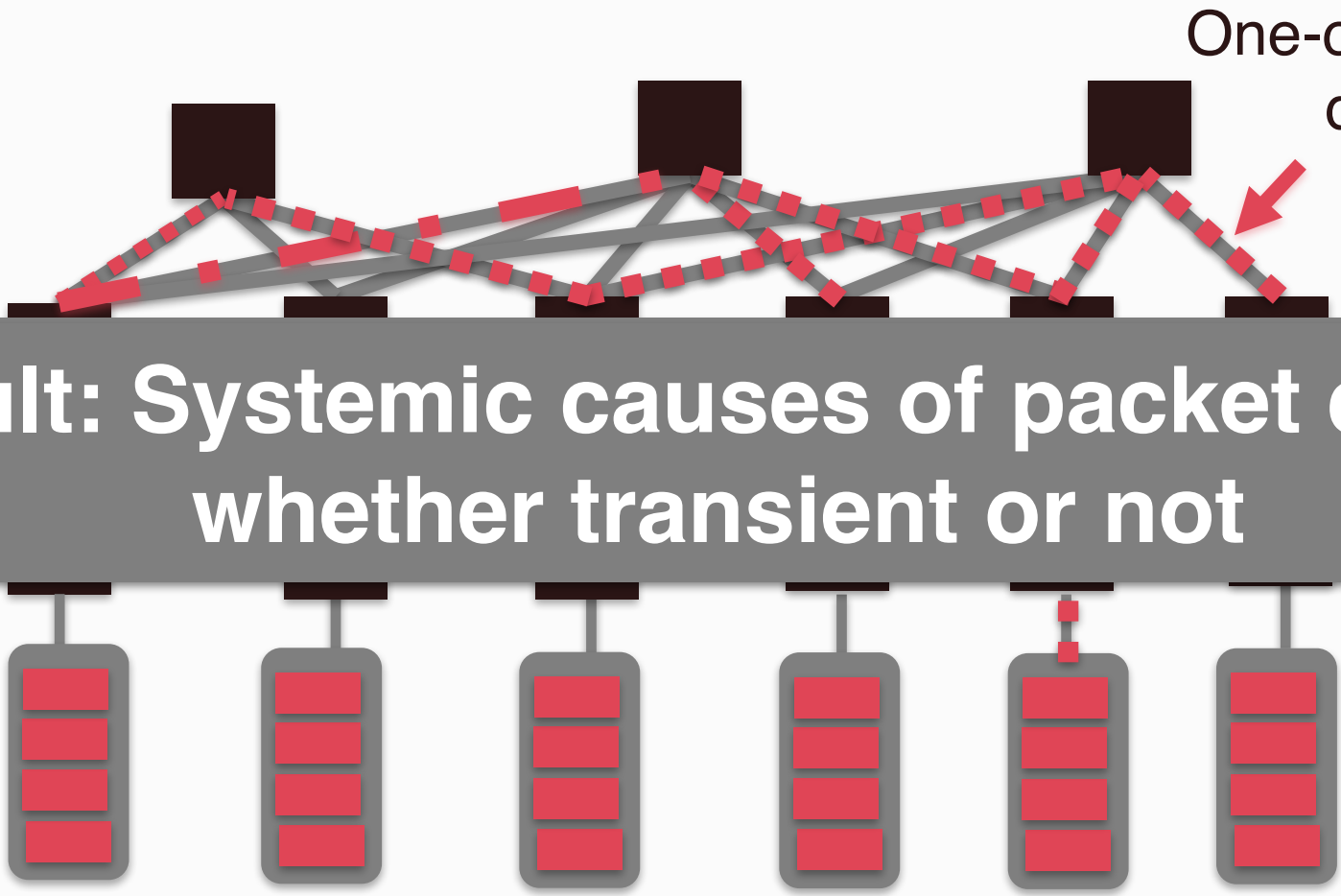
But operators don't always know where the failures are either

Clouds operate at massive scales



Each Data center has millions of devices

Low congestion drop rates add noise



Fault: Systemic causes of packet drops whether transient or not

* Z., Danyang, et al. "Understanding and mitigating packet corruption in data center networks."

Fault: Systemic causes of packet drops whether transient or not

Noise: One-off packet drop due to buffer overflows

Talk outline

- Solution requirements
- A strawman solution and why its impractical
- The 007 solution
  - Design
  - How it finds the cause of every TCP flow's drops
  - Theoretical guarantees
- Evaluation

Solution Requirements

- Detect short-lived failures
- Detect concurrent failures
- Robust to noise

Want to avoid infrastructure changes

- Costly to implement and maintain
- Sometimes not even an option
  - Example: changes to flow destinations (not in the DC)

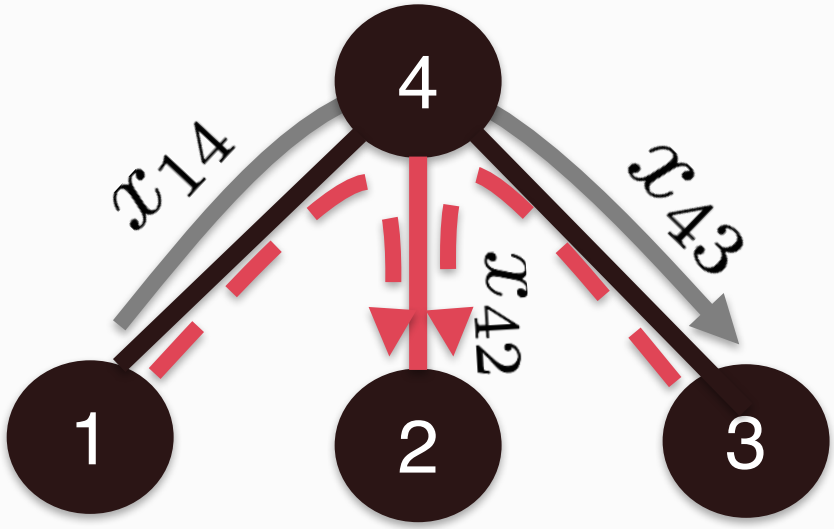
A “strawman” solution

- Suppose
  - we knew the path of all flows
  - we knew of every packet drop
- Tomography can find where failures are

If we assume there are enough flows

Example of doing tomography

$$x_{14} + x_{43} = 0$$
$$x_{14} + x_{42} = 1$$
$$x_{34} + x_{42} = 1$$



$$x_{ij} \in \{0, 1\}$$

$x_{ij} = 0$ if link not dropping packets

$x_{ij} = 1$ if link dropping packets

Only solvable if we have N independent equations

N = number of links in the network

Tomography is not always practical

Theoretical challenges

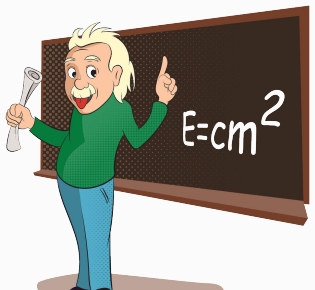
Engineering challenges

Set of equations doesn't fully specify a solution

- Number of active flows may not be sufficient
- Becomes NP hard

Many approximate solutions

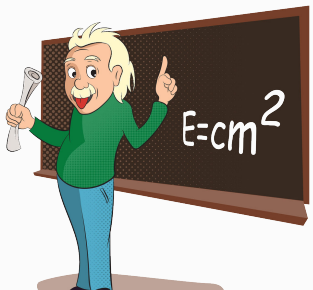
- MAX_COVERAGE (PathDump-OSDI 2016)
- They are sensitive to noise



Assume small number of failed links

AND

Fate Sharing across flows



Tomography is not always practical

Engineering challenges

- Finding path of all flows is hard

✗ Pre-compute paths

  – ECMP changes with every reboot/link failure

  – Hard to keep track of these changes

✗ Traceroute (TCP)

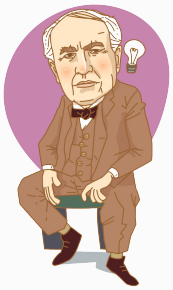
  – ICMP messages use up switch CPU

  – NATs and Software load balancers

✗ Infrastructure changes

  – Labeling packets, adding metadata

  – Costly



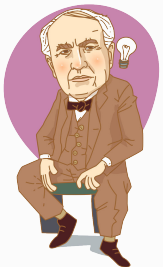
We show in this work

- Simple traceroute-based solution
  - Minimal overhead on switches
  - Tractable (not NP)
  - Resilient to noise
  - No infrastructure changes (host based app)

We *prove* its accurate

We can fix problems with traceroute

- Overhead on switch CPU
  - Only find paths of flows with packet drops
  - Limit number of traceroutes from each host
  - Explicit rules on the switch to limit responses
- NATs and Software load balancer
  - See paper for details



How the system works

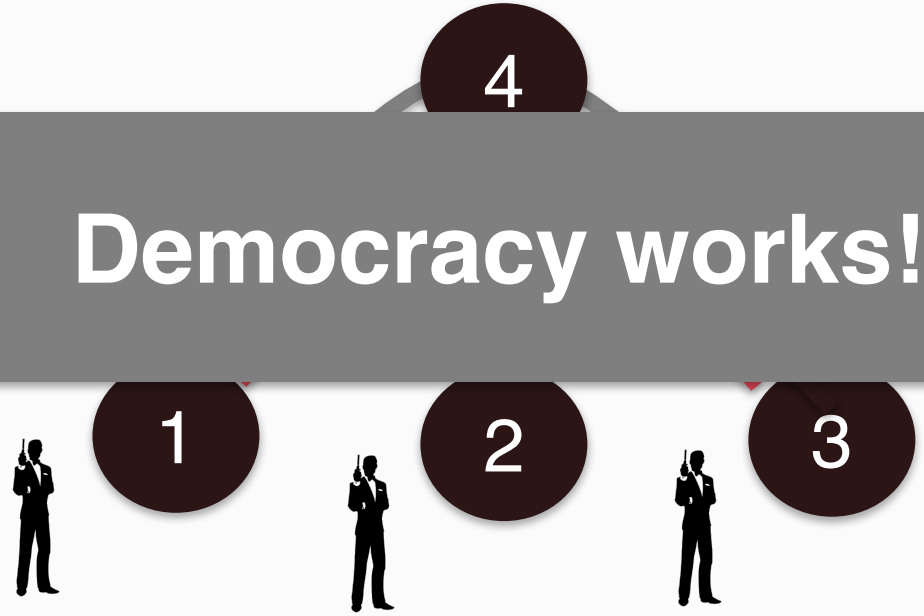
Monitoring agent:

Votes: if you don't know who to blame just blame everyone!

Notified of each TCP retransmission (ETW)

Path discovery agent finds the path of the failed flows

How the system works



Democracy works!

Can diagnose TCP flows

- Using *votes* to compare drop rates
  - For each flow we know the links involved
  - Link with most votes most likely cause of drops

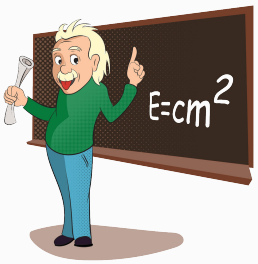
Assume ~~small number of failed links and~~ fate sharing across flows

Attractive features of 007

- Resilient to noise
- Intuitive and easy to implement
- Requires no changes to the network

We give theoretical guarantees

- We ensure minimal impact on switch CPU
  - Theorem bounding number of traceroutes
- We *prove* the voting scheme is 100% accurate when the noise is bounded
  - Depends on the network topology and failure drop rate



Questions to answer in evaluation

- Does 007 work in practice?
  - Capture the right path for each flow?
  - Find the cause of drops *for each flow* correctly?
- Are votes a good indicator of packet drop rate?
- What level of noise can 007 tolerate?
- What level of traffic skew can 007 tolerate?

Does 007 work in practice

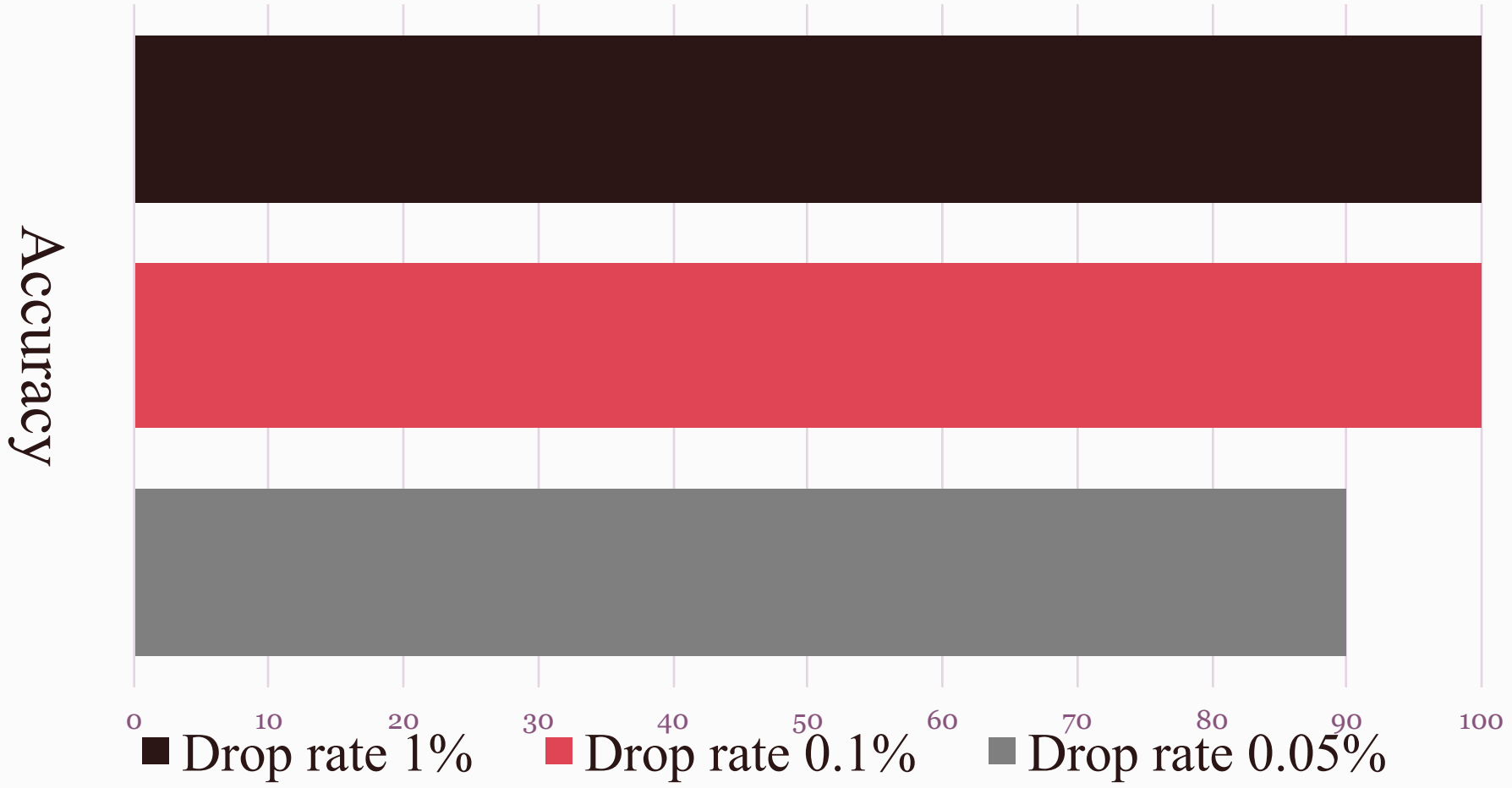
5 hour experiment

- Comparison to EverFlow (ground truth)
  - Do Traceroutes go over the right path? **YES**
  - Does 007 find the cause of packet drops? **YES**

Two month deployment

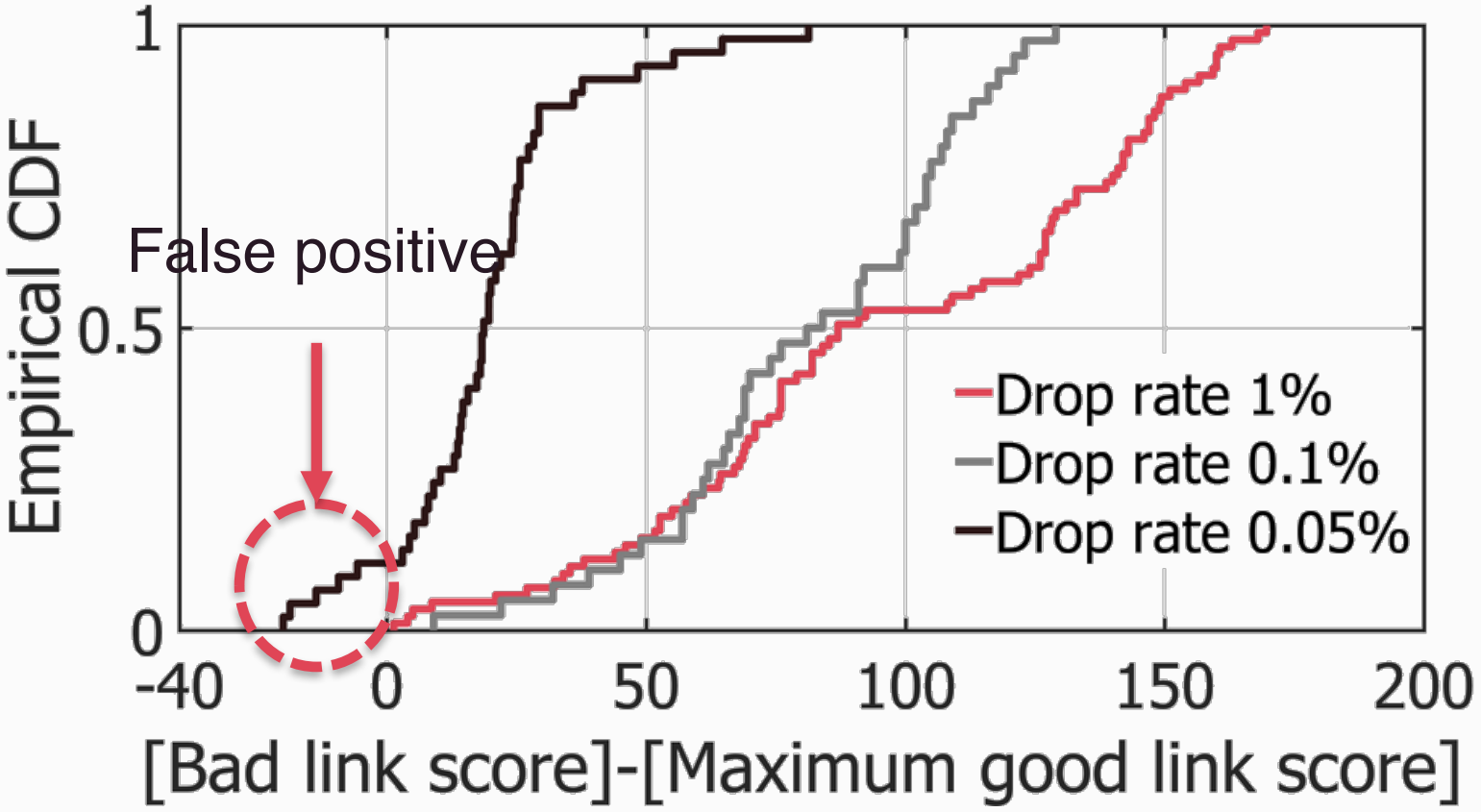
- Types of problems found in production:
  - Software bugs
  - FCS errors
  - Route flaps
  - Switch reconfigurations

Are votes correlated with drops?



Are votes correlated with drops?

- Test cluster (we know ground truth)

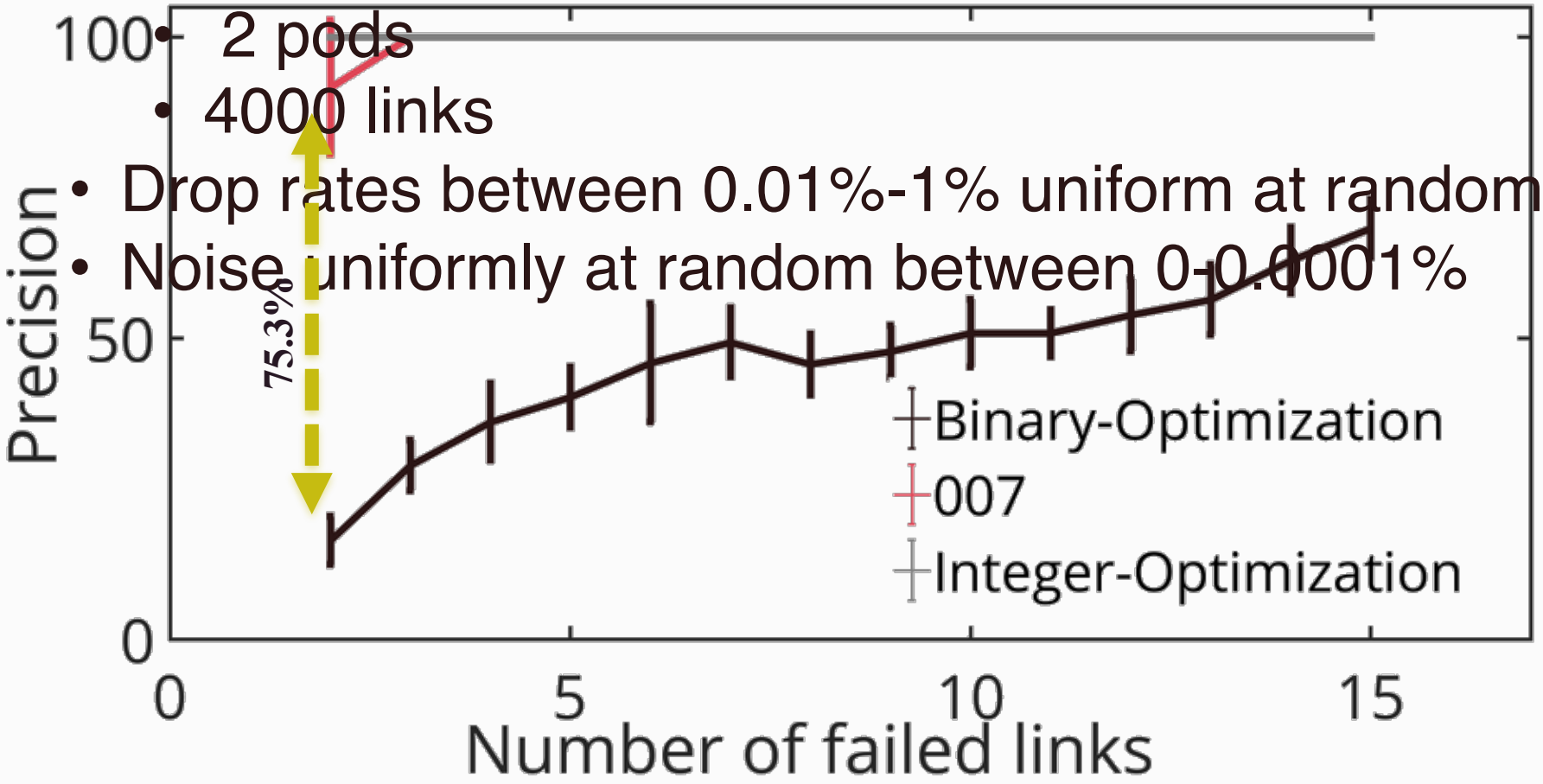


Comparison to MAX_COVERAGE

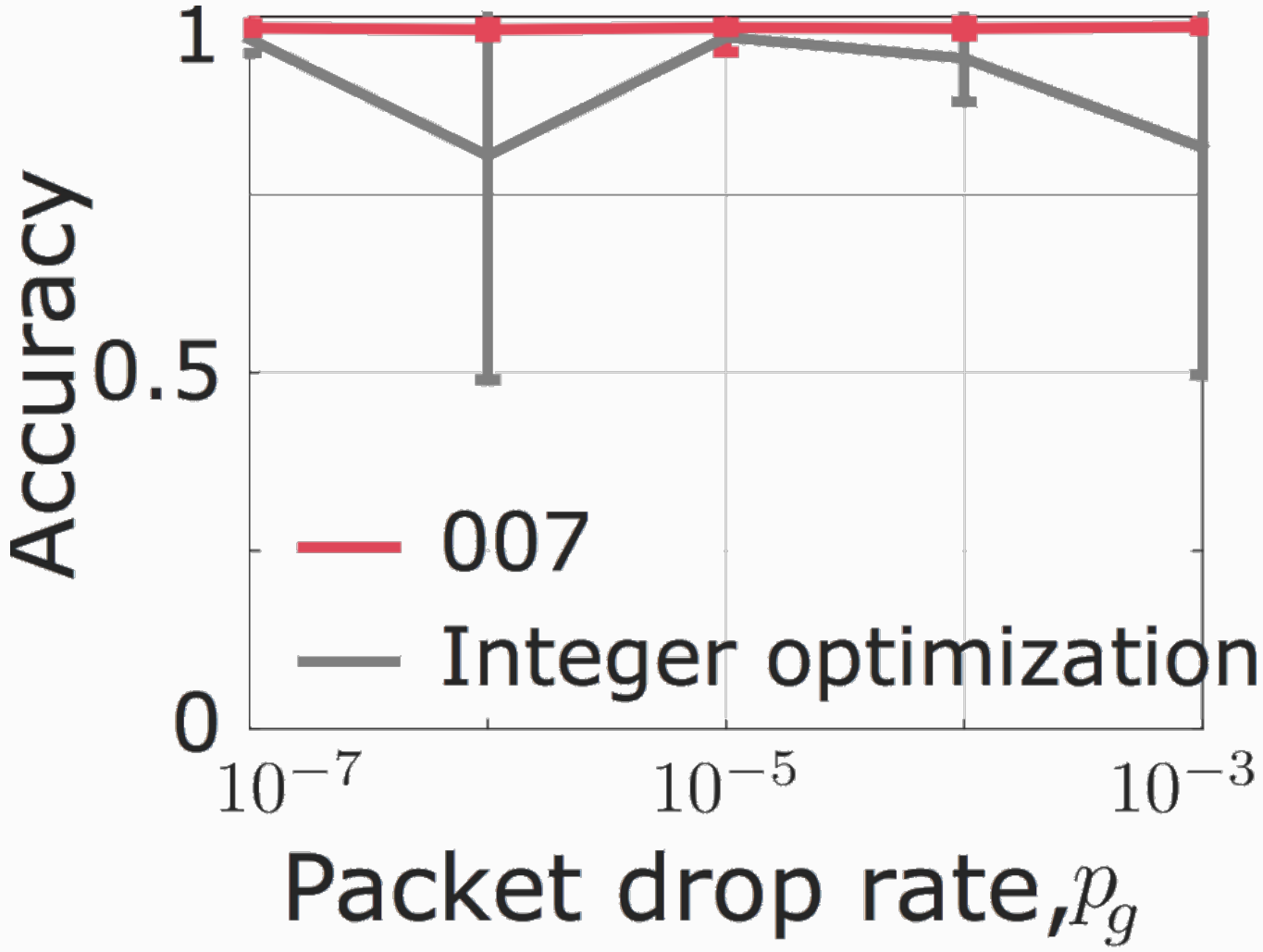
- MAX_COVERAGE (PathDump- OSDI 2016)
  - Approximate solution to a binary optimization
  - See 007 extended version for proof
  - Highly sensitive to noise
- Integer optimization
  - Improvement on the binary optimization approach
  - Reduces sensitivity to noise

Binary optimization underperforms

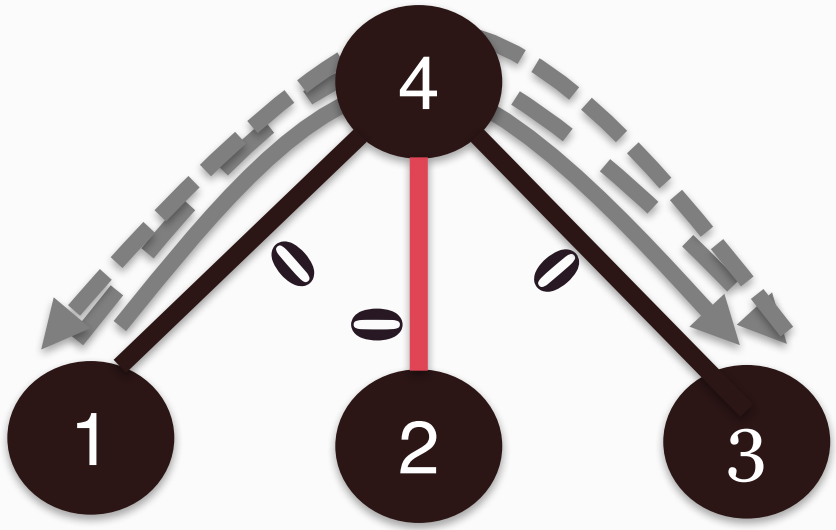
- Clos topology
  - 2 pods
  - 4000 links
- Drop rates between 0.01%-1% uniform at random
- Noise uniformly at random between 0-0.0001%



Is 007 robust to noise?



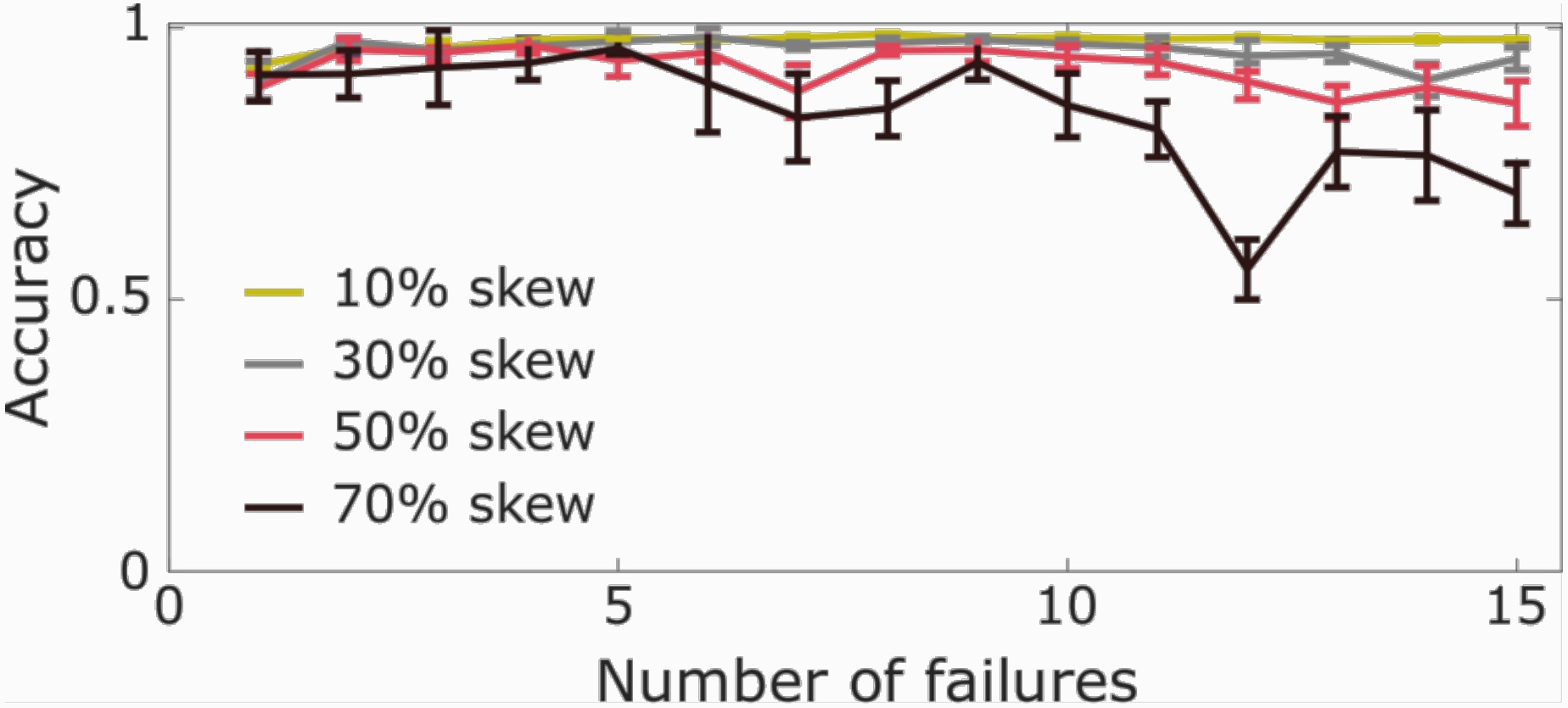
Skewed traffic causes problems



We don’t care about this *particular* case, because…
The failure isn’t impacting any traffic
But what if it had?

Is 007 impacted by traffic skew?

- More simulation results in the paper



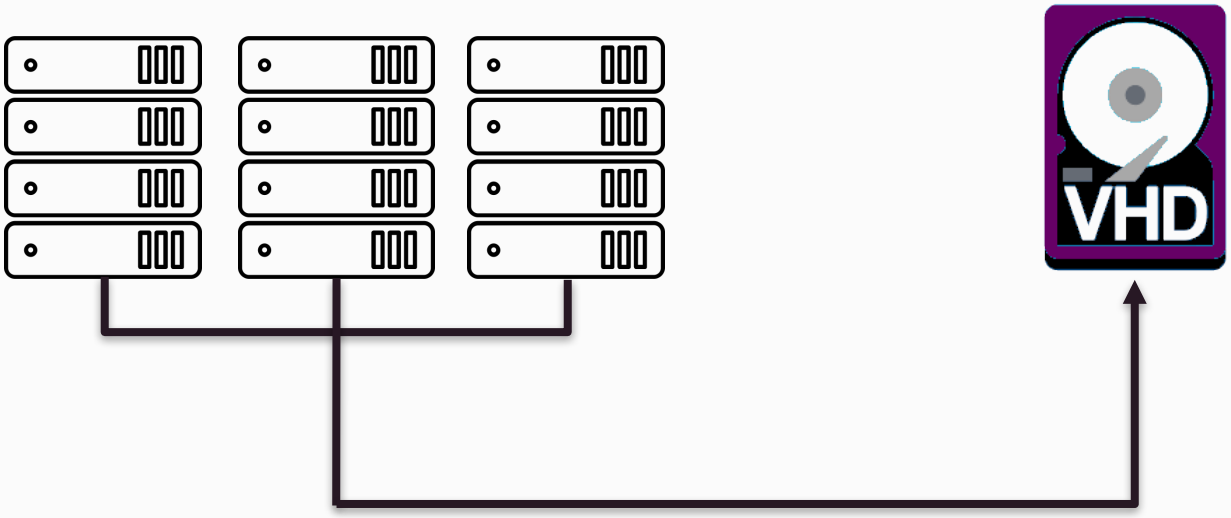
Conclusion

- **007:** ***simple voting scheme***
- Finds cause of problems for each flow
- Allows operators to prioritize fixes
- Analytically ***proven*** to be accurate
- Contained at the end host as an **application**
  - No changes to the network or destinations

Thank You

- Adi Aditya
- Alec Wolman
- Andreas Haeberlen
- Ang Chen
- Deepal Dhariwal
- Ishai Menache
- Jiaxin Cao
- Monia Ghobadi
- Mina Tahmasbi
- Omid AlipourFard
- Stefan Saroiu
- Trevor Adams

An example closer to home



Guaranteed Accurate

- Theorem:

For $n_{pod} \geq \frac{n_0}{n_1} + 1$, *Vigil will rank with probability* $1 - 2e^{-O(N)}$ *the bad links that drop packets with probability higher than all good links that drop packets with probability* p_b *if* p_g

$$p_g \leq \frac{1 - (1 - p_b)^{c_l}}{\alpha c_u}$$

where N *is the total number of connections between hosts,* α *and* c_l *are lower and upper bounds, respectively, on the number of* c_u *packets per connection.*

Minimal impact on switch CPU

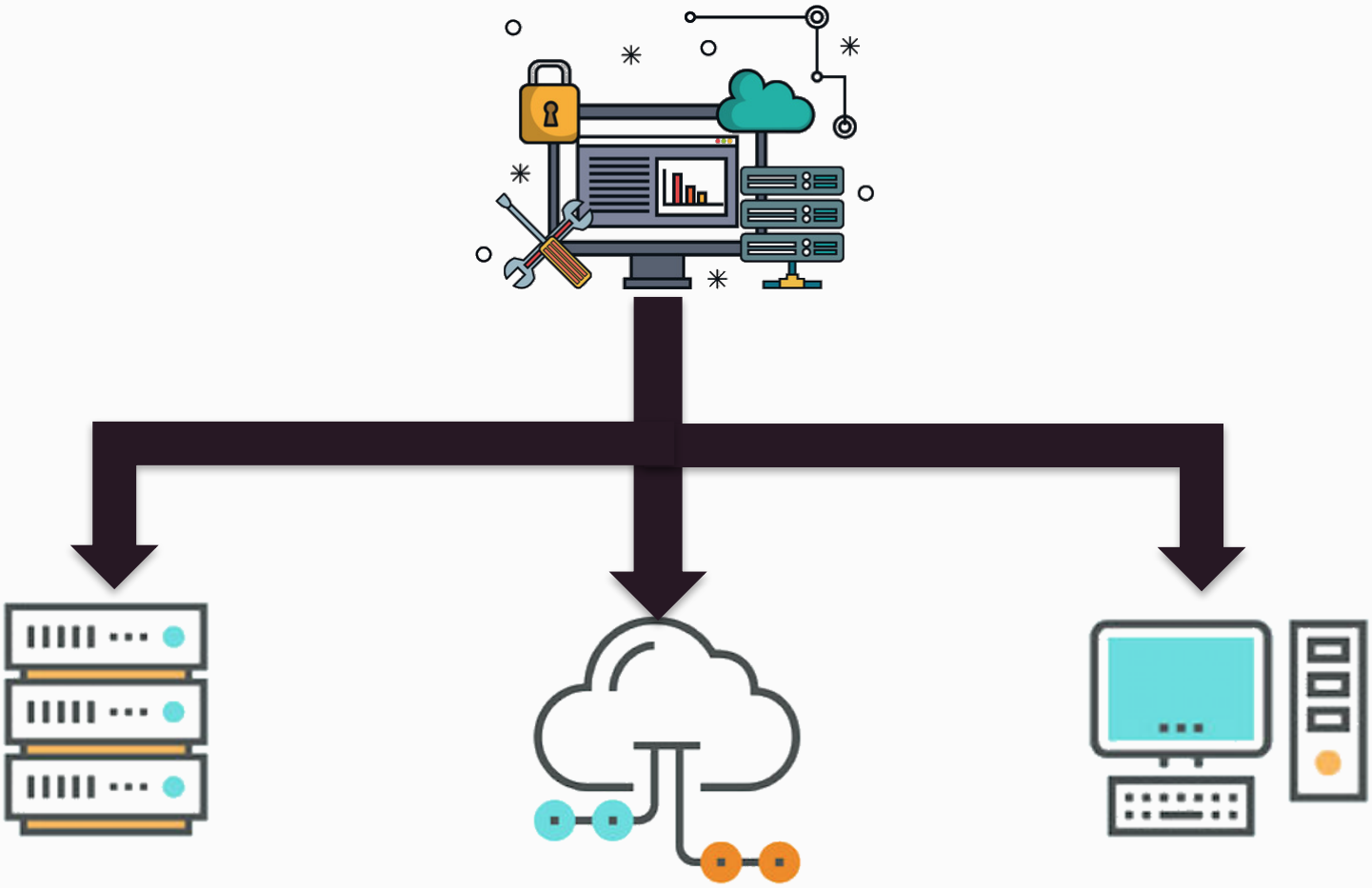
- Theorem:

The rate of ICMP packets generated by any switch due to a traceroute is below C_t *if the rate at which hosts trigger traceroutes is upper bounded as* T_{max}

$$C_t \leq \frac{n_1 n_2 T_{max}}{H \max\left[n_2, \frac{n_0^2(n_{pod}-1)}{n_0 n_{pod}-1}\right]},$$

Where n_0, n_1, n_2 *are the number of ToR,* T_1*, and* T_2 *switches respectively and* H *is the number of hosts under each ToR.*

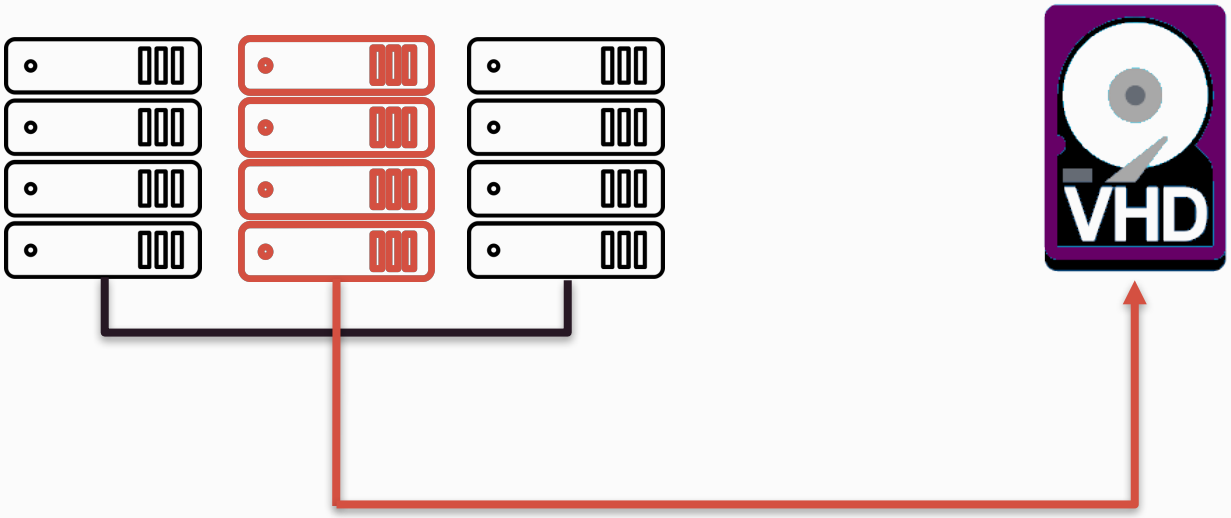
Failures are complicated



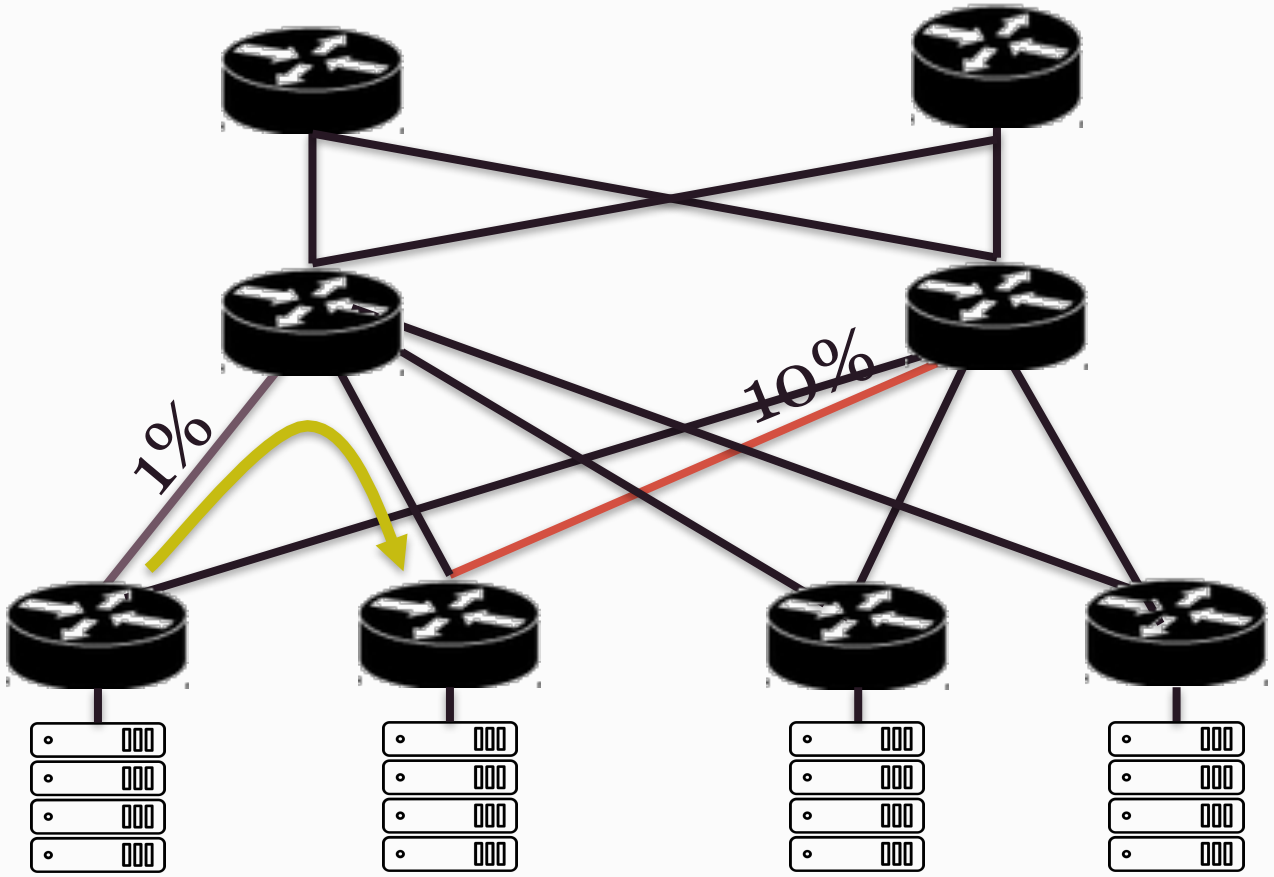
We can now prioritize fixes

- *We can answer questions like:*
  - *Why are connections to storage failing?*
  - *What is causing problems for SQL connections?*
  - *Why do I have bad throughput to* a.b.c.d?

An example closer to home



More than finding a few failed links



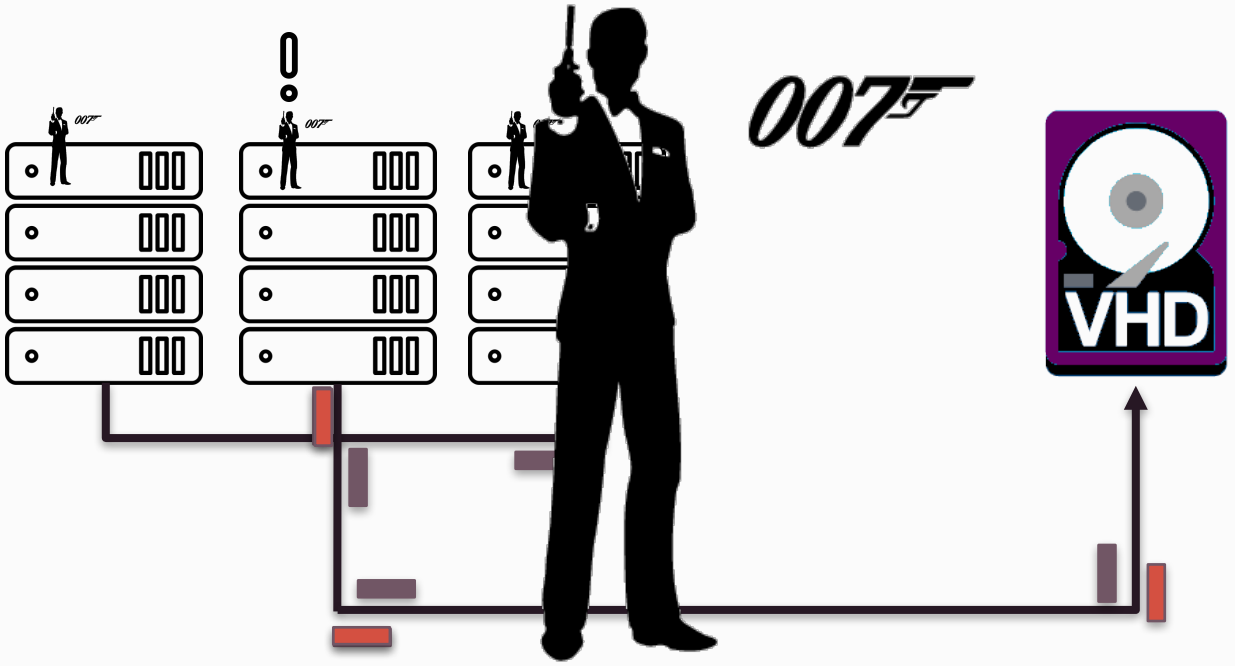
Past solutions don't help

- Don't allow for always on monitoring
  - Pingmesh [SIGCOMM-15]
  - EverFlow [SIGCOMM-15]
  - TRAT [SIGCOMM-02]
  - Other Tomography work
- Require changes to network/remote hosts
  - Marple [SIGCOMM-17]
  - PathDump [OSDI-16]
  - Link-based anomaly detection [NSDI-17]

Finding paths is also hard

- Infrastructure changes are costly
  - DSCP bit reserved for other tasks
  - Cannot deploy any changes on the destination end-point
- Reverse engineering ECMP also difficult
  - Can get the ECMP functions from vendors
  - Seed changes with every reboot/link failure
  - Hard to keep track of these changes
- Only option left: Traceroute
  - ICMP messages use up switch CPU
  - We cannot find the path of all flows
- Problem is not always fully specified
- Approximate solutions are NP hard
- And the approach is sensitive to noise

Our Solution



It detects retransmissions as soon as they happen through ETW

007 Monitors TCP connections at the host

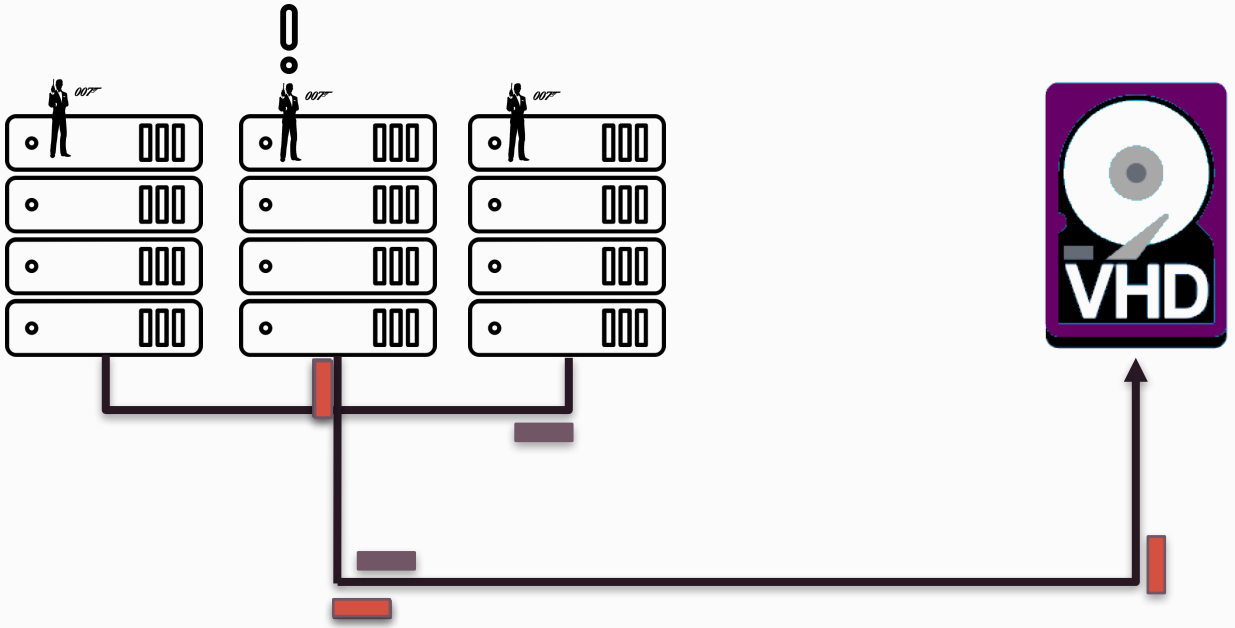
Mapping DIPs to VIPs

- Connections are to Virtual IPs
  - SYN packets go to a Software Load Balancer (SLB)
  - The host gets configured with a physical IP
  - All other packets in the connections use the physical IP
- Traceroute packets must use the physical IP

An evaluation with skewed traffic

- Traffic concentrated in one part of network
- Extreme example: most flows go to one ToR
  - Small fraction of traffic goes over failed links
  - Votes can become skewed
  - We call this a hot ToR scenario

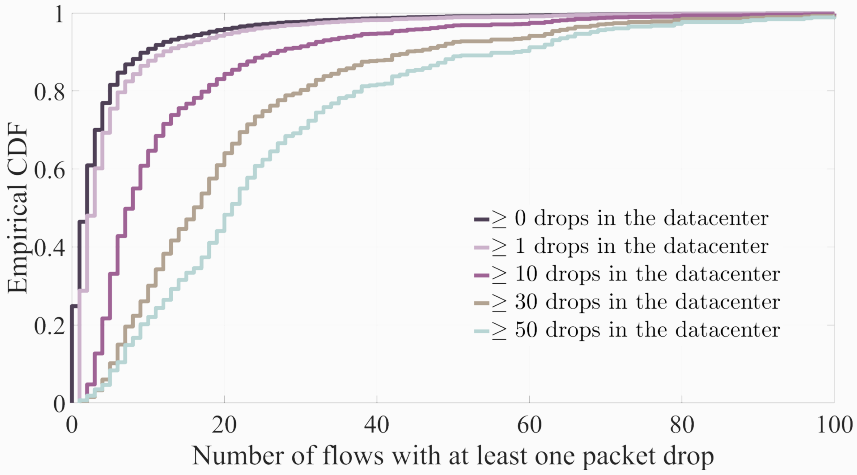
Our Solution

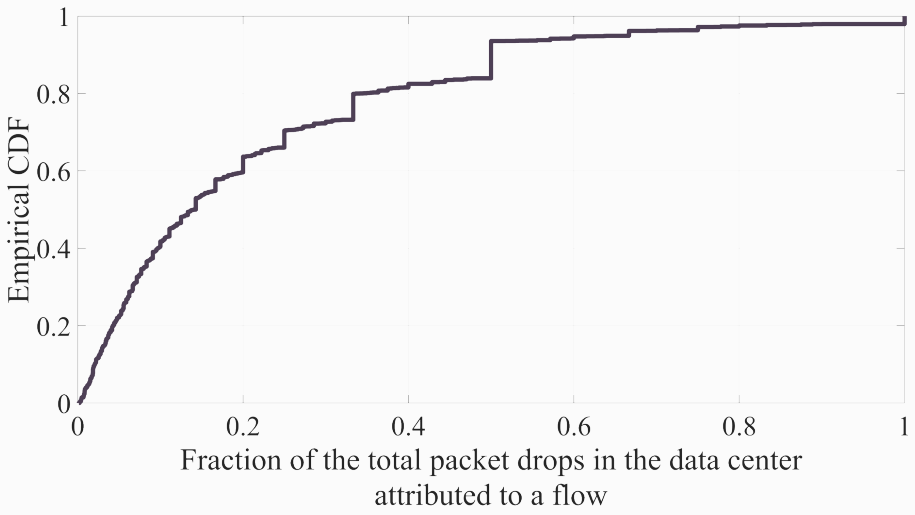


007 Monitors TCP connections at the host

It detects retransmissions as soon as they happen through ETW

Observation





Data gathered using the monitoring agent of NetPoirot
Uses ETW to get notifications of TCP retransmissions

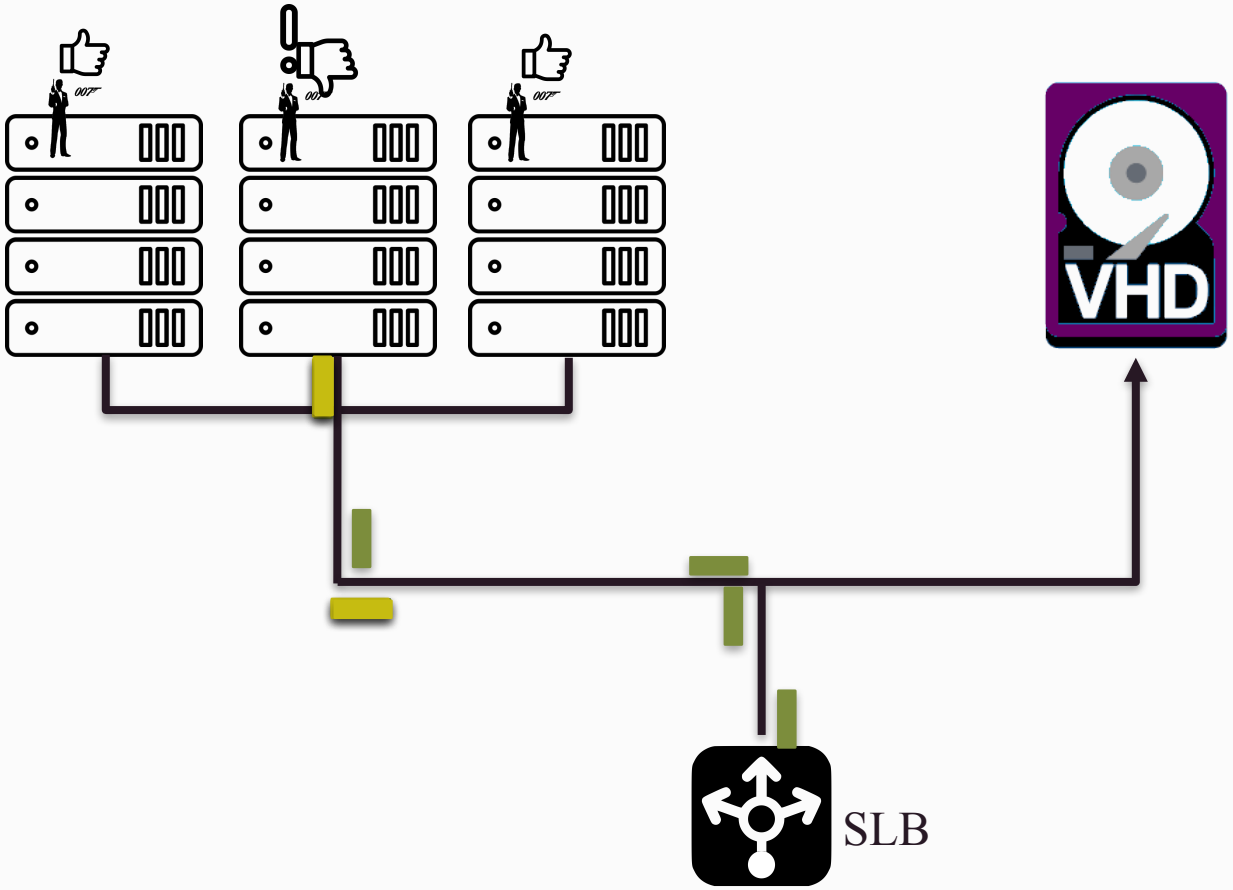
If path of *all* flows was known

- Given TCP statistics for existing flows
  - We know the paths that have problems
  - Without having to send any probe traffic
  - Without having to rely on packet captures
- We can also find the failed links

We can now prioritize fixes

- *We can answer questions like:*
  - *Why are connections to storage failing?*
  - *What is causing problems for SQL connections?*
  - *Why do I have bad throughput to* a.b.c.d?
- Just one catch:
  - Needs to know retransmissions
  - Ok for infrastructure traffic (e.g. storage)
  - See paper on how to extend to VM traffic

Each connection votes on the status of links
Get the ECMP traceroute mapping from SLB
bad links get a vote of 1
good links get a vote of 0

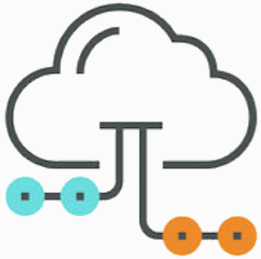


Where in the network?

Holding the network accountable

- Given impacted application find links responsible
  - Allows us to prioritize fixes
- Given a failed device quantify its impact
  - Estimate cost of failures in customer impact

Failures are hard to diagnose



High CPU load
High I/O load
Reboots
Software bugs

BGP link flaps
FCS errors
misconfigurations
Switch Reboots
Congestion
Hardware bug
+
Millions of devices

Bad design
Software bugs
High CPU usage
High memory usage