New Challenges in Network Diagnosis

Minlan Yu
Harvard University

Joint Work with Yuliang Li, Jiaqi Gao, Rui Miao, Mohammad Alizadeh, Nofel Yaseen, Robert MacDavid, Felipe Vieira Frujeri, Vincent Liu, Ricardo Bianchini, Ramaswamy Aditya, Xiaohang Wang, Henry Lee, David Maltz, Behnaz Arzani

Diagnosing Network Applications

- Many networked applications in large production networks

- Require high Availability and SLO (Service-level Objectives)

- Need to quickly locate problems when things go wrong

New Challenges for Diagnosis

Ultra low latency
(The killer microseconds)

Growing complexity
(Involves many components and layers in cloud-scale services)

At Scale
(~100K servers/VMs/processes)

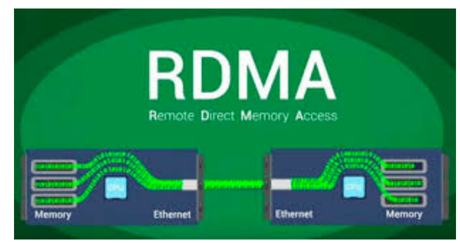
Challenge 1: Ultra Low Latency

- The killer microseconds

Ultra low latency Applications
(real-time reinforcement learning, large-scale distributed systems, packet-level traffic analysis)

GPU, TPU
O(10us)

flash O(10us),
NVM O(1us)



RDMA O(1us)

Many Fine-time scale Events

Performance is sensitive to many fine-time scale events

- More intermittent events
- Many events happen at the same time
- Small events have cascading impact

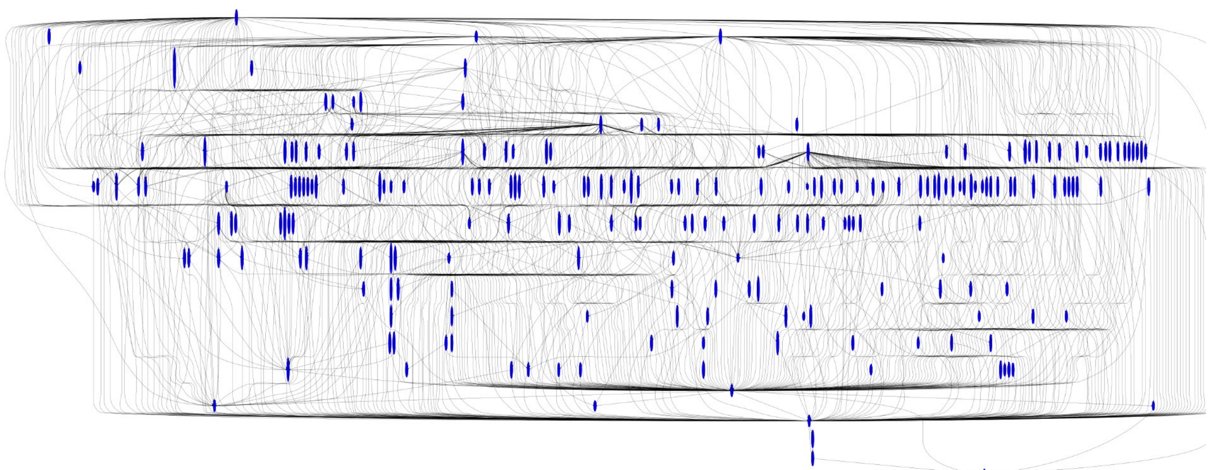
across components and over time

Link failure
Loop
Context switching
Packet burst
Traffic surge
DDoS
Packet delay
Congestion
Burst Loss
Lost packet
ad bursts
Interrupts
Incast
Gabage collection

How to collect these fine-grained events at scale?
How to correlate them with performance problems?

Challenge 1: Ultra Low Latency at Scale

- Overall latency ≥ latency of slowest component (Tail latency!!!)
  - small blips on individual flows cause delays
  - touching more machines increases likelihood of delays



Bing query workflow (SIGCOMM'13)

Challenge 2: Growing Complexity

- Developers have to master growing complexity

Packet losses Long delay Burst Throughput drops Transient stalls Connectivity issues	Networking Storage Databases Load balancing Security ISPs	Optical links Switches Servers NICs Accelerators	Topology Routing Device configs OS stack Transport …

Find needles in the haystack

Challenge 2: Growing Complexity

- Developers have to master growing complexity
- The blame game
  - "There must be something wrong on the component that I don't control or understand"
  - "Things work fine at my component before and nothing changed"
- Network is often the target for blames
  - Interconnected with many components
  - Less visible to other upper layer applications

Automatically identify the right team/component

Challenge 2: Growing Complexity

- Developers have to master growing complexity
- The blame game
  - "There must be something wrong on the component that I don't control or understand"
  - "Things work fine at my component before and nothing changed"
- Network is often the target for blames
  - Interconnected with many components
  - Less visible to other upper layer applications

Automatically identify the right team/component

New Challenges for Diagnosis

Ultra low latency
(The killer microseconds)

Growing complexity
(Involves many teams and layers in cloud-scale services)

At Scale
(~100K servers/VMs/processes)

Fine-grained data collection at scale → Automatic analysis to identify the responsible components

This talk

- DETER: Record-and-Replay for TCP
  - Collect detailed yet lightweight packet information at scale
- Scouts: Domain-customized incident routing
  - Automatically direct incident tickets to the right team

DETER

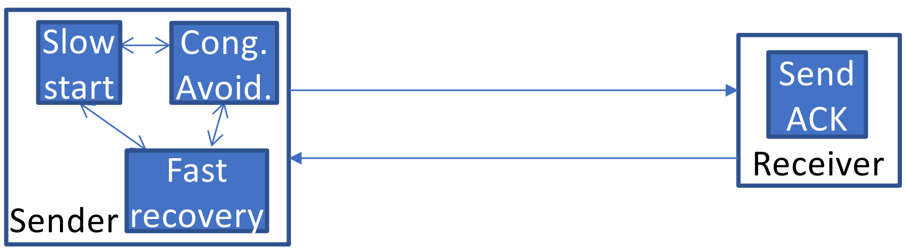
Detailed packet-level information
for TCP diagnosis

TCP performance diagnosis is important

- Highly distributed applications in large production networks
  - These apps rely on high throughput, low latency of all the TCP connections
- Yet, TCP problems happen all the time
  - Tail latency matters
  - A single flow with long latency can slow down the entire job

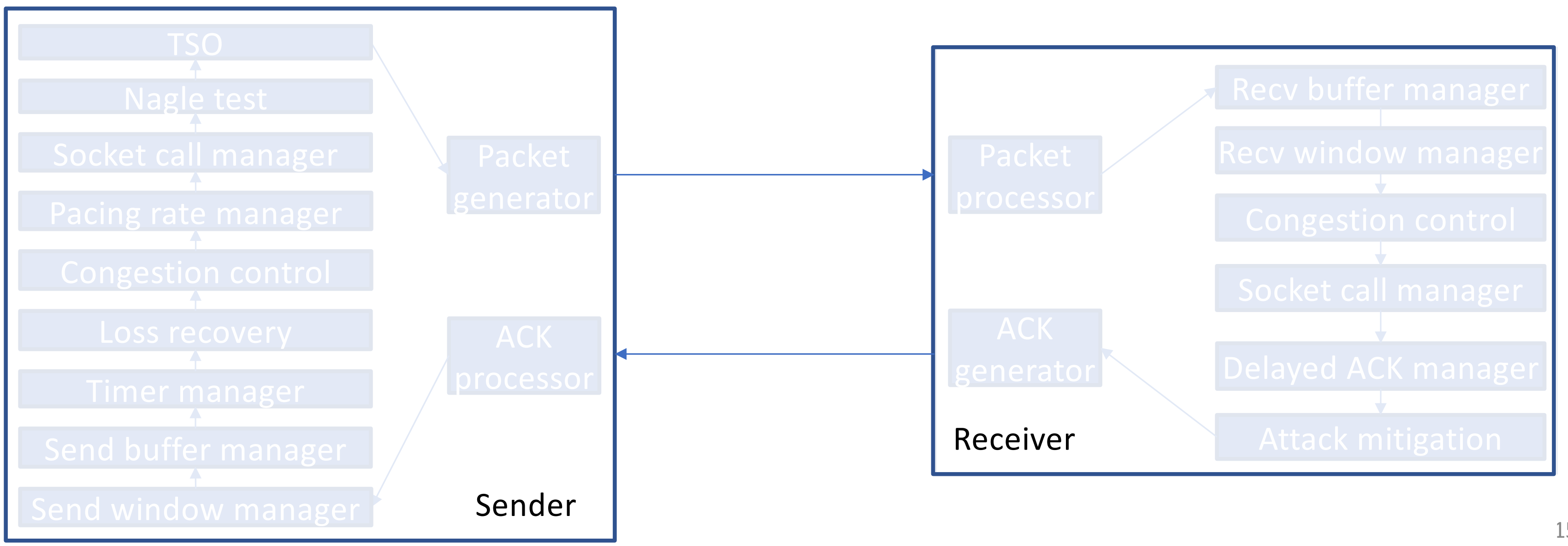
Why is diagnosing TCP hard?

- TCP in the text book:



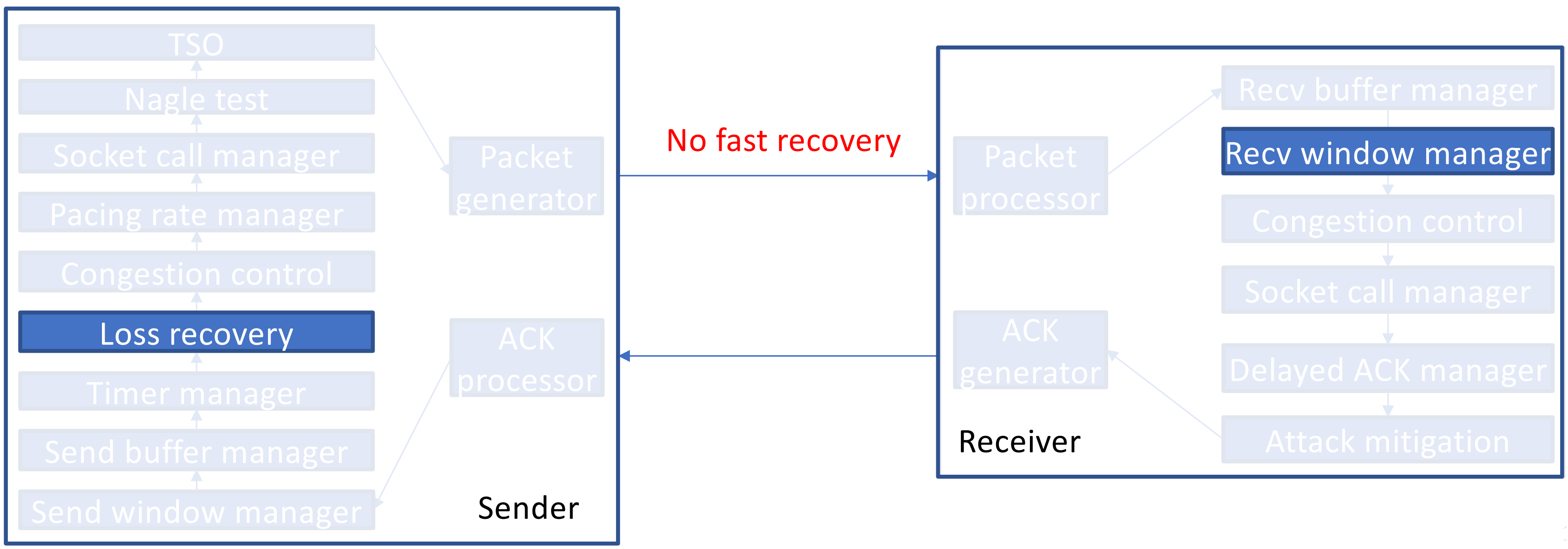
TCP is complex!

- Reality…



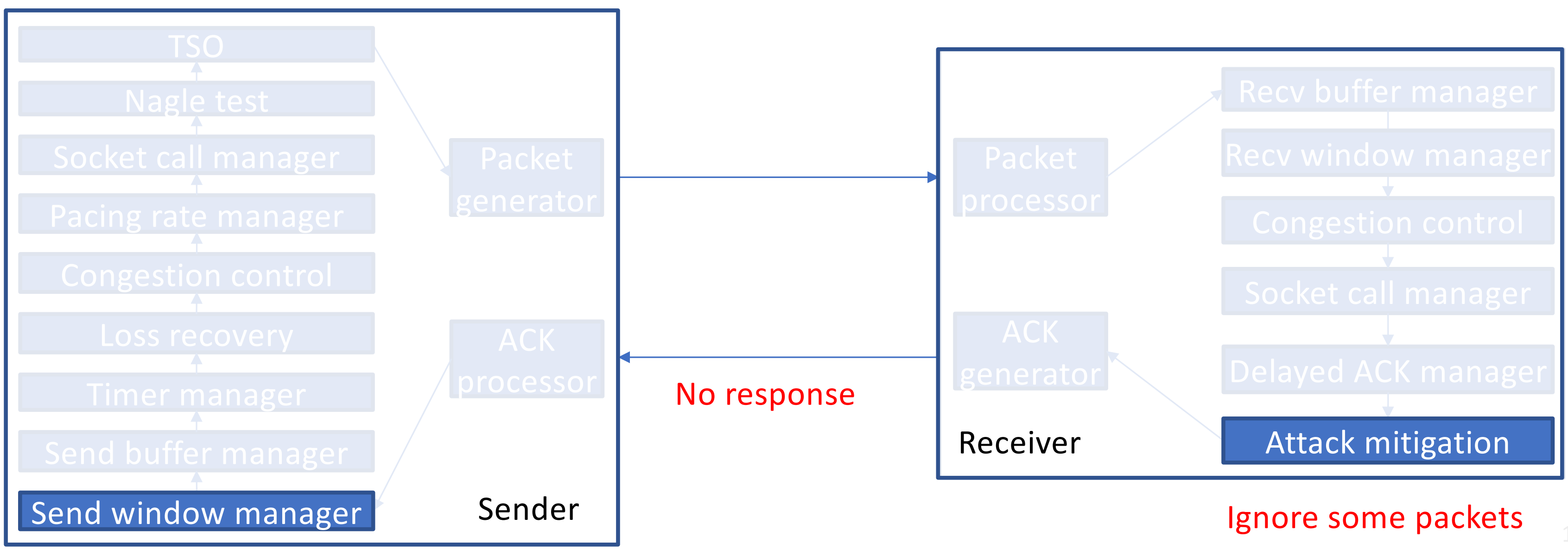
TCP is complex!

- Unexpected interactions between diff components



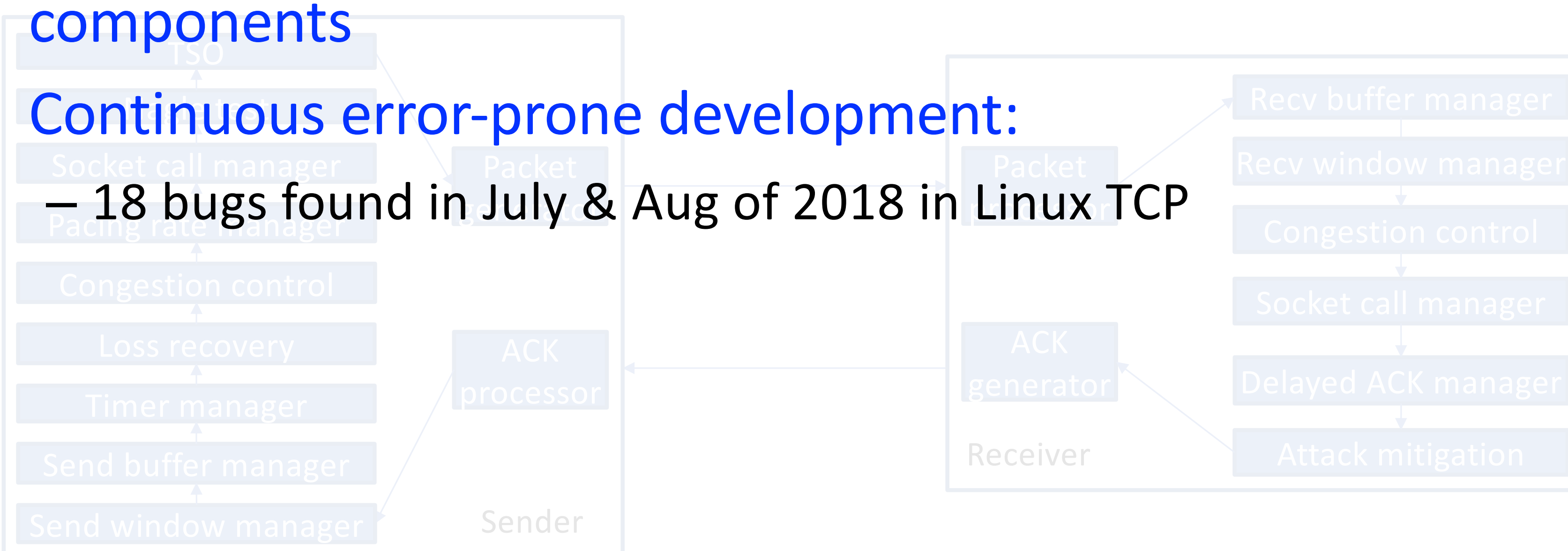
TCP is complex!

- Unexpected interactions between diff components



TCP is complex!

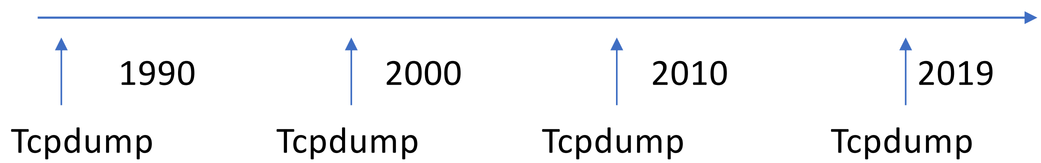
- Unexpected interactions between diff components
- 63 parameters in Linux 4.4 TCP that tune the behaviors of diff components
- Continuous error-prone development:
  – 18 bugs found in July & Aug of 2018 in Linux TCP



How do we diagnose TCP today?

Tcpdump

Detailed diagnosis is not scalable

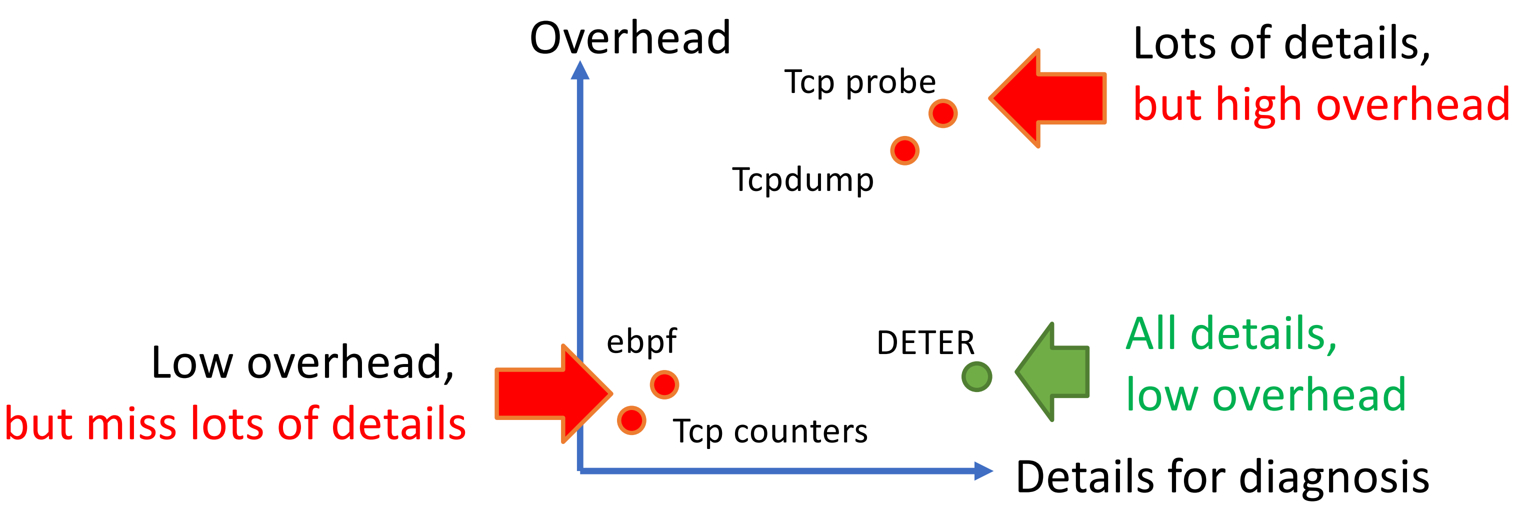


- Bandwidth: 10Mbps to 100Gbps
- #hosts: 10s to 100,000s

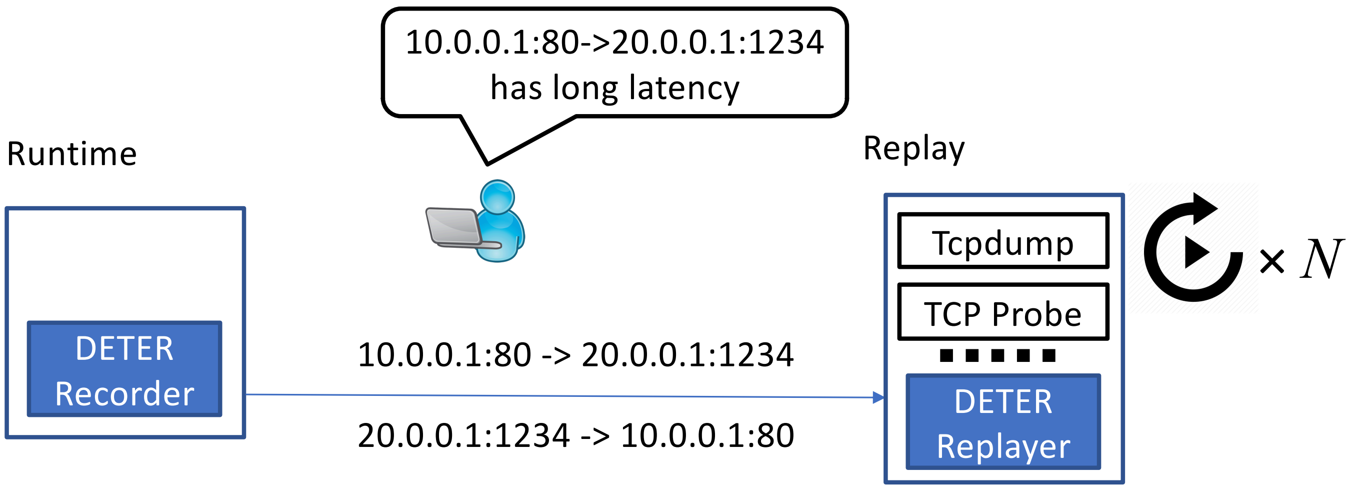
- Too much overhead!

Tension between more details and low overhead

- Existing tools cannot achieve both — Runtime record = Data for diagnosis
- DETERministic Record and Replay — Runtime record < Data for diagnosis



DETER overview



Lightweight record

- Run continuously
- On all hosts

Deterministic replay

- Capture packets/counters
- Trace executions
- Iterative diagnosis

Lightweight record **Deterministic replay**

Intuition for being lightweight

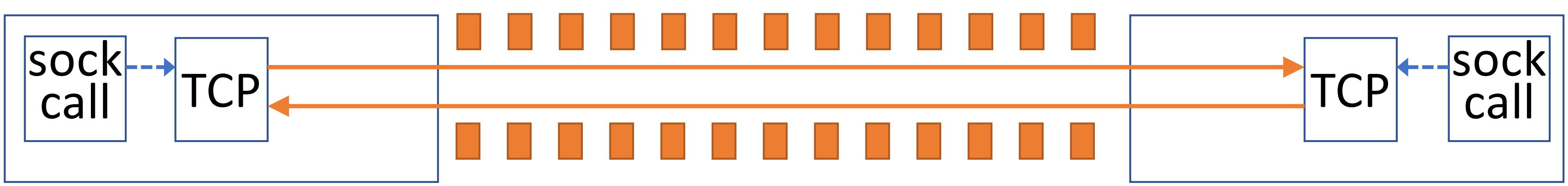
Lightweight record

Deterministic replay

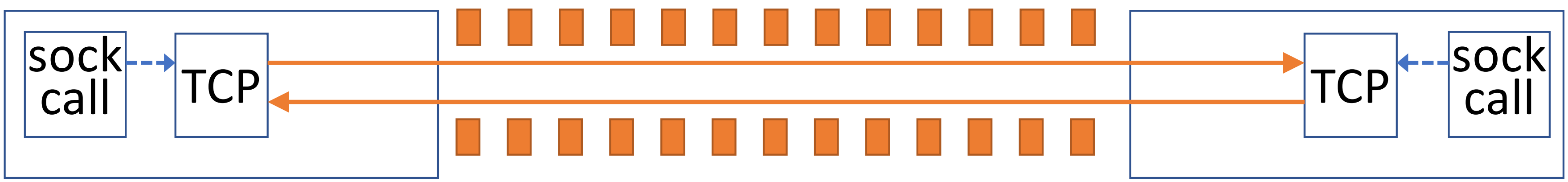
FAIL!

Record socket calls

Automatically generate packets



Non-deterministic interactions w/ many parties



Non-deterministic interactions w/ many parties

Key contribution:

- Identifying the minimum set of data that enables deterministic replay

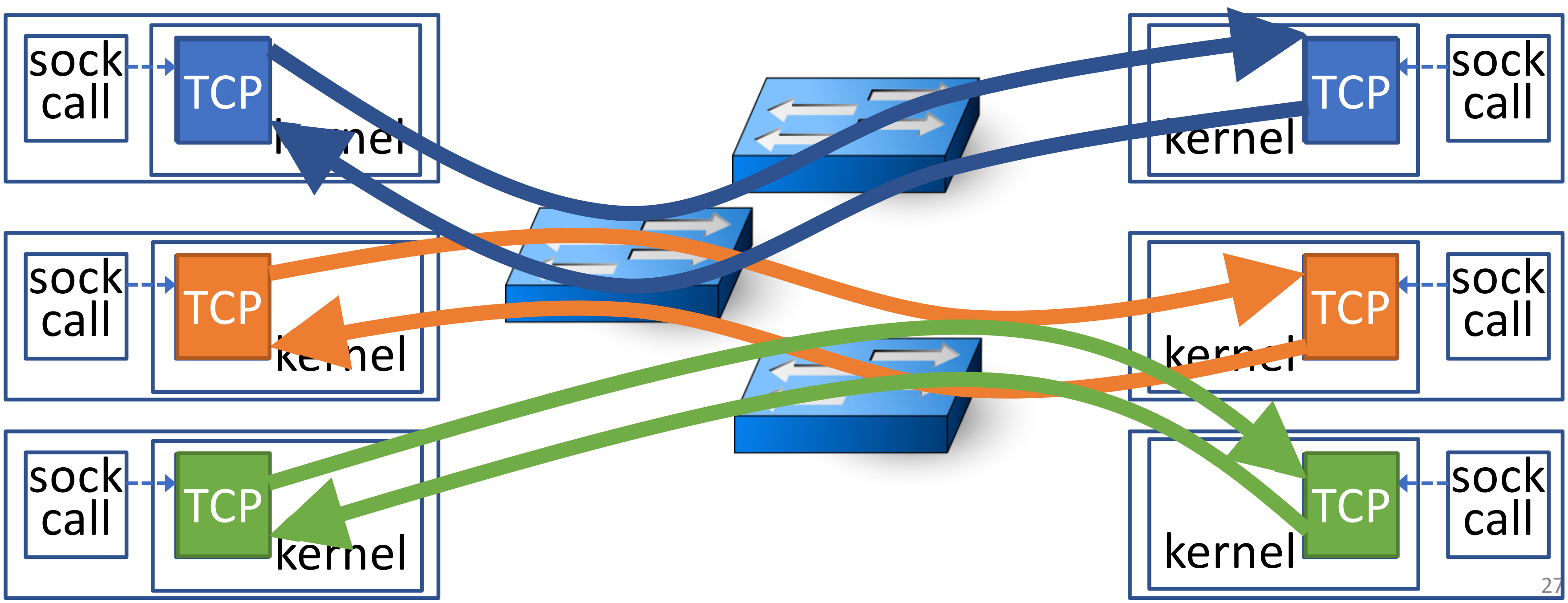
Two challenges:

- Network wide: non-deterministic interactions across switches and TCP
- On host: non-determinisms within the kernel

Butterfly effect

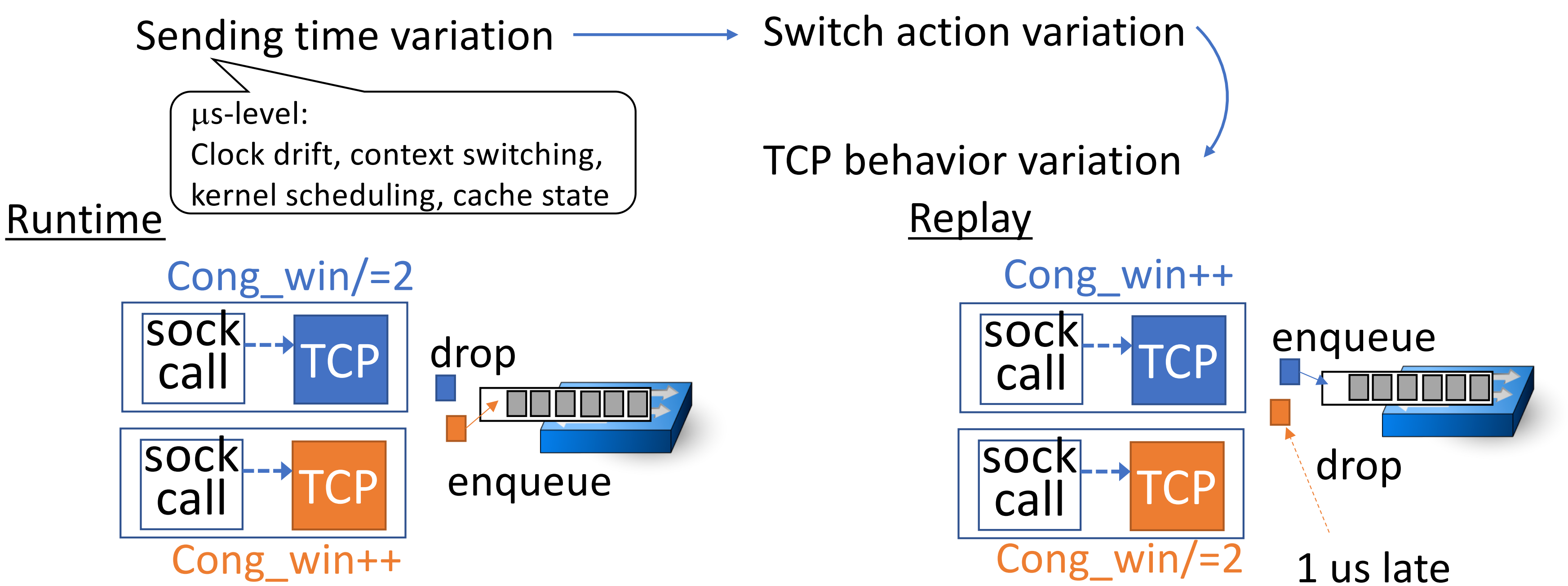
Challenge 1: butterfly effect

- The **closed loop** between TCP and switches amplifies small noises



27

Challenge 1: butterfly effect

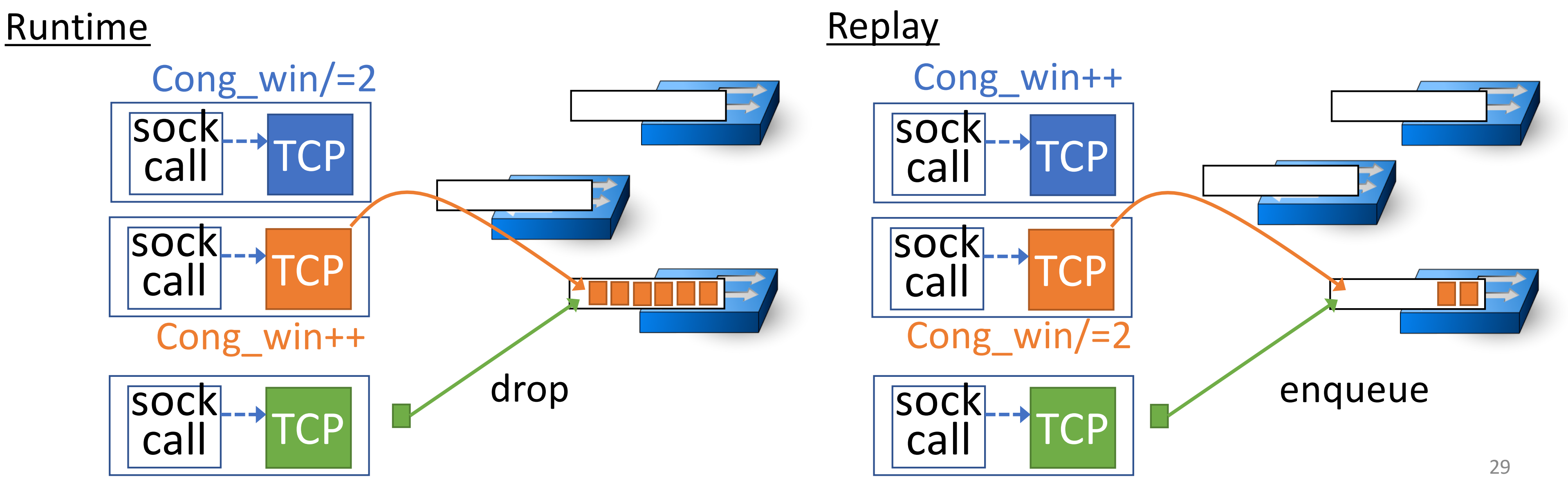


Challenge 1: butterfly effect

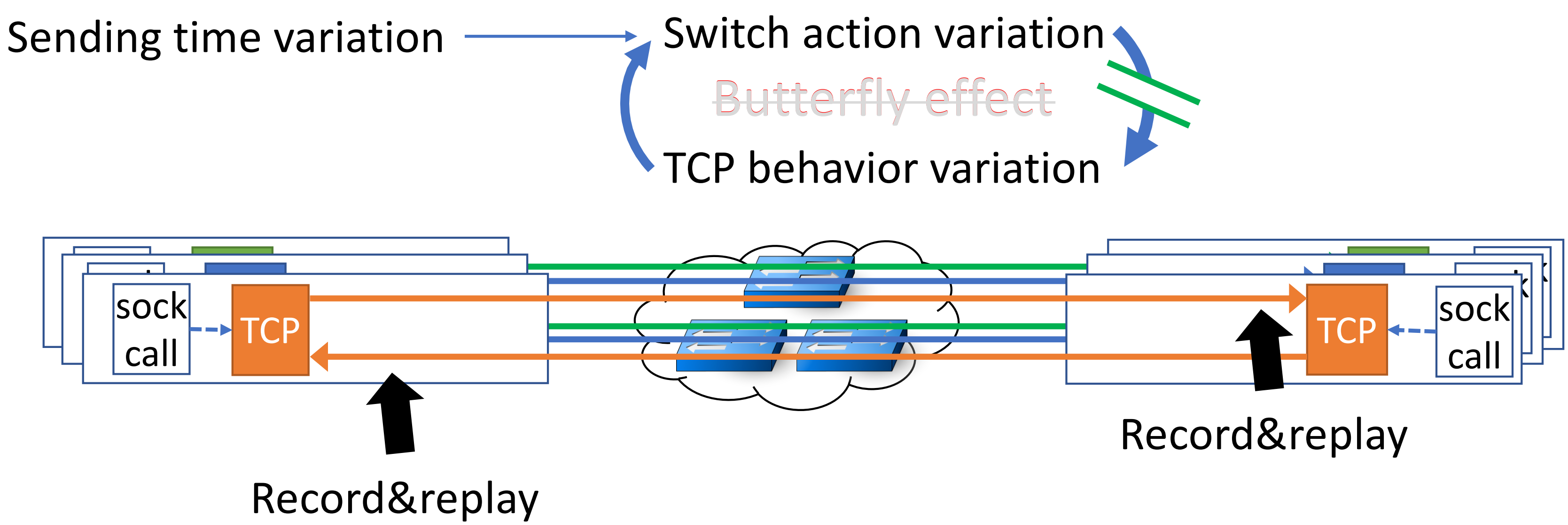
Sending time variation → Switch action variation

Butterfly effect

TCP behavior variation

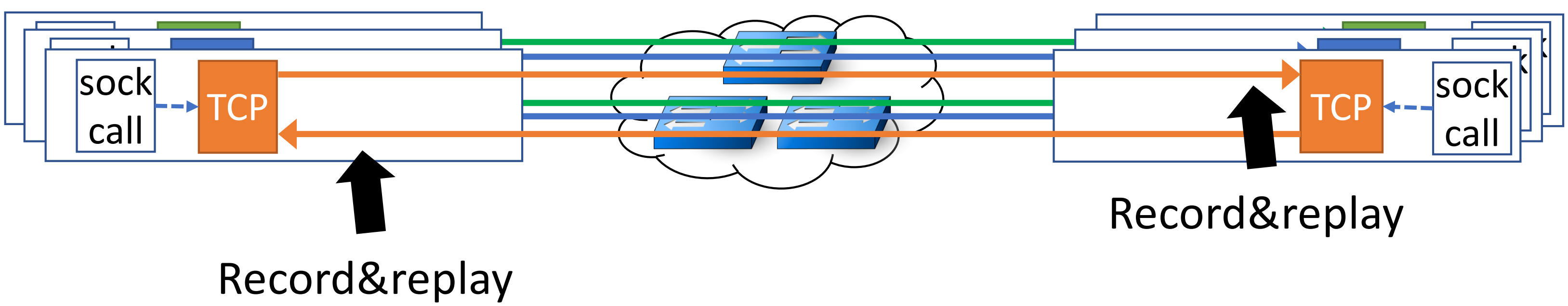


Reduce Butterfly Effect



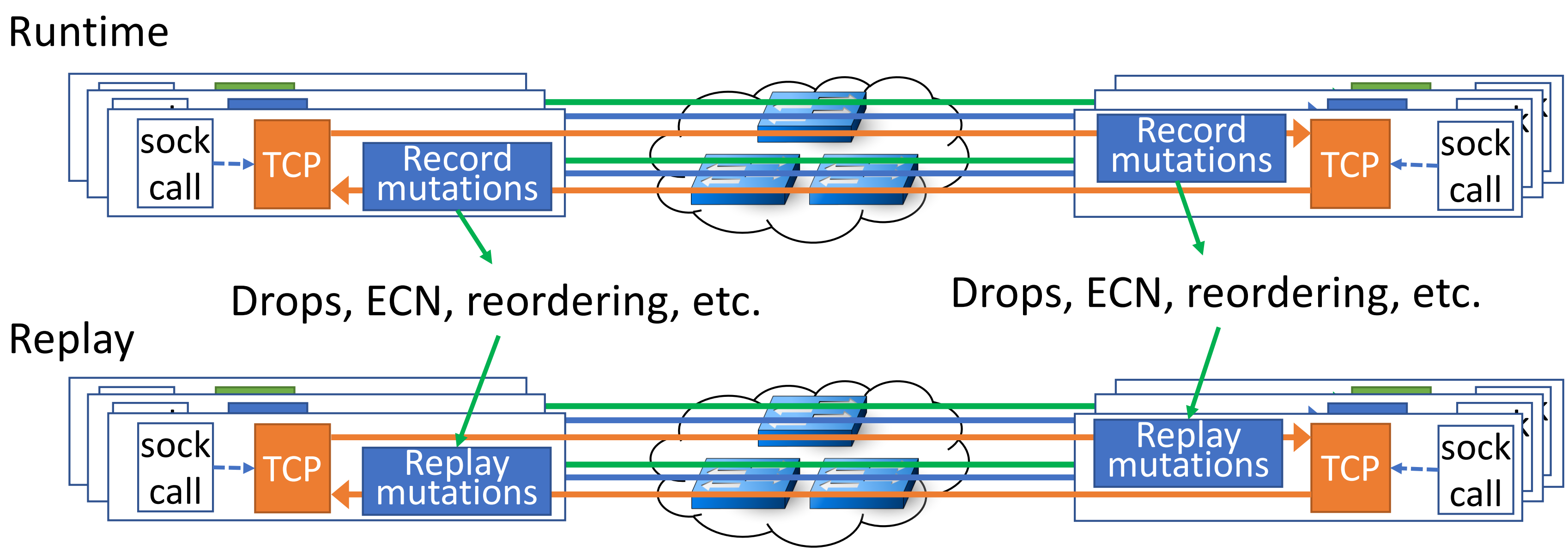
Challenge 1: butterfly effect

- ~~Record all the packets into TCP?~~ High overhead



Challenge 1: butterfly effect

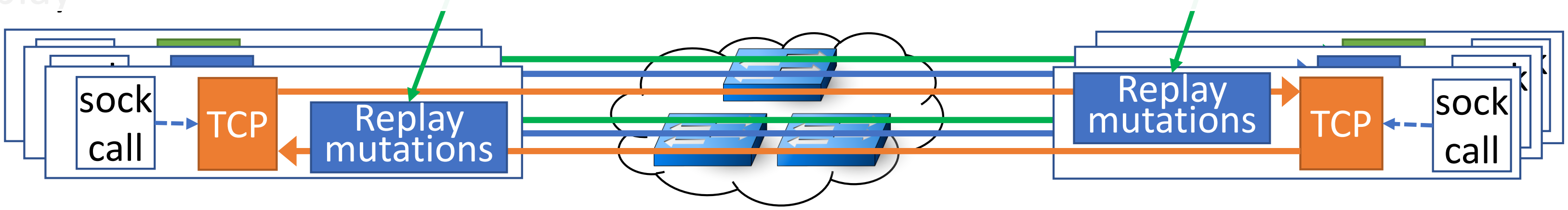
- Solution: record&replay **packet stream mutations**



Challenge 1: butterfly effect

- Solution: record&replay **packet stream mutations**
  + **Low overhead**:
    Drop rate < 10^{-4};
    ECN: 1 bit/packet;
    Reordering is rare
  + **Replaying each TCP connection is independent**
    Connections interact via drops and ECN, which we replay.
  + **Need no switches for replay**

Resource-efficient replay:
- Just need two hosts



Implementation

- DETER in Linux 4.4
  - Just need 139 lines of changes to Linux TCP
- Lightweight recording
  - Storage: 2.1%~3.1% compared to compressed packet header traces.
  - CPU: < 1.5%

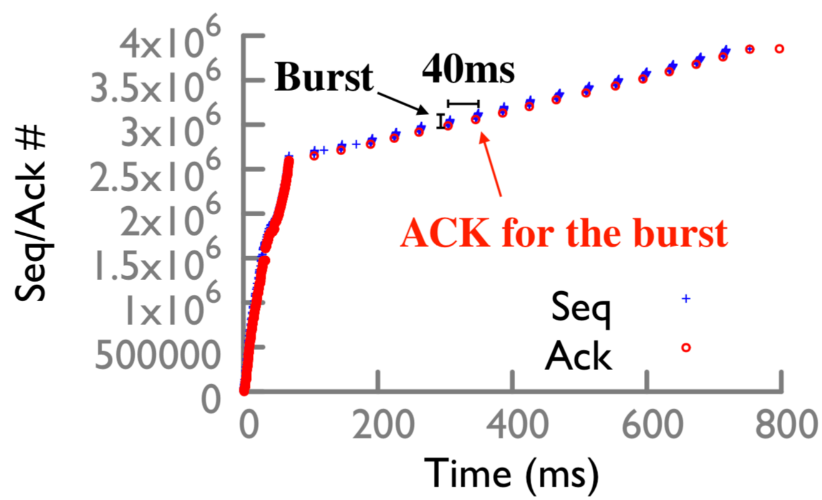
Case study in Spark

- Terasort 200 GB on 20 servers (4 cores each) on EC2, 6.2K connections
- Replay and collect trace for flows with 99.9 percentile latency

Flow size (MB)	<0.1	[0.1, 1]	[1, 10]	>10
RTO	8	3	4	0
FR	74	0	0	0
Delayed ACK	0	0	18	0
Rwnd=0	0	0	1	1
Slow start	0	0	1	0

Case study in Spark

- Iteratively debug individual flows
  - E.g., delayed ACK
- Packet traces
  - Burst-40ms-ACK pattern
- Trace TCP executions
  - The receiver explicitly delays the ACK,
  - because the recv buffer is shrinking
  - Caused by the slow receiver



Other use cases

- RTO caused by different reasons
  - Delayed ACK, Exponential backoff,
  - small messages, misconfiguration of receiver buffer size
- We can also diagnose problems in the switches
  - Because we have traces, we can push packets into the network
  - In simulation (requires modeling switch data plane accurately)
  - Case study: A temporary blackhole caused by switch buffer sharing

Conclusion

- Performance diagnosis in large networks is challenging
  - Many problems in the TCP stack

- DETER enables light weight recording and deterministic TCP replay
  - Key challenge: butterfly effect between TCP and switches
  - Record & replay packet stream mutations to break the closed loop

- Deter is opensourced
  - https://github.com/harvard-cns/deter

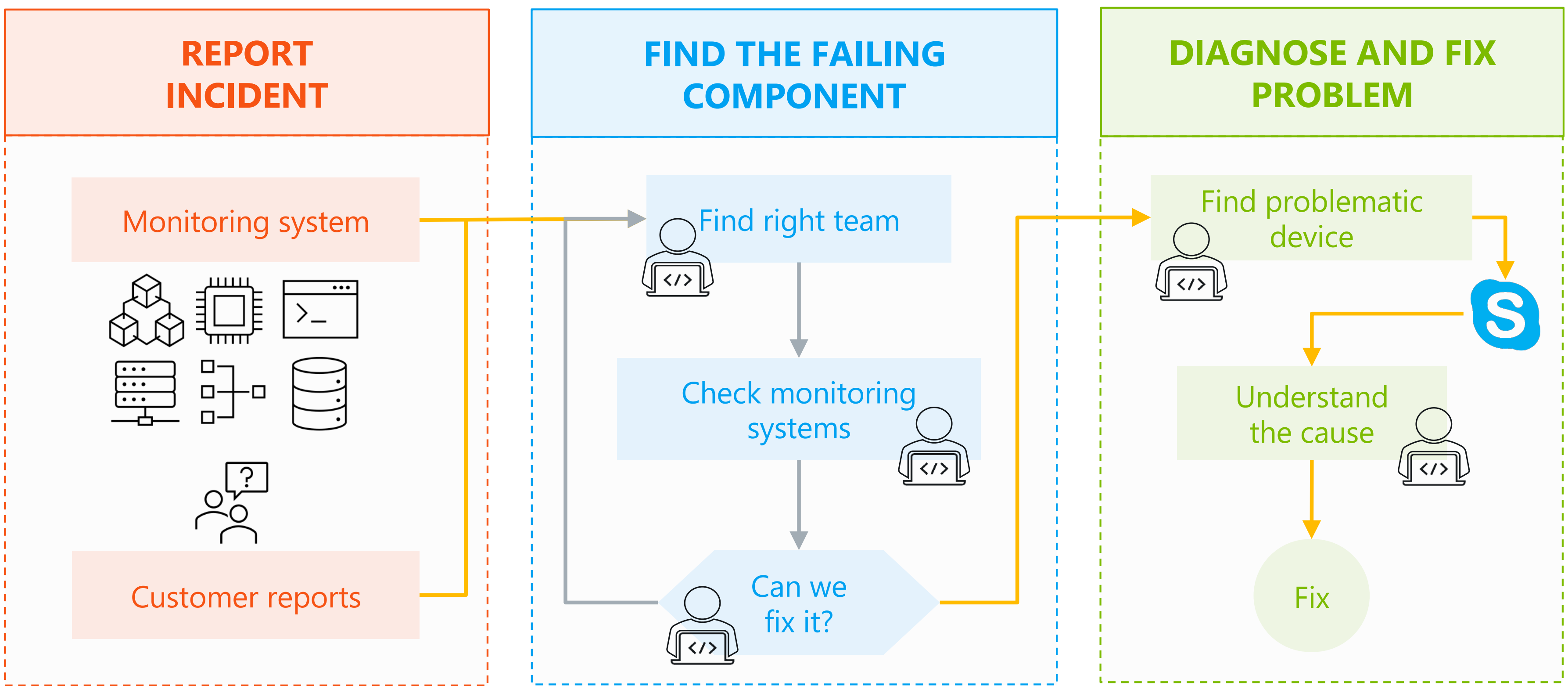
Scouts

Automatic diagnosis
using domain-customized incident routing

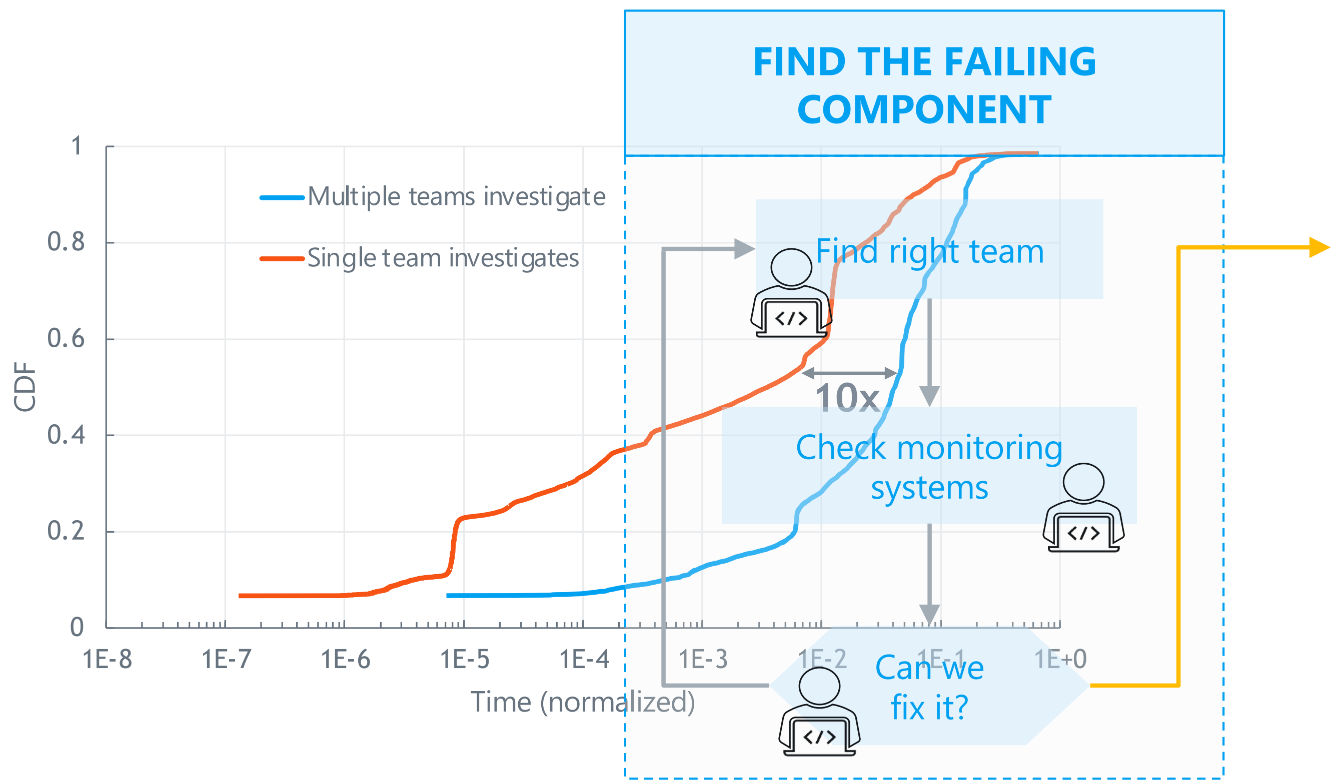
Incidents can and do happen

Number of public incidents between February to July 2020	69	21
Maximum resolution time	14 h 12 m	19 h 49 m
Average resolution time	4 h 40 m	5 h 28 min

Life cycle of an incident

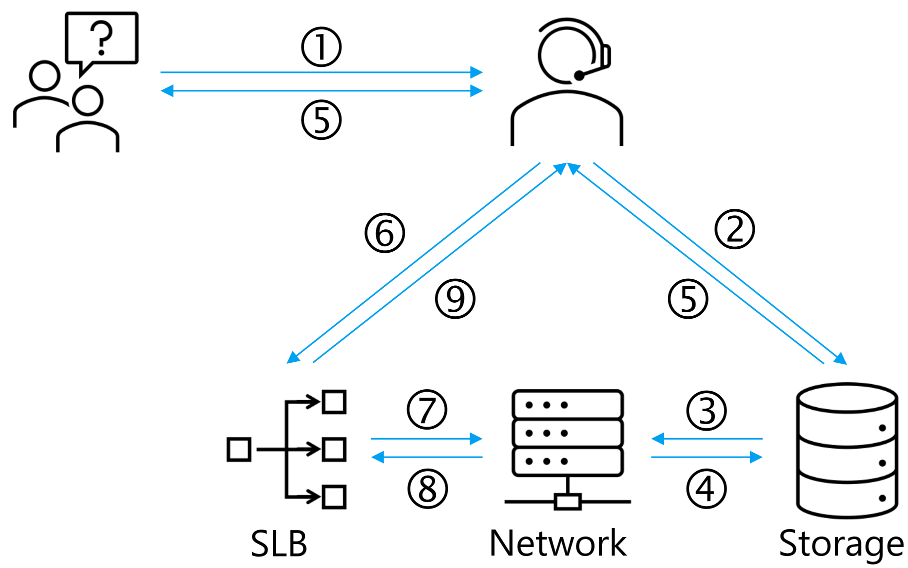


Finding the right team is time consuming



Example incident: storage problem

1. Can't write to storage!
2. Must be storage issue
3. Storage is good, network must be slow
4. No congested links
5. Need more information from customer
6. Connection fail to init, SLB failing
7. SLB is good, network must be dropping
8. Packet is reaching to SLB
9. Customer opens too many connections and exhaust SNAT pool, behavior is expected

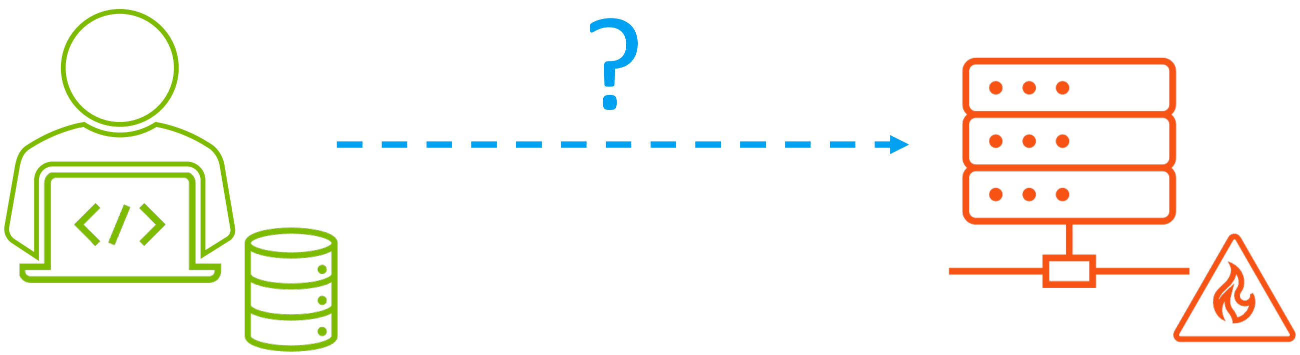


Why multiple teams get involved?

Studied 200 misrouted incidents in Azure

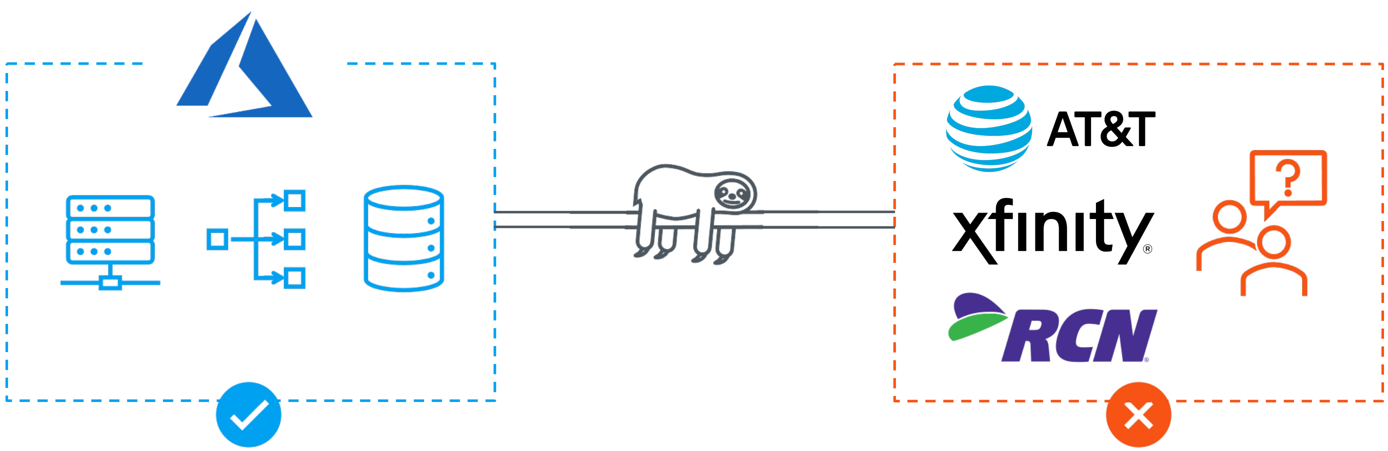
1. Lack of domain knowledge

- Storage team doesn't know network is functioning or not
- Team level dependencies are hard to reason about



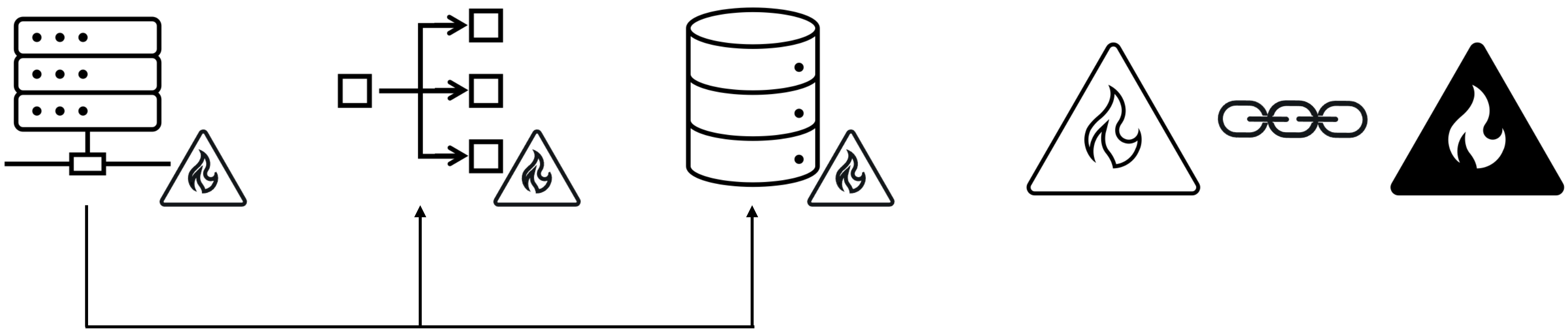
2. No cloud teams are responsible, more misrouting

- ISP or customer outside the cloud is experiencing issues



3. Concurrent incidents

▷ One failure causes multiple incidents in multiple teams

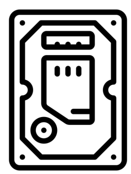


How to reduce misrouting?

Existing solutions

Application specific
diagnosis system

NetPoirot
[SIGCOMM-16]

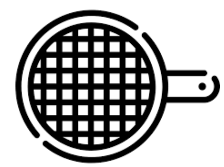


DeepView
[NSDI-18]

Sherlock
[SIGCOMM-07]

Too many applications in the data center

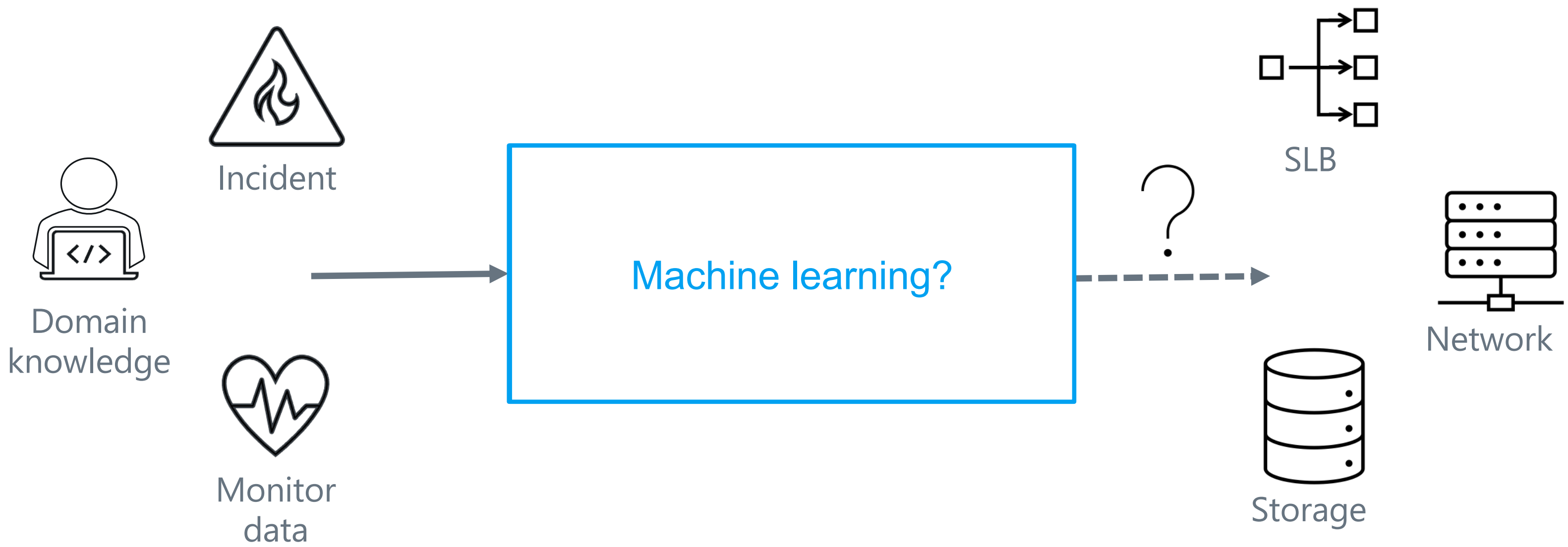
Natural language
processing



NetSieve
[NSDI-13]

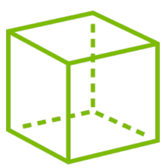
Ignores essential domain knowledge

Incident routing problem revisit



Solve the whole problem at once?

- Hard to build a single, monolithic incident routing system



Curse of dimensionality

Huge feature vector with no enough training examples

Uneven instrumentation

A subset of teams will always have gaps in monitoring

Constantly changing

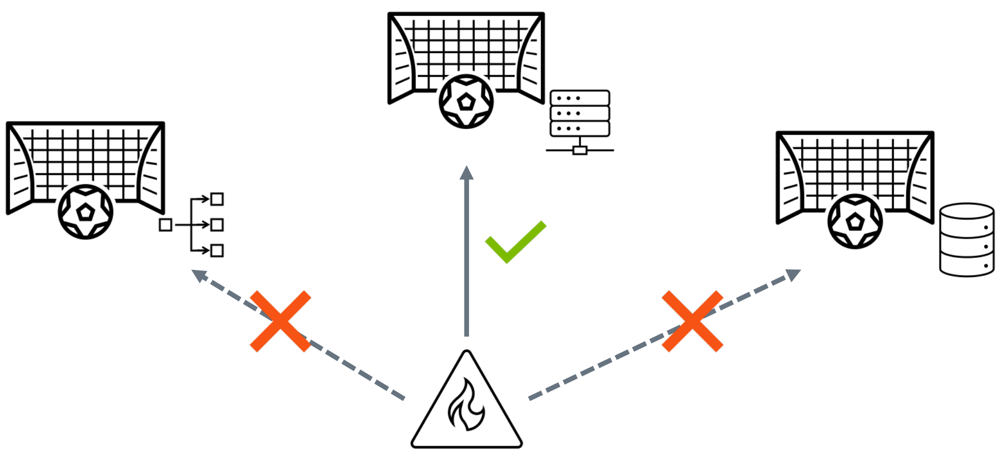
Stale components and monitors

Limited visibility

Hard to understand appropriate feature set for each team

Scout: team-specialized ML-assisted gatekeeper

- "Is my team responsible for the incident?"

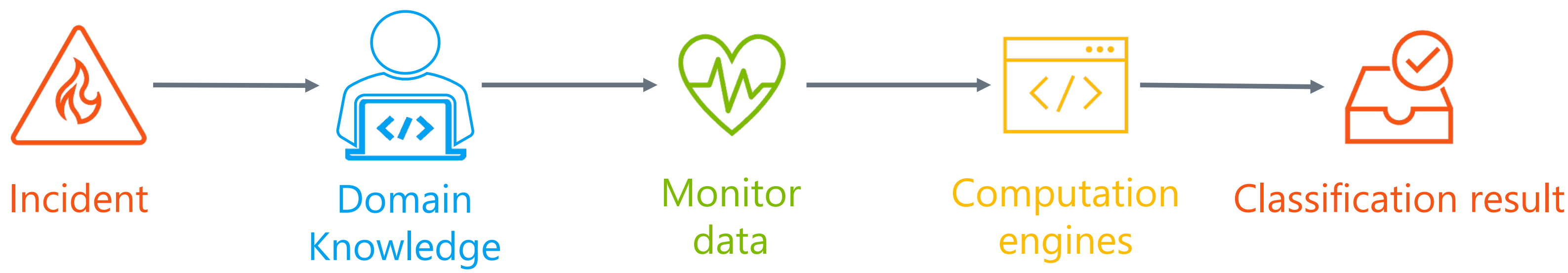


One team, one scout

Leverage domain knowledge

Evolve independently

Scout design



Physical networking team

Scope

Every switch & router in DC

11 Monitor systems

PingMesh, Everflow, NetBouncer, etc.

Statistics

58% incidents investigated by PhyNet went through multiple teams

97.6 hours per day wasted on unnecessary investigations

CHALLENGE 1

- *How to process huge amount of monitoring data?*

Millions of devices in the Cloud

Incident guided investigation

Incident Description | Devices | Monitor data

"Server X.c10.dc3 is experiencing problem connecting to storage cluster c4.dc1"

Server: X.c10.dc3
Cluster: c4.dc1

CPU Usage
Link loss rate
Ping latency

CHALLENGE 2

▷ *How to create a feature vector out of the monitoring data?*

Different incidents have variable number of devices

Mixed types of monitoring data (event-driven vs time series)

How to build a fixed width feature vector?

- Per-component feature
  - Event: count number of events during the incident period
  - Time-series: normalize and calculate statistics (percentiles, average, etc.)
- Multiple components
  - Compute statistics across multiple components (percentiles, average, etc.)

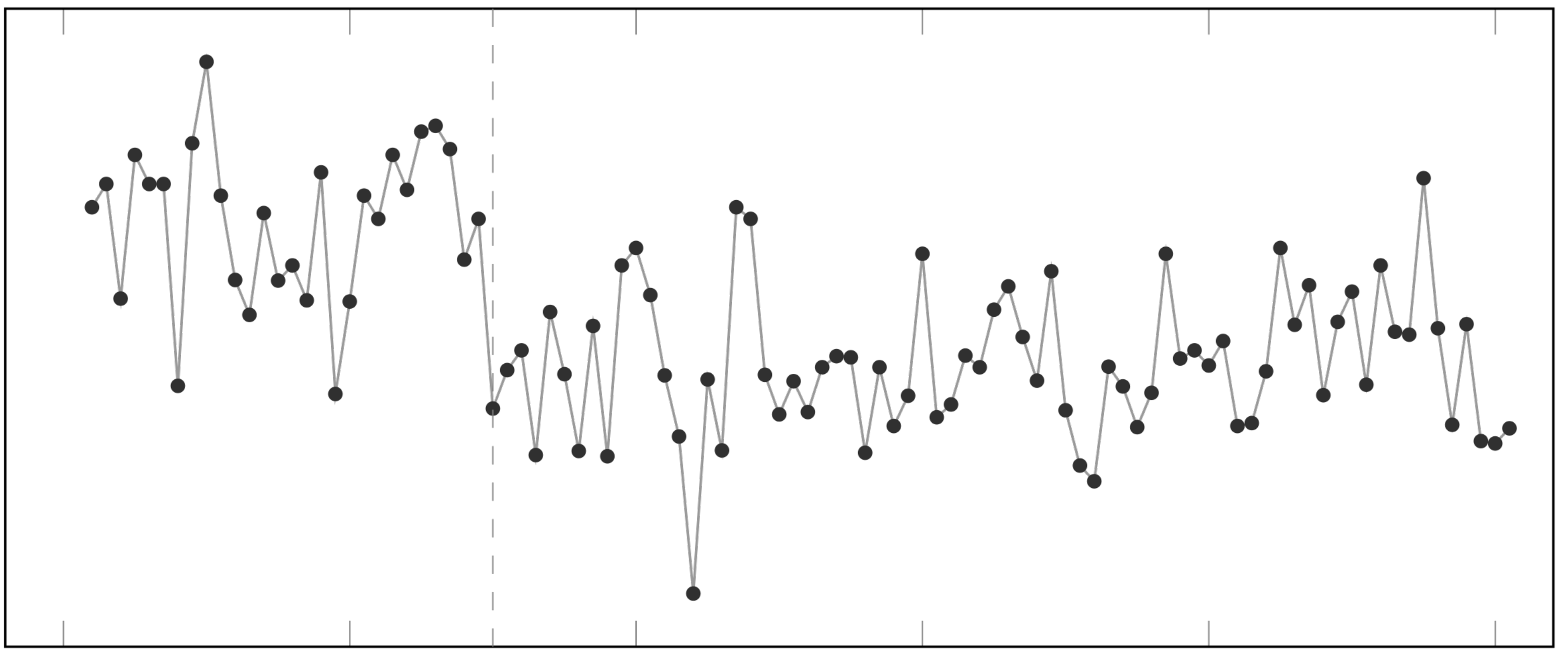
CHALLENGE 3

- *Which computation engine?*

Supervised learning：random forest

- Learns based on history incidents, high accuracy
- Low accuracy on new incidents
- Interpretable, able to provide more insights

Change point detection for new incidents



Change point detection for new incidents

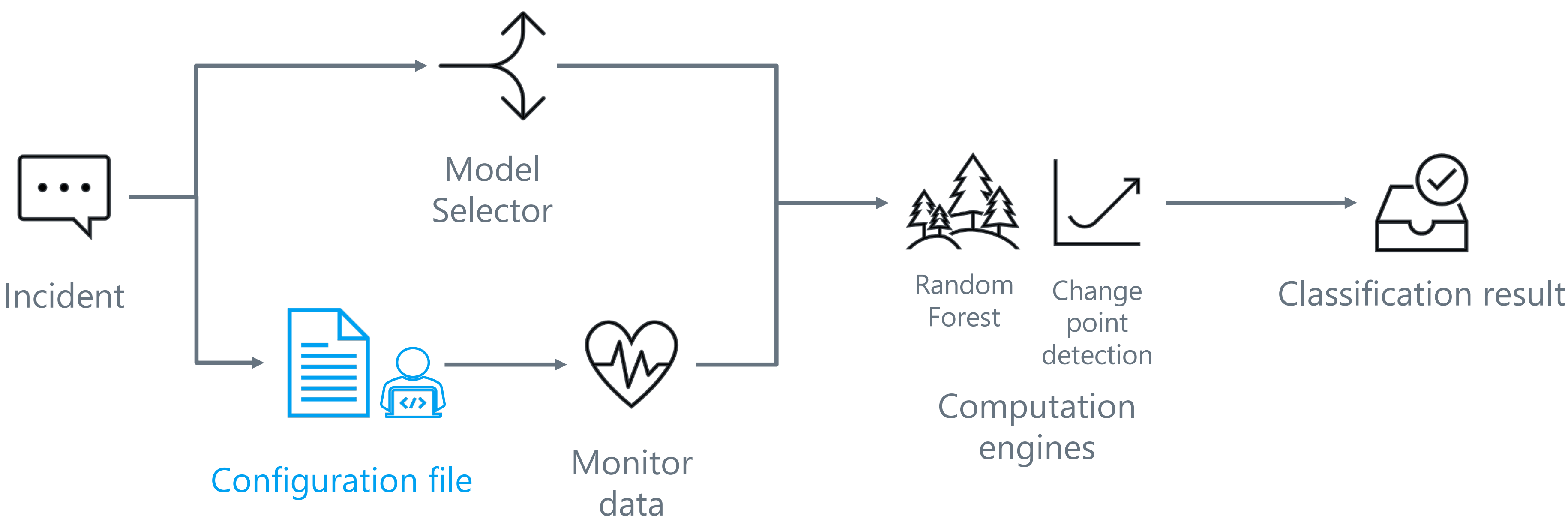
- Easy to compute
- Higher accuracy on new incidents
- Low accuracy on old incidents

Model selector

Incident itself tells whether it is new or not

Use meta-learning to identify new incidents

The anatomy of a Scout



Evaluation

Evaluation setup

DATASET

9 months of incidents in Azure

Randomly split into training and testing set

LABEL

Whether incident is resolved by PhyNet

BASELINE

Current incident routing system without Scout

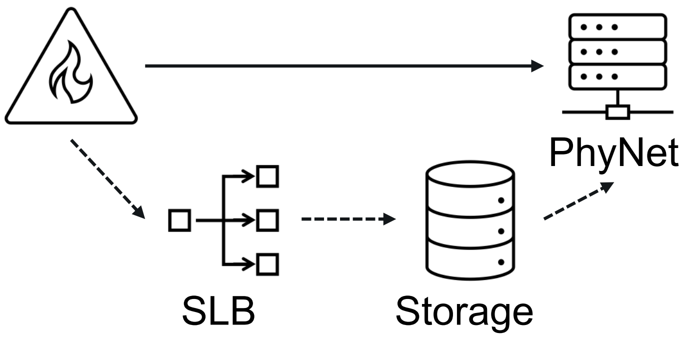
Runbooks, past-experience, NLP based routing system

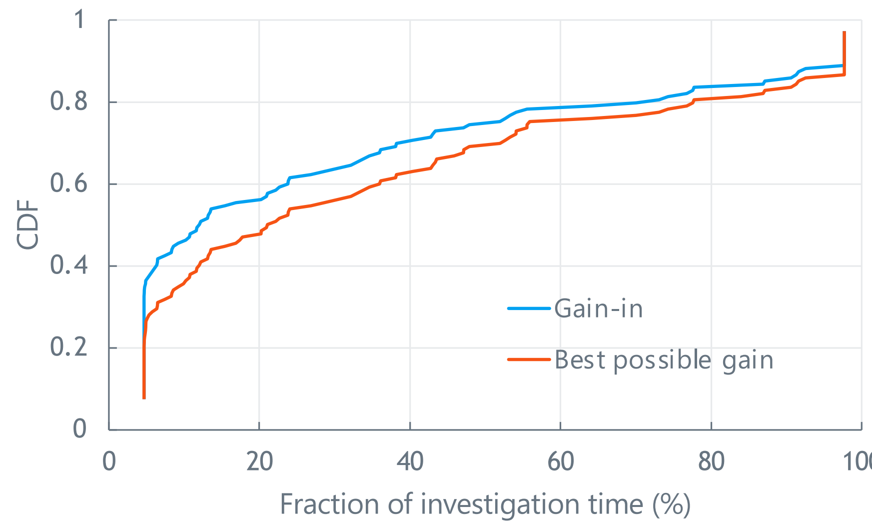
Overall performance

	Precision	Recall	F1-Score
Baseline	87.2%	91.9%	0.89
PhyNet Scout	97.5%	97.7%	0.98
Delta	10.3%	5.8%	0.09

10% improvement in accuracy means significant reduction in investigation time

Benefit of the PhyNet Scout

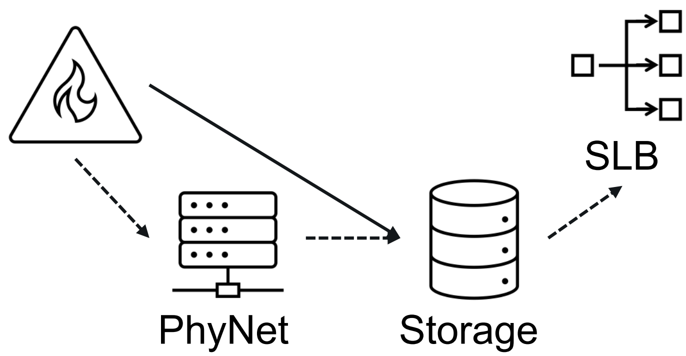




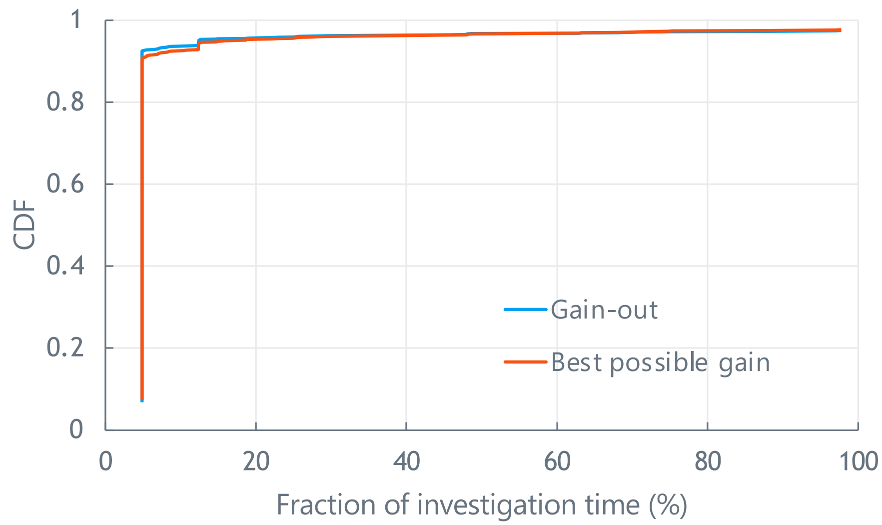
Gain in
Send incident to PhyNet directly

Save more than 20% of the total investigation time in 40% of incidents

Benefit of the PhyNet Scout



Gain out
Reject incident to PhyNet



Close to the best possible gain

Conclusion

- Incident misrouting is the main challenge for maintaining service level objectives in the cloud
- Scout: a distributed team-specialized gate-keeper can reduce investigation time.

This talk

- DETER: Record-and-Replay for TCP
  - Collect detailed yet lightweight packet information at scale

- Scouts: Domain-customized incident routing
  - Automatically direct incident tickets to the right team

Future Directions in Diagnosis

- New application trends
  - Ultra low latency: the killer microseconds
  - Complex structure, especially with cross-layer design
  - Diagnosis is increasingly important for performance optimization
- Open questions
  - How to collect fine-time scale events at large scale?
  - How to tear apart causal relations across layers, across components, across applications?
  - Data driven approaches for diagnosis
  - Customized diagnosis tools for new applications