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 $O(10\text{us})$
- flash $O(10\text{us})$, NVM $O(1\text{us})$
- RDMA $O(1\text{us})$
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

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 $\geq$ latency of slowest component (Tail latency!!!)
  - small blips on individual flows cause delays
  - touching more machines increases likelihood of delays
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
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
- DETERministic Record and Replay

Runtime record $\equiv$ Data for diagnosis
Runtime record $\prec$ Data for diagnosis
DETER overview

**Lightweight record**
- Run continuously
- On all hosts

**Deterministic replay**
- Capture packets/counters
- Trace executions
- Iterative diagnosis

10.0.0.1:80 - 20.0.0.1:1234 has long latency

10.0.0.1:80 -> 20.0.0.1:1234
20.0.0.1:1234 -> 10.0.0.1:80

× N
Lightweight record  Deterministic replay
Intuition for being lightweight

Lightweight record

Record socket calls

Deterministic replay

Automatically generate packets

socket call \rightarrow TCP \rightarrow TCP \rightarrow sbarang call
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.
Challenge 1: butterfly effect

Sending time variation → Switch action variation → TCP behavior variation

μs-level:
Clock drift, context switching, kernel scheduling, cache state

Runtime

Replay

Sending time variation
Cong_win/=2
sock call → TCP → drop

Cong_win++
sock call → TCP

Switch action variation

TCP behavior variation

Cong_win++
sock call → TCP → enqueue

Cong_win/=2
sock call → TCP → drop

1 us late
Challenge 1: butterfly effect

- Sending time variation
- Switch action variation
- TCP behavior variation

Runtime

- Cong_win/=2
- socket call → TCP

Replay

- Cong_win++
- socket call → TCP
- Cong_win/=2
- socket call → TCP
- enqueue
- drop

Butterfly effect
Reduce Butterfly Effect

- Sending time variation
- Switch action variation
- TCP behavior variation

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

<table>
<thead>
<tr>
<th>Flow size (MB)</th>
<th>&lt;0.1</th>
<th>[0.1, 1]</th>
<th>[1, 10]</th>
<th>&gt;10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTO</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>FR</td>
<td>74</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Delayed ACK</td>
<td>0</td>
<td>0</td>
<td><strong>18</strong></td>
<td>0</td>
</tr>
<tr>
<td>Rwnd=0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Slow start</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>Company A</th>
<th>Company B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of public incidents</td>
<td>69</td>
<td>21</td>
</tr>
<tr>
<td>between February to July 2020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum resolution time (h:m)</td>
<td>14:12</td>
<td>19:49</td>
</tr>
<tr>
<td>Average resolution time (h:m)</td>
<td>4:40</td>
<td>5:28</td>
</tr>
</tbody>
</table>
Life cycle of an incident

**REPORT INCIDENT**
- Monitoring system
- Customer reports

**FIND THE FAILING COMPONENT**
- Find right team
- Check monitoring systems
- Can we fix it?

**DIAGNOSE AND FIX PROBLEM**
- Find problematic device
- Understand the cause
- Fix
Finding the right team is time consuming

- **CDF**
- **Time (normalized)**

**FIND THE FAILING COMPONENT**

- Multiple teams investigate
- Single team investigates

- Find right team
- Check monitoring systems
- Can we fix it?

42
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

Machine learning?

Domain knowledge

Monitor data

Incident

SLB

Network

Storage
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

Incident → Domain Knowledge → Monitor data → Computation engines → Classification result
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:

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

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

Monitor data:

CPU Usage
Link loss rate
Ping latency
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

- Incident
- Configuration file
- Model Selector
- Monitor data
- Random Forest
- Change point detection
- Computation engines
- Classification result
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

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>87.2%</td>
<td>91.9%</td>
<td>0.89</td>
</tr>
<tr>
<td><strong>PhyNet Scout</strong></td>
<td>97.5%</td>
<td>97.7%</td>
<td>0.98</td>
</tr>
<tr>
<td><strong>Delta</strong></td>
<td>10.3%</td>
<td>5.8%</td>
<td>0.09</td>
</tr>
</tbody>
</table>

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