Motivation: Root cause analysis

- Networks can (and frequently do!) have bugs
- We need a good debugger!

From: alice@xyz.com
To: Admin (bob@xyz.com)
Title: Help!

My server is receiving suspicious traffic from 4.3.2.0/24--it should have been sent to the low-security server. Packets from 4.3.3.0/24 are still being routed correctly. Can you help?
Debugging networks with provenance

- Existing debuggers tell us what happened
  - Example: NetSight [NSDI’14]
- Provenance offers a richer explanation
  - Example: Y! [SIGCOMM’14]
Problem: The explanation can be too big!

Packet arrives at the wrong server

Symptom

C received packet
B sent packet
Rule match on B

Rule 7: Next-hop=port2

Root cause
What can we do?

Idea: Reason about the differences between the symptom and the reference

From: alice@xyz.com
To: Admin (bob@xyz.com)
Title: Help!

My server is receiving suspicious traffic from 4.3.2.0/24—it should have been sent to the low-security server. Packets from 4.3.3.0/24 are still being routed correctly. Can you help?

Outages mailing list
Sept.–Dec. 2014: 66% have references!
Differential provenance

- **Input:** a bad symptom and a good reference
- Debugger reasons about the differences
- **Output:** root cause

Rule 7’s next hop is wrong!
Outline

• Motivation: Root cause analysis

• Differential provenance
  • Background: Provenance
  • Strawman solution
  • Algorithm

• Evaluation
  • Prototype implementation
  • Usability
  • Query processing speed
  • Complex network diagnostics

• Conclusion
Background: Provenance

- Provenance tracks causal connections between network events and state [ExSPAN-SIGMOD’10]
  - Provenance graph: Vertexes $\rightarrow$ event/state. Edge $\rightarrow$ causality
  - Provenance tree: Recursive explanation of an event/state
• Strawman solution: Find vertexes that are different in the two trees
• Problem: The **diff can be larger than the individual trees**!
• Observation: The diff can be larger than the individual trees
• Reason: “Butterfly effect”
  • A small initial difference can lead to drastically different events later on
Outline

✓ • Motivation: Root cause analysis
✓ • Key insight
✓ • Differential provenance
  • Background: Provenance
  • Strawman solution
✓ • Algorithm
✓ • Evaluation
  • Prototype implementation
  • Usability
  • Query processing speed
  • Complex network diagnostics
✓ • Conclusion
Algorithm: Refinement #1

- This approach finds only the (small) initial differences
  - The (potentially large) consequences are ignored
Algorithm: Refinement #1 (Cont’d)

- Approach: Roll back the execution, change the first faulty node, and roll forward again to align the trees
How to preserve crucial differences?

- Problem: There are differences that we need to preserve
  - Example: The packets whose provenance we are looking at
Solution: Establish equivalence

• Establish an equivalence relation between the trees
  • Example: IP addresses 4.3.2.1 and 4.3.3.1
  • Values on the trees can be identical, equivalent, or different
• Goal: Make the trees equivalent, not necessarily identical!
Algorithm: Refinement #2

- Roll back the execution to a divergence point.
- Change the faulty node to be like the correct node.
- Roll forward the execution to align the trees.

• Benefit: Preserves the crucial differences between the trees.
Establishing and propagating equivalence

- Start with an initial equivalence relation between the packets
  - Establish a mapping between packet fields that are different
  - Keep track of the mapping while going up the tree
    - Stop at the first non-equivalent(!) node
    - More general approach: taint analysis
Propagating equivalence with taints

- **Approach:**
  - Create *taints* for equivalent fields
  - Propagate taints up the tree
  - Repeat until we find a non-equivalent node
Changing the faulty node

- Change the faulty node to its equivalent: Pkt(4.3.2.1)@C → Pkt(4.3.2.1)@E
  - Have dependent nodes → Create their equivalents recursively
    - Example: Flow(next=C) → Flow(next=E)
  - No dependent nodes → Insert its equivalent
    - Example: Insert Flow(next=E)
- See paper for how to propagate taints in the reverse direction
Problem: Multiple faults

- Problem: There could be more than one difference between the two trees
- Solution: Repeat until the trees are completely aligned
Refinement #3: Final algorithm

Roll back the execution to a divergence a non-equivalent point

Change the faulty node to be like the correct node its equivalent

Roll forward the execution to align the trees

Completely equivalent?

YES

Output changes

NO
Rolling forward the execution

- Roll the execution forward to align the trees
  - Output the accumulated change(s): Flow(next=C) → Flow(next=E)!

Bad provenance

Reference provenance
Outline

• Motivation: Root cause analysis
• Differential provenance
  • Background: Provenance
  • Strawman approach
  • Algorithm
• Evaluation
  • Prototype implementation
  • Usability
  • Query processing speed
  • Complex network diagnostics
• Conclusion
Prototype implementation: DiffProv

• Mostly focuses on Network Datalog (NDlog) [CACM ’2009] programs, where provenance is easy to see
• NetCore [NSDI ’13] programs are also supported
• Applicable beyond SDN: Hadoop MapReduce
• Integrated with Mininet + the Beacon controller; based on Rapidnet
Evaluation: Overview

✓ Q1: How well does DiffProv find the root cause?

Q2: How much overhead does DiffProv incur at runtime?

✓ Q3: How quickly does DiffProv answer diagnostic queries?

Q4: How well does DiffProv recognize bad reference events?

✓ Q5: How well does DiffProv work for complex networks?
Experimental setup

• We adapted seven diagnostic scenarios:
  • SDN1: Broken flow entry [Empr.Soft.Eng.’09]
  • SDN2: Multi-controller inconsistency [CoNEXT’14]
  • SDN3: Unexpected rule expiration [P2P’13]
  • SDN4: Multiple faulty entries [Empr.Soft.Eng.’09]
  • Complex network diagnostics [CoNEXT’12]
  • MR1: Configuration changes [Industry collaborators]
  • MR2: Code changes [Industry collaborators]

• Baseline: Y!, a provenance debugger without reference support [SIGCOMM’14]
How well does DiffProv find the root cause?

Provenance (symptom)  201 nodes
Provenance (reference)  156 nodes
Naïve diff  278 nodes
DiffProv  1 node!
How well does DiffProv find the root cause? (Cont’d)

<table>
<thead>
<tr>
<th>Query</th>
<th>SDN1</th>
<th>SDN2</th>
<th>SDN3</th>
<th>SDN4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of faults</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DiffProv</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Reference</td>
<td>156</td>
<td>156</td>
<td>156</td>
<td>201/201</td>
</tr>
<tr>
<td>Symptom</td>
<td>201</td>
<td>156</td>
<td>201</td>
<td>156/145</td>
</tr>
<tr>
<td>Plain tree diff</td>
<td>278</td>
<td>238</td>
<td>74</td>
<td>278/218</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
<th>MR1-D</th>
<th>MR2-D</th>
<th>MR1-I</th>
<th>MR2-I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. of faults</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DiffProv</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Reference</td>
<td>1051</td>
<td>1001</td>
<td>588</td>
<td>588</td>
</tr>
<tr>
<td>Symptom</td>
<td>1051</td>
<td>848</td>
<td>588</td>
<td>438</td>
</tr>
<tr>
<td>Plain tree diff</td>
<td>164</td>
<td>306</td>
<td>240</td>
<td>216</td>
</tr>
</tbody>
</table>

- DiffProv finds one or two nodes (the faulty rules or MapReduce configuration entries), which are the actual root cause
How long does DiffProv take to find the root causes?

- DiffProv answered most of our queries within one minute!
How well does DiffProv work in complex networks?

- Setup: larger topology, complex config, background traffic
  - ‘Forwarding error’ scenario [ATPG-CoNEXT’12]
  - Stanford network: 757,000 forwarding entries and 1,500 ACL rules
  - Multiple faults: Injected 20 additional faulty entries
  - Background traffic: 12GB traffic, 69 protocol types

- Results:
  - DiffProv: the faulty entry for misconfigured subnet – one node
  - Identified the root cause despite heavy interference

- Why is DiffProv not confused by the interference?
  - Provenance captures causality, not merely correlations!
Summary

• Debugging networks is hard
  • Need good debuggers to find root causes!

• Key insight: We can use reference events
  • We often have more information than we are using
  • Idea: Reason about the differences between bad events and reference events

• Approach: Differential provenance
  • We have built a prototype debugger for SDNs
  • Applicable to other distributed systems beyond SDNs

• Result: Very precise diagnostics
  • Differential provenance can often identify a single root cause

Thank you!