NetCheck: Network Diagnoses from Blackbox Traces

Yanyan Zhuang*\^, Eleni Gessiou*, Fraida Fund*, Steven Portzer@, Monzur Muhammad^, Ivan Beschastnikh^, Justin Cappos*

(*)New York University, (^)University of British Columbia, (@)University of Washington
How do we troubleshoot this?

How does Esmeralda know how to fix this?
Goal

- Find bugs in networked applications
  - Large complex unknown applications

- Large complex unknown networks

- Understandable output / fix
Motivation

Chrome Client

Apache Server
Motivation

Chrome Client

Apache Server

Different traffic (ICMP)
Often different result

ping
Motivation

Chrome Client

Requires detailed protocol / app knowledge

Apache Server

packet capture
Motivation

Need a model per application

Model apps
Magpie, Xtrace, Pip...

Chrome Client

Apache Server
Motivation

Chrome Client

Network Config Analysis

Header Space Analysis, etc.

Apache Server

Model & Config

Model & Config

Model & Config

Model & Config
Motivation

Need detailed network knowledge HW + config

Chrome Client

Network Config Analysis

Model & Config

Apache Server

Model & Config

Model & Config

Model & Config
Motivation

Chrome Client

Apache Server
NetCheck

Chrome Client

programmer

programmer

Apache Server

THE ARPA NETWORK
DEC 1969
4 NODES
Conceptual Sketch of Original Internet
Outline

- Motivation
- **NetCheck Overview**
- Trace Ordering
- Network Model
- Fault Classification
- Results / Conclusion
NetCheck overview

Application
  ↓
Traces
  ↓
NetCheck
  ↓
Likely Faults
  ↓
Fail
NetCheck overview

Application

\[\downarrow\]

Traces

\[\downarrow\]

NetCheck

\[\downarrow\]

Likely Faults

Fail

ktrace

strace
NetCheck overview

Application

↓

Traces

↓

NetCheck

↓

Likely Faults

Input

Host Traces

Ordering Algorithm

syscall

simulation result

Network Model

Diagnoses Engine

simulation state errors

Diagnosis Output
NetCheck overview

Application

↓

Traces

↓

NetCheck

↓

Likely Faults

Network Configuration Issues

Traffic Statistics

Problem Detected
Outline

- Motivation
- NetCheck Overview
- Trace Ordering
- Network Model
- Fault Classification
- Results / Conclusion
Traces

Series of locally ordered system calls
Don’t want to modify apps or use a global clock
Gathered by strace, ktrace, systrace, truss, etc.
Call arguments and “return values”

<table>
<thead>
<tr>
<th>Call Function</th>
<th>Arguments</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket()</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>bind(3, ...)</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>listen(3, 1)</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>accept(3, ...)</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>recv(4, &quot;HTTP&quot;, ...)</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>close(4)</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Call arguments and return values

Return buffer
What we see is this:

Node A
1. socket() = 3
2. bind(3, ...) = 0
3. listen(3, 1) = 0
4. accept(3, ...) = 4
5. recv(4, "Hello", ..) = 5
6. close(4) = 0

Node B
1. socket() = 3
2. connect(3, ...) = 0
3. send(3, "Hello", ..) = 5
4. close(3) = 0

- one trace per host
- local order but no global order

Q: how do we reconstruct what really happened?
What we want is this

A1. socket() = 3
B1. socket() = 3
A2. bind(3, .. .) = 0
A3. listen(3, 1) = 0
B2. connect(3,...) = 0
A4. accept(3, ...) = 4
B3. send(3, "Hello", ...) = 5
A5. recv(4, "Hello", ...) = 5
B4. close(3) = 0
A6. close(4) = 0

The ground truth
What we want is this

A1. socket() = 3
B1. socket() = 3
A2. bind(3, .. .) = 0
A3. listen(3, 1) = 0
B2. connect(3,...) = 0
A4. accept(3, ...) = 4
B3. send(3, "Hello", ...) = 5
A5. recv(4, "Hello", ...) = 5
B4. close(3) = 0
A6. close(4) = 0

The ground truth

Goal: find an equivalent interleaving
Observation 1: Order Equivalence

Node A
1. `socket()` = 3
2. `bind(3, ...)` = 0
3. `listen(3, 1)` = 0
4. `accept(3, ...)` = 4
5. `recv(4,"Hello", ..)` = 5
6. `close(4)` = 0

Node B
1. `socket()` = 3
2. `connect(3,...)` = 0
3. `send(3, "Hello",.)` = 5
4. `close(3)` = 0

- one trace per host
- local order but no global order

Q: how do we reconstruct what really happened?

The `socket()` calls are not visible to the other side

Some orders are equivalent!
Observation 2: Return Values Guide Ordering

Node A
1. socket() = 3
2. bind(3, ...) = 0
3. listen(3, 1) = 0
4. accept(3, ...) = 4
5. recv(4,"Hello", ..) = 5
6. close(4) = 0

Node B
1. socket() = 3
2. connect(3,...) = 0
3. send(3, "Hello",..) = 5
4. close(3) = 0

- one trace per host
- local order but no global order

Q: how do we reconstruct what really happened?
Return values guide ordering

One valid ordering: all syscalls returned successfully.

A2. bind(3, ...) = 0
A3. listen(3, 1) = 0
B2. connect(3, ...) = 0

A second valid ordering: connect failed with ECONNREFUSED.

A2. bind(3, ...) = 0
B2. connect(3, ...) = -1, ECONNREFUSED
A3. listen(3, 1) = 0

A call’s return value may-dep-end-on a remote call’s action

Result indicates order of calls
Deciding call order

full set of may-depend-on relations
Ordering Algorithm

Input traces

Algorithm process

Output Ordering

Host A trace:
A1. socket(...) = 4  B1. socket(...) = 3
A2. bind(4, ...) = 0  B2. connect(3, ...) = 0
A3. listen(4, 1) = 0  B3. send(3, "Hello", ...) = 5
A4. accept(4, ...) = 6
A5. recv(6, "Hello", ...) = 5
Ordering Algorithm

Input traces

A
- socket
  - bind
  - listen
  - accept
  - recv

B
- socket
  - connect
  - send

Algorithm process

Try socket on host A: accepted

Output Ordering

A
- socket

---

Host A trace:
A1. socket(...) = 4
A2. bind(4, ...) = 0
A3. listen(4, 1) = 0
A4. accept(4, ...) = 6
A5. recv(6, "Hello", ...) = 5

Host B trace:
B1. socket(...) = 3
B2. connect(3, ...) = 0
B3. send(3, "Hello", ...) = 5
Ordering Algorithm

Input traces

A
- listen
- accept
- recv

B
- connect
- send

Algorithm process

Try connect on host B: rejected

Output Ordering

A
- socket

B
- socket

A
- bind

Host A trace:
A1. socket(…) = 4
A2. bind(4, …) = 0
A3. listen(4, 1) = 0
A4. accept(4, …) = 6
A5. recv(6, "Hello", …) = 5

Host B trace:
B1. socket(…) = 3
B2. connect(3, …) = 0
B3. send(3, "Hello", …) = 5
Ordering Algorithm

Input traces
A
- listen
- accept
- recv

B
- connect
- send

Algorithm process
Try listen on host A: accepted

Output Ordering
A
- socket
- listen
B
- socket

Host A trace:
A1. socket(...) = 4
A2. bind(4, ...) = 0
A3. listen(4, 1) = 0
A4. accept(4, ...) = 6
A5. recv(6, "Hello", ...) = 5

Host B trace:
B1. socket(...) = 3
B2. connect(3, ...) = 0
B3. send(3, "Hello", ...) = 5
Ordering Algorithm

Input traces

TCP BUFFER: “”

Algorithm process

Try recv on host A: rejected

Output Ordering

Host A trace:
A1. socket(...) = 4
A2. bind(4, ...) = 0
A3. listen(4, 1) = 0
A4. accept(‘’) = 6
A5. recv(6, “Hola!” ...) = 5

Host B trace:
B1. socket(...) = 3
B2. connect(3, ...) = 0
B3. send(3, "Hello", ...) = 5
Ordering Algorithm

Input traces

A
recv

B
send

TCP BUFFER: ""

Algorithm process

Try send on host B: accepted

Output Ordering

A
socket

B
socket

A
bind

A
listen

B
connect

A
accept

A

send

Host A trace:
A1. socket(...) = 4
A2. bind(4, ...) = 0
A3. listen(4, 1) = 0
A4. accept(...) = 6
A5. recv(6, "Hola", ...) = 5

Host B trace:
B1. socket(...) = 3
B2. connect(3, ...) = 0
B3. send(3, "Hello", ...) = 5

39
Ordering Algorithm

Input traces
A
recv

B
None

Algorithm process
Try send on host B: accepted

TCP BUFFER: “Hello”

Output Ordering

A
socket

B
socket

A
bind

A
listen

B
connect

A
accept

B
send

Host A trace:
A1. socket(…) = 4
A2. bind(4, ...) = 0
A3. listen(4, 1) = 0
A4. accept(“”) = 6
A5. recv(6, “Hola!” ... = 5

Host B trace:
B1. socket(…) = 3
B2. connect(3, ...) = 0
B3. send(3, "Hello", ...) = 5
Ordering Algorithm

Input traces

TCP BUFFER: “Hello”

Algorithm process

Try recv on host A: Fatal Error

Output Ordering

socket

socket

bind

listen

connect

accept

send

recv

recv

A

B

A

A

B

A


Host A trace:
A1. socket(…) = 4
A2. bind(4, ...) = 0
A3. listen(4, 1) = 0
A4. accept(…) = 6
A5. recv(6, “Hola!” ...) = 5

Host B trace:
B1. socket(…) = 3
B2. connect(3, ...) = 0
B3. send(3, “Hello”, ...) = 5
Outline

- Motivation
- NetCheck Overview
- Trace Ordering
- **Network Model**
- Fault Classification
- Results / Conclusion
Network Model

- Simulates invocation of a syscall
  - datagrams sent/lost
    - reordering / duplication is notable
  - track pending connections
  - buffer lengths and contents
    - send -> put data into buffer
    - recv -> pop data from buffer

- Simulation outcome
  - \textit{Accept} → can process (correct buffer)
  - \textit{Reject} → wrong order (incomplete buffer)
  - \textit{Permanent reject} → abnormal behavior (incorrect buffer)
Network Model

- Simulates invocation of a syscall
- Capture programmer assumptions
  - Assumes a simplified network view
    - Assume transitive connectivity
    - Little, random loss
    - No middle boxes
  - Assume uniform platform
    - Flag OS differences
How Model Return Values Impact Trace Ordering

- Blackbox Tracing mechanism

Trace Ordering: linear running time (total trace length) * number of traces
Outline

- Motivation
- NetCheck Overview
- Trace Ordering
- Network Model
- Fault Classification
- Results / Conclusion
Fault Classifier

- Goal: Decide what to output
- Problem: Show relevant information
- Fault classifier: global (rather than local) view
  - uncovers high-level patterns by extracting low-level features
    - Examples: middleboxes, non-transitive connectivity, MTU, mobility, network disconnection
  - All look like loss, but have different patterns in the context of other flows
Fault Classifier

- Options to show different levels of detail
- Network admins / developers
  - detailed info
- End users
  - Classification
  - Recommendations

Network Configuration Issues

Traffic Statistics

Problem Detected
Outline

• Motivation
• NetCheck Overview
• Trace Ordering
• Network Model
• Fault Classification
• Results / Conclusion
Evaluation: Production Application Bugs

- Reproduce reported bugs from bug trackers (Python, Apache, Ruby, Firefox, etc.)
  - A total of 71 bugs
  - Grouped into 23 categories
    - Virtualization incurred/portability bugs
    - SO_REUSEADDR behaves differently across OSes
    - accept inherit O_NONBLOCK
    - ...
  - Correct analysis of >95% bugs
Evaluation: Observed Network Faults

- Twenty faults observed in practice on a live network
  - MTU bug
    - Intermediary device
  - Port forward
    - Traffic sent to non-relevant addresses
  - Provide supplemental info
    - packet loss
    - buffers being closed with data in
  - 90% of cases correctly detected
General Findings in Practice

- **Middle boxes**
  - Multiple unaccepted connections
    - client behind NAT in FTP
- **TCP/UDP**
  - non-transitive connectivity in VLC
- **Complex failures**
  - VirtualBox send data larger than buffer size
  - Pidgin returned IP different from bind
  - Skype NAT + close socket from a different thread
- **Used on Seattle Testbed** seattle.poly.edu
NetCheck Performance Overhead

![Graph showing performance overhead for different applications]

- Firefox
- Skype
- Telnet
- SSH
- VLC
Conclusion

Built and evaluated NetCheck, a tool to diagnose network failures in complex apps

● Key insights:
  ○ model the programmer’s misconceptions
  ○ relation between calls $\rightarrow$ reconstruct order

● NetCheck is effective
  ○ Everyday applications & networks
  ○ Real network / application bugs
  ○ No per-network knowledge
  ○ No per-application knowledge

Try it here: https://netcheck.poly.edu/