NetCheck: Network Diagnoses from Blackbox Traces

https://netcheck.poly.edu/

Yanyan Zhuang†‡, Eleni Gessiou†, Steven Portzer*, Fraida Fund†
Monzur Muhammad†, Ivan Beschastnikh‡, Justin Cappos†

† NYU ‡ UBC *

Motivation

Networked application failures

- Challenging to understand and to fix.
- Fail for complex reasons
  - ✧ In-network state;
  - ✧ State at remote end-host, e.g., MTU, NAT, firewalls, IPv6, etc.

Failure diagnoses

- Problem: many popular apps are not open source; network configuration is not available.
- Current solution: ping/traceroute for reachability, but not app-level issues.

Our Solution: NetCheck

- Diagnoses network issues from syscall traces at multiple end-hosts.
- Does not require clock sync, network or app-specific info.

Challenges & Contributions

Challenges

Accuracy: ambiguity in reconstruction.

- ✧ Without clock sync, multiple orderings of end-host syscalls possible. An example:

```
Host A trace:                 Host B trace:
A1. send("hello") = 5         B1. send("hi") = 2
A2. recv("hi")    = 2         B2. recv()     = -1, EWOULDBLOCK
```

(a) Two input host traces

```
Valid ordering 1:
B1. send("hi")     = 2
B2. recv()         = -1, EWOULDBLOCK
A1. send("hello")  = 5
A2. recv("hi")     = 2
```

(b) A valid ordering

```
Valid ordering 2:
A1. send("hello")  = 5
B1. send("hi")     = 2
A2. recv("hi")     = 2
B2. recv()         = -1, EWOULDBLOCK
```

(c) Another valid ordering!

```
A1. send("hello")  = 5
A2. recv("hi")     = 2
B1. send("hi")     = 2
B2. recv()         = -1, EWOULDBLOCK
```

(d) An invalid ordering of (a)

Network complexity: diagnosing issues in real networks.

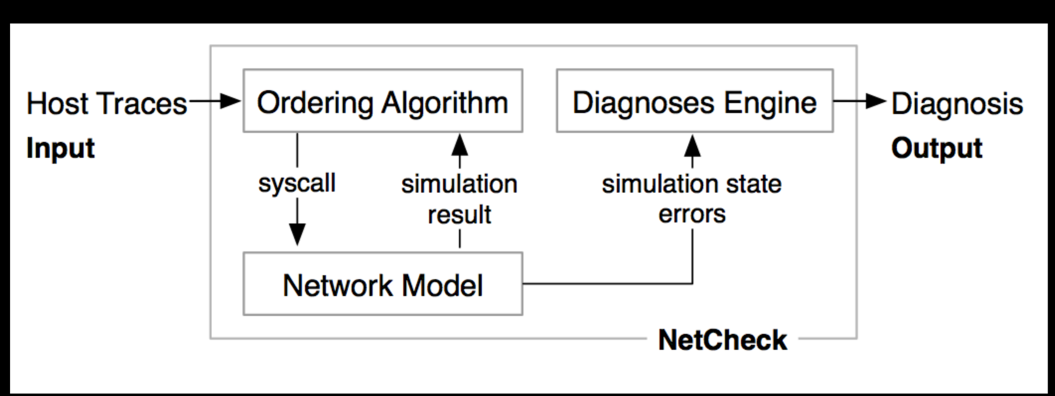
- ✧ Host traces omit information about physical network or environment.

Efficiency: must explore an exponential space of possible orderings.

NetCheck Contributions

- Derive a plausible global ordering as an ***approximation*** for the ground truth.
- Model ***expected simple*** network behavior to identify the unexpected.
- A best-case ***linear time*** algorithm to find a plausible global ordering.

NetCheck Design



(1) Ordering host traces

- Key to efficiency: reconstructs order based on POSIX syscall dependencies.
  - ✧ Dependencies derived from POSIX spec.

(2) Model-based syscall simulation

- Simulates syscalls to find a global order.
- Treats network & application as a blackbox, requires no app-specific info.

```
Host A trace:                          Host B 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, "Hola!", ...)    = 5
```

(e) An example input traces

- Simulates developer-expected network semantics (i.e., the fallacies).
  - ✧ Network model state: connections, buffers, datagrams, etc.
  - ✧ Simulating a syscall results in:
    - Accept;
    - Reject;
    - Permanent Reject.

```
A1. socket(...)             = 4
B1. socket(...)             = 3
A2. bind(4, ...)            = 0
A3. listen(4, 1)            = 0
B2. connect(3,...)          = 0
A4. accept(4, ...)          = 6
B3. send(3,"Hello",...)     = 5
A5. recv(6,"Hola!",...)     = 5
```

(f) A valid global ordering of (e)

(3) Fault diagnoses engine

- Analyzes the model state and simulation errors to derive a diagnosis:
  - ✧ 9 high-level rules.
  - ✧ Make results more meaningful.

```
[Warning] trace A: ('recv_syscall', (1, 'Hola!', 1024, 0))
    => MSG_DONT_MATCH: [Possible Network Misbehavior]
    Message received does not match the data sent by
    the socket.
```

(g) An example diagnoses of (e)

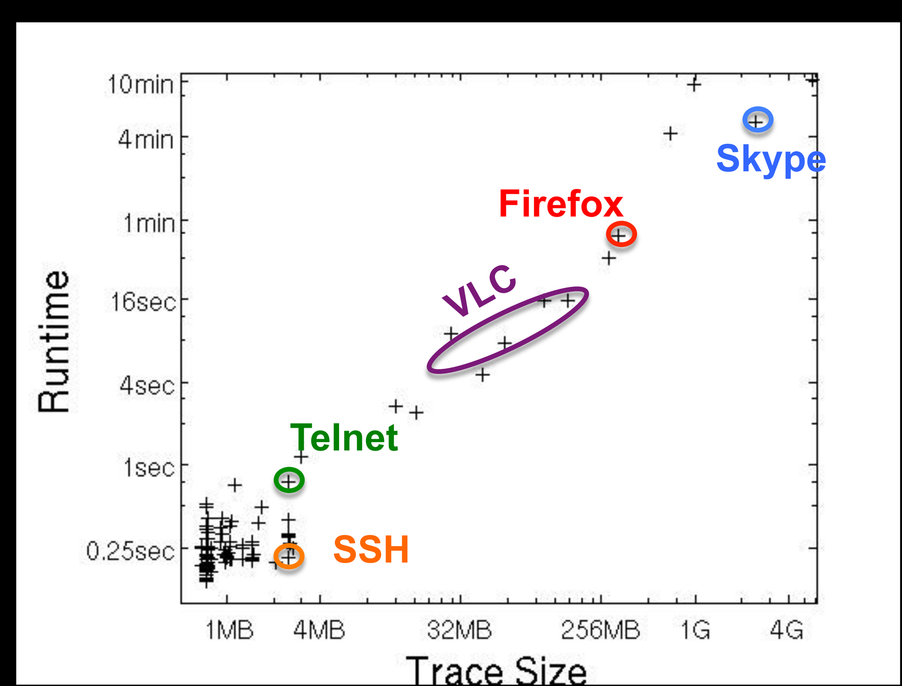
Evaluation

Accuracy

- ***Reproduced known bugs in multiple open source projects***
  - ✧ 46 bugs from public bug trackers of 30 popular projects.
  - ✧ Reproduced issue from each report: 71 traces, 24 categories.
  - ✧ Correctly detected and diagnosed ***95.7%*** of bugs considered.
- ***Diagnosed injected failures in a real network***
  - ✧ Admin replicated and injected network-related bugs.
  - ✧ Diagnosed ***90%*** of the injected bugs with a false positive rate of ***3%***.
- ***Diagnosed root causes of popular apps***
  - ✧ FTP client
    - Client behind NAT
    - High data loss
  - ✧ Pidgin
    - IP change
    - Message loss
  - ✧ Skype
    - Data loss due to delay
    - A different thread closes socket
    - Client behind NAT
  - ✧ VirtualBox (newly discovered bug)
    - Virtualization misbehavior

Efficiency

- Runtime performance overhead.
  - ✧ Between ***linear*** and ***quadratic***



Acknowledgements

National Science Foundation Awards 1223588 & 1205415, NSF Graduate Research Fellowship Award 1104522, the NYU WIRELESS research center and the Center for Advanced Technology in Telecommunications (CATT).