

# NetCheck: Network Diagnoses from Blackbox Traces

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

(<sup>\*</sup>)New York University, (<sup>^</sup>)University of British  
Columbia, (<sup>@</sup>)University of Washington

# Goal

- Find bugs in networked applications
  - Large complex unknown applications



- Large complex unknown networks



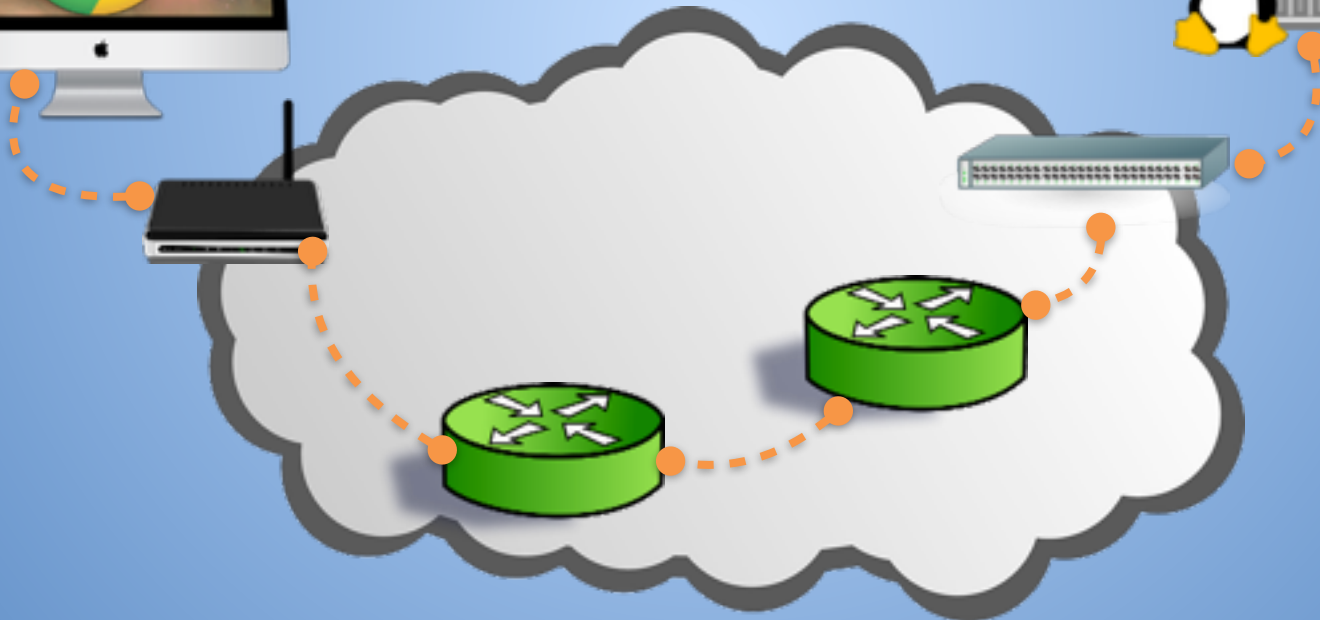
- Understandable output / fix

# Motivation

Chrome Client



Apache Server





# Motivation

Chrome Client



probing  
ping

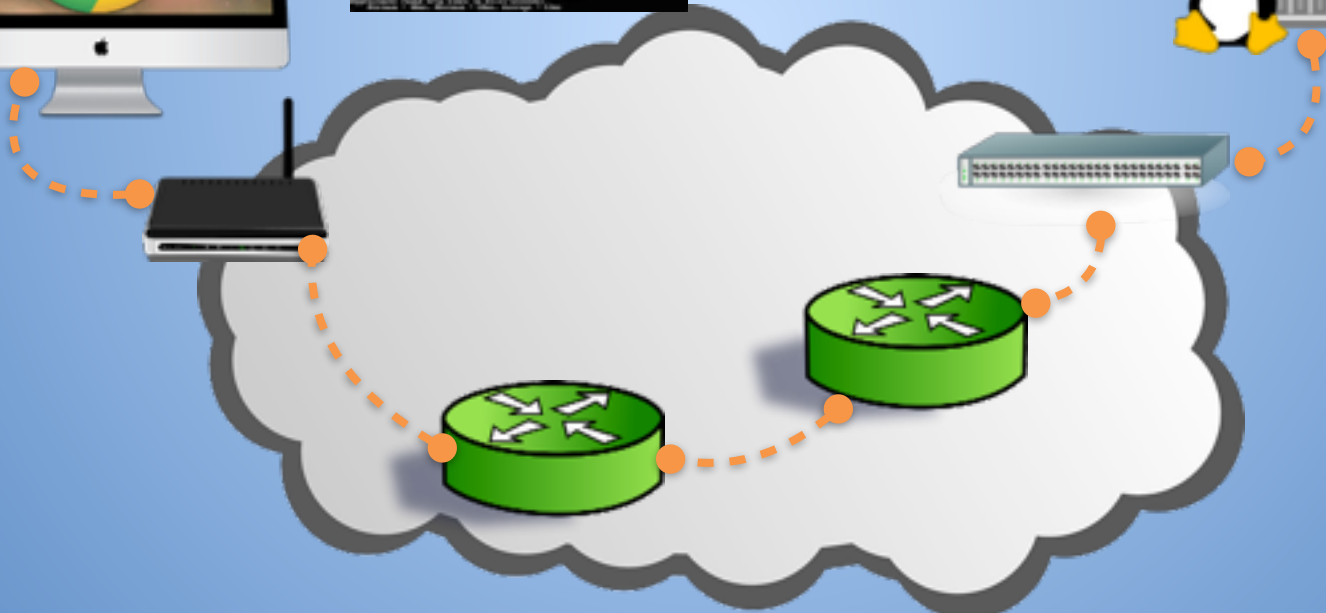
```
ping -n -c 1 google.com
Pinging google.com [66.255.67.227] with 32 bytes of data:
Request timed out.

ping -n -c 1 google.com
Pinging google.com [66.255.67.227] with 32 bytes of data:
Request timed out.

ping -n -c 1 google.com
Pinging google.com [66.255.67.227] with 32 bytes of data:
Request timed out.
```

Different traffic (ICMP)  
Often different result

Apache Server

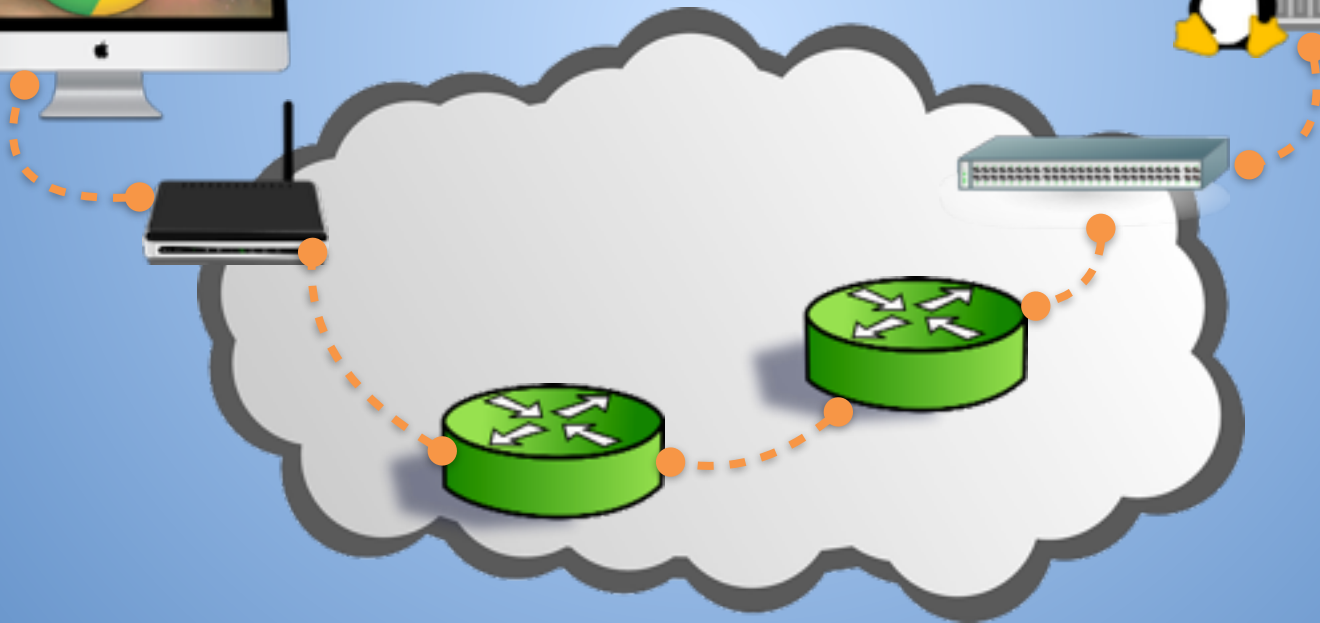


# Motivation

Chrome Client



Apache Server



# Motivation

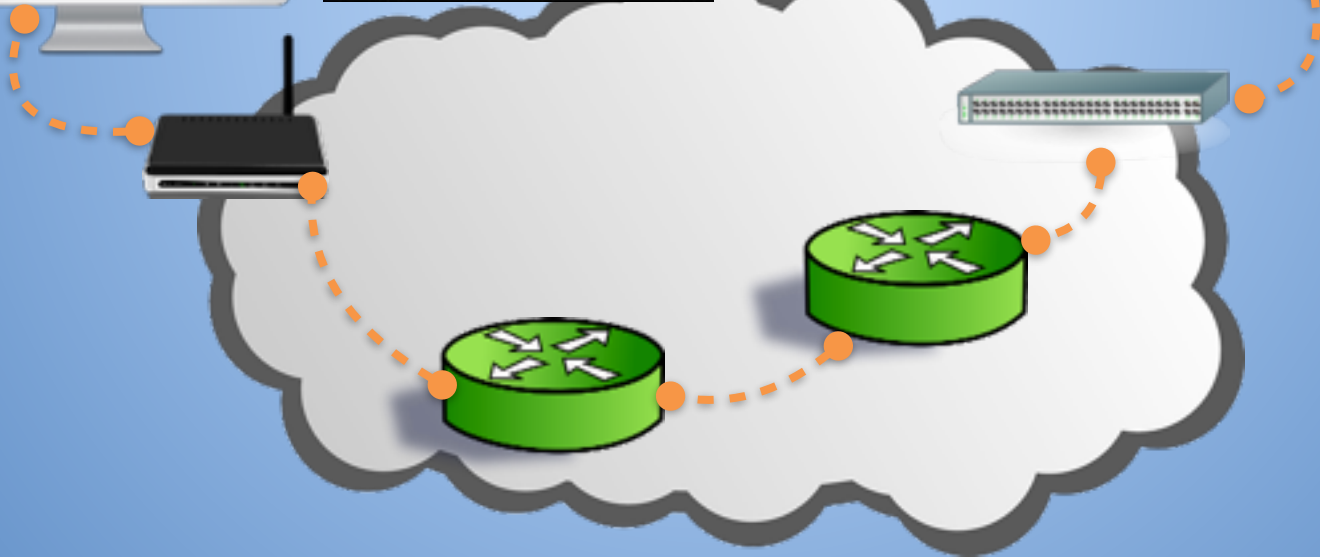
Chrome Client



packet capture



Apache Server



# Motivation

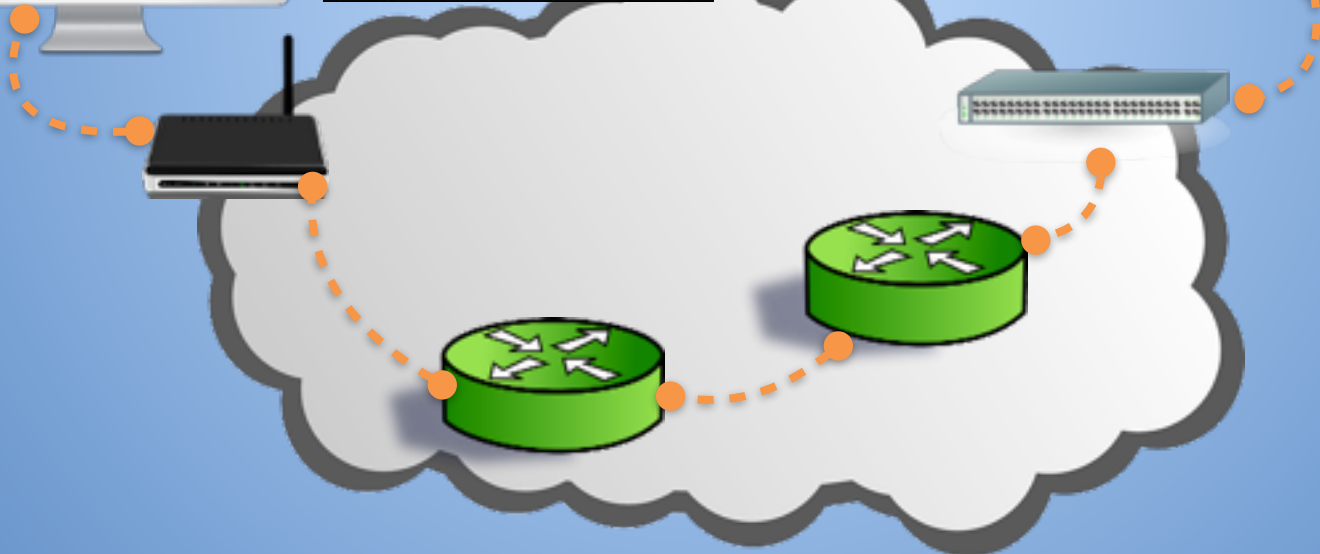
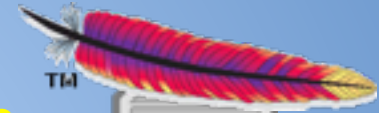
Requires detailed protocol / app knowledge

Apache Server

Chrome Client



packet capture



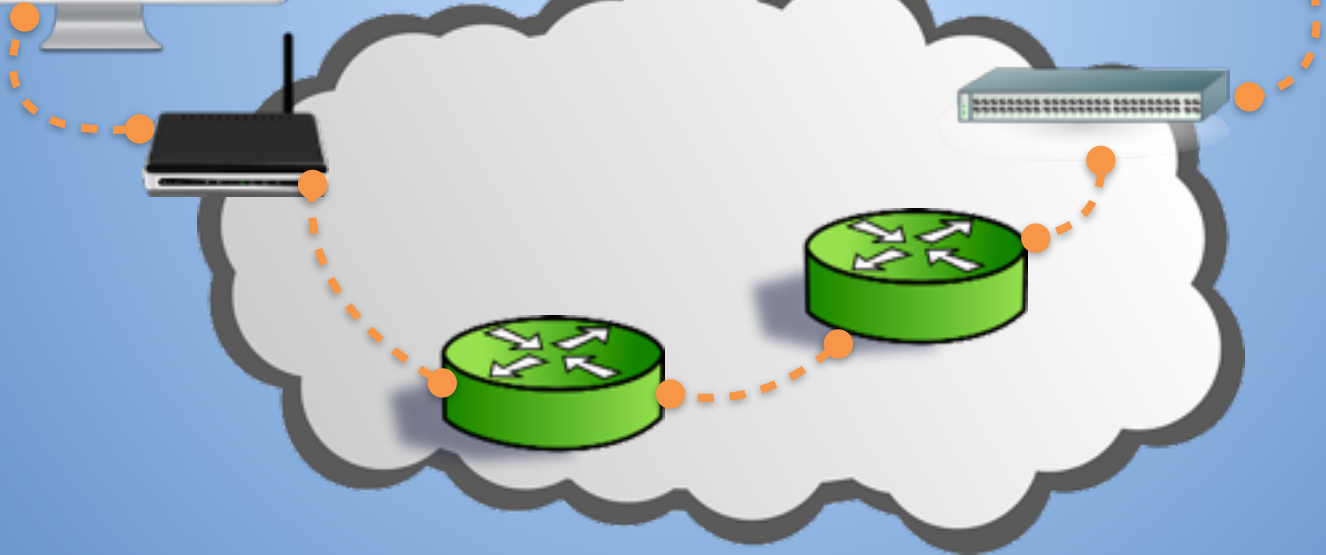


# Motivation

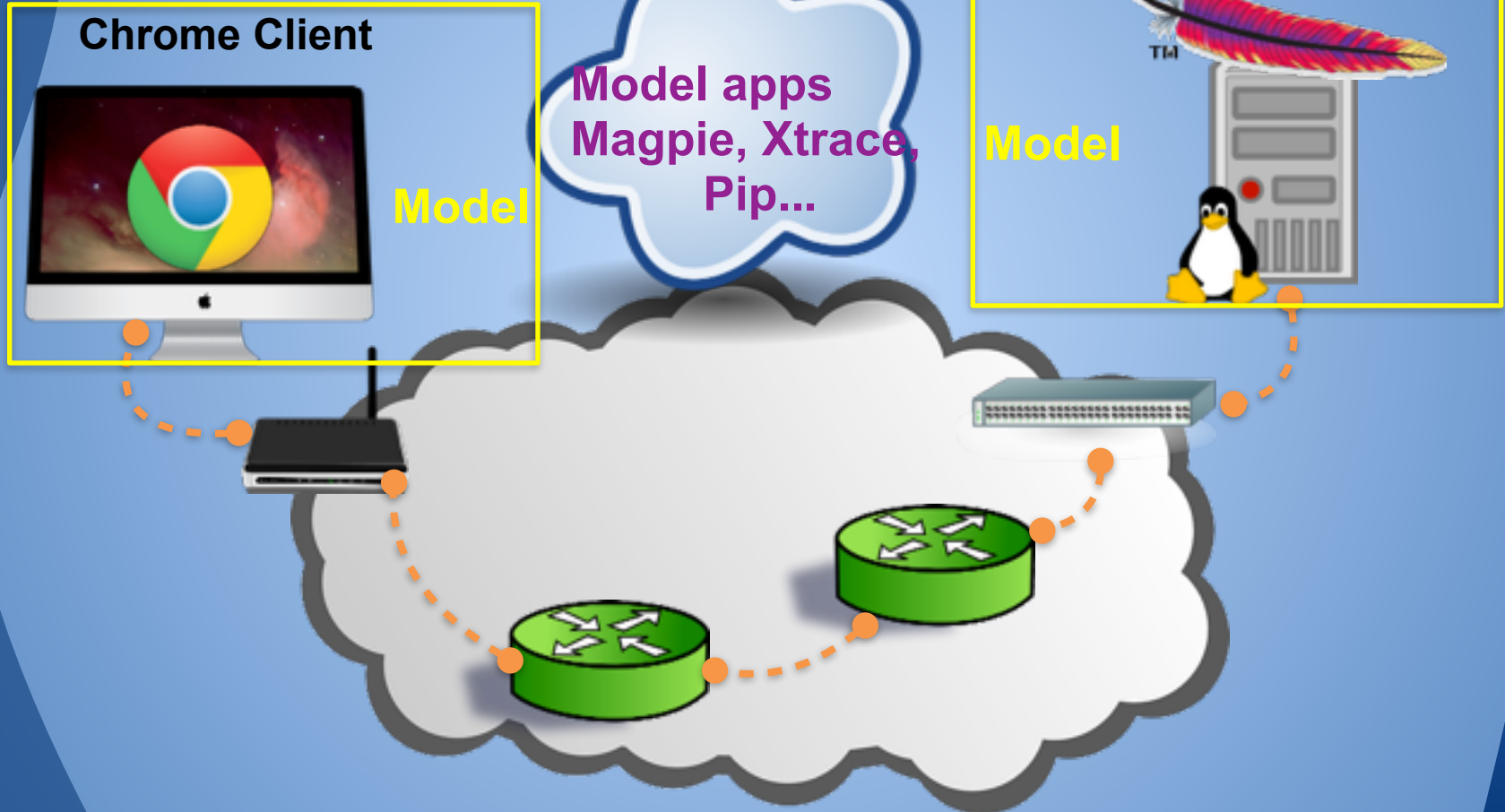
Chrome Client



Apache Server



# Motivation



# Motivation

Need a model  
per application


**Chrome Client**



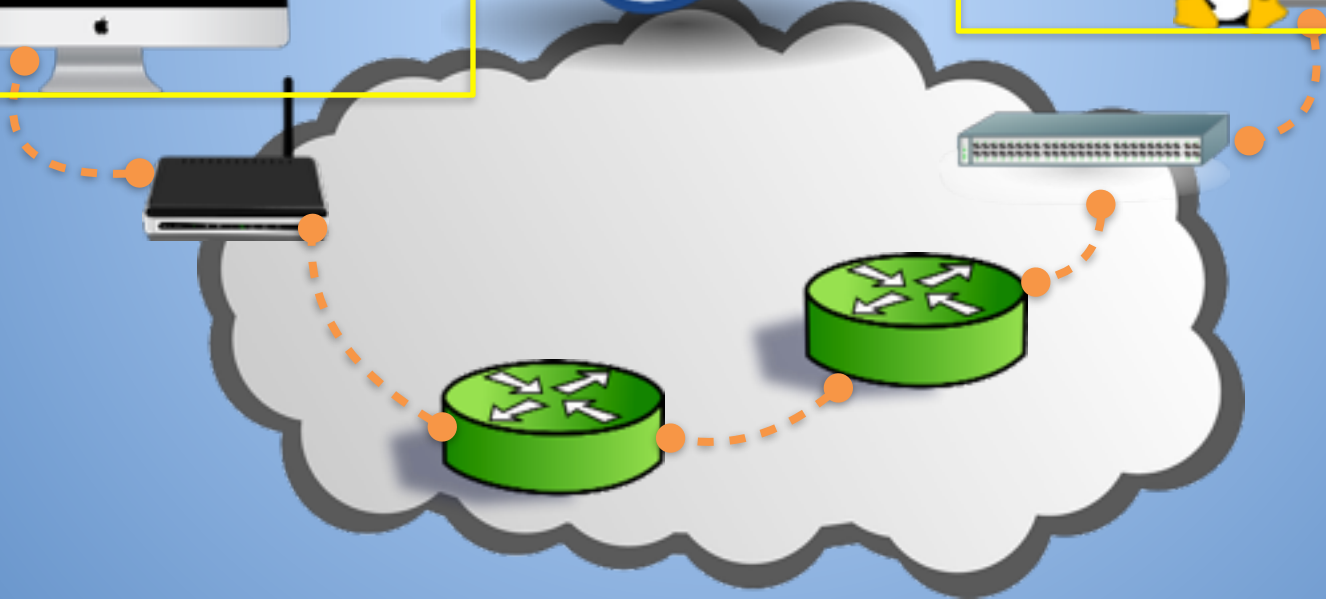
**Model**

Model apps  
Magpie, Xtrace,  
Pip...

**Apache Server**



**Model**

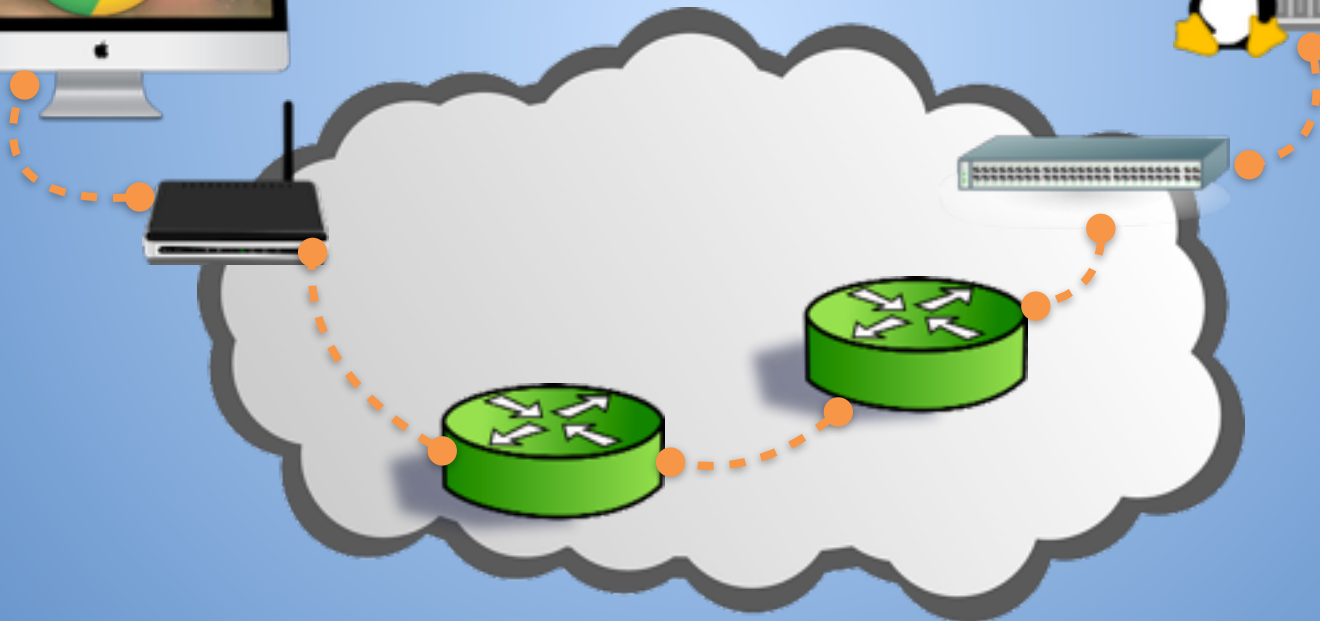


# Motivation

Chrome Client



Apache Server



# Motivation

Header  
Space  
Analysis, etc.

Apache Server

Chrome Client

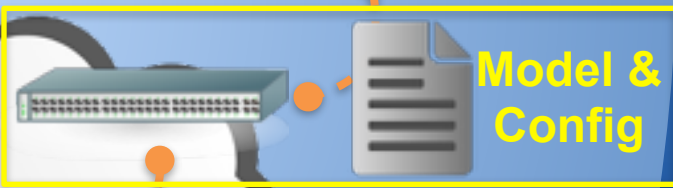


Network Config  
Analysis

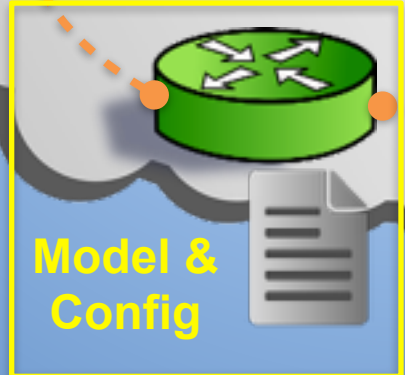
Model &  
Config



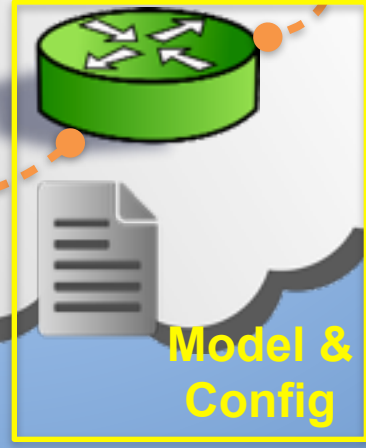
Model &  
Config



Model &  
Config



Model &  
Config



# Motivation

Need detailed network knowledge  
HW + config

Chrome Client



Apache Server



Network Config Analysis

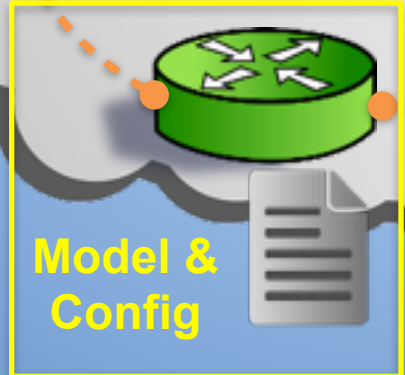
Model & Config



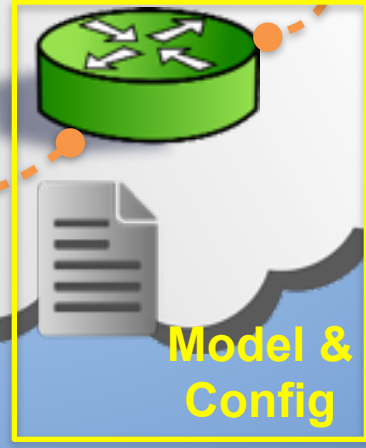
Model & Config



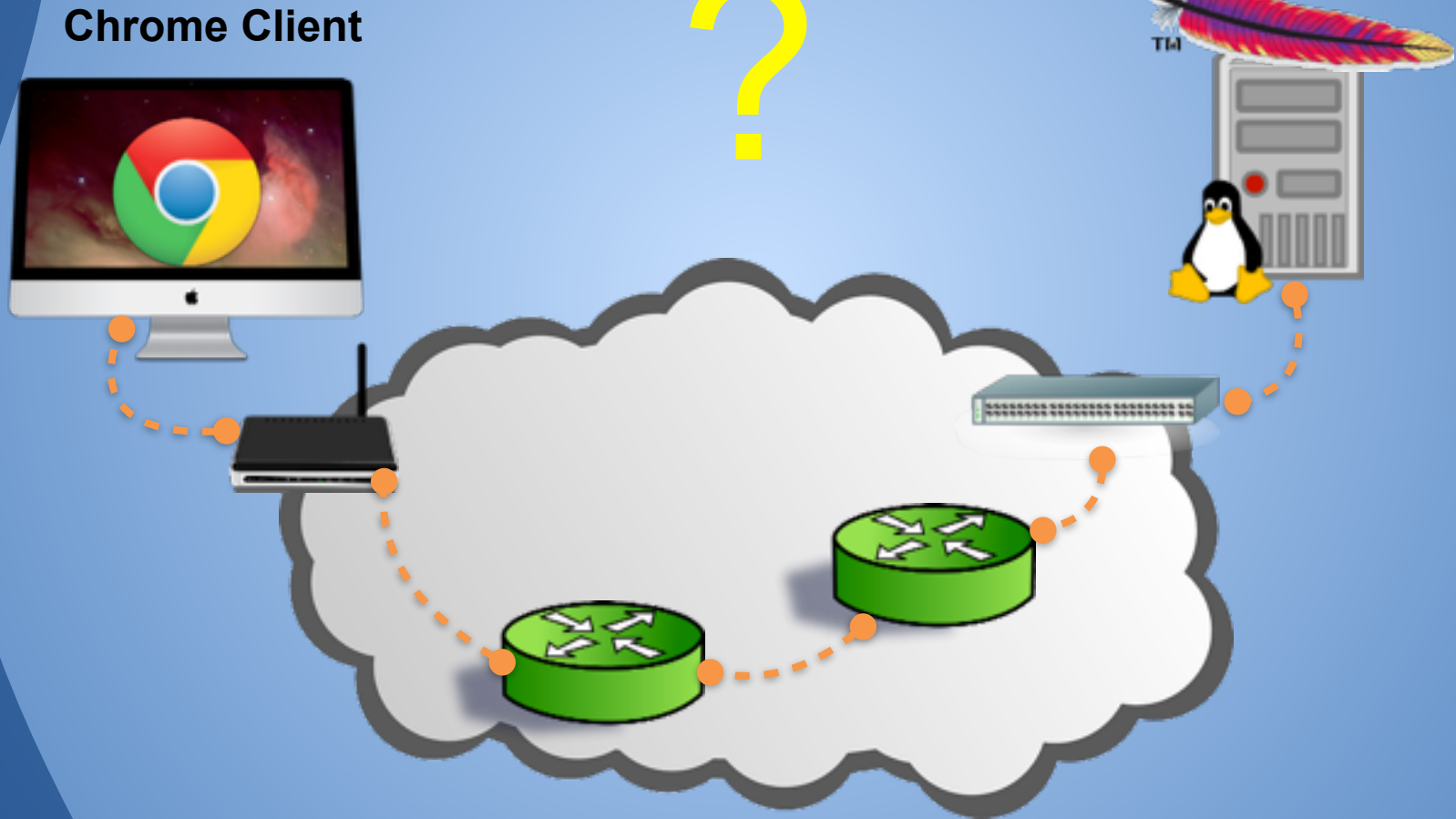
Model & Config



Model & Config



# Motivation



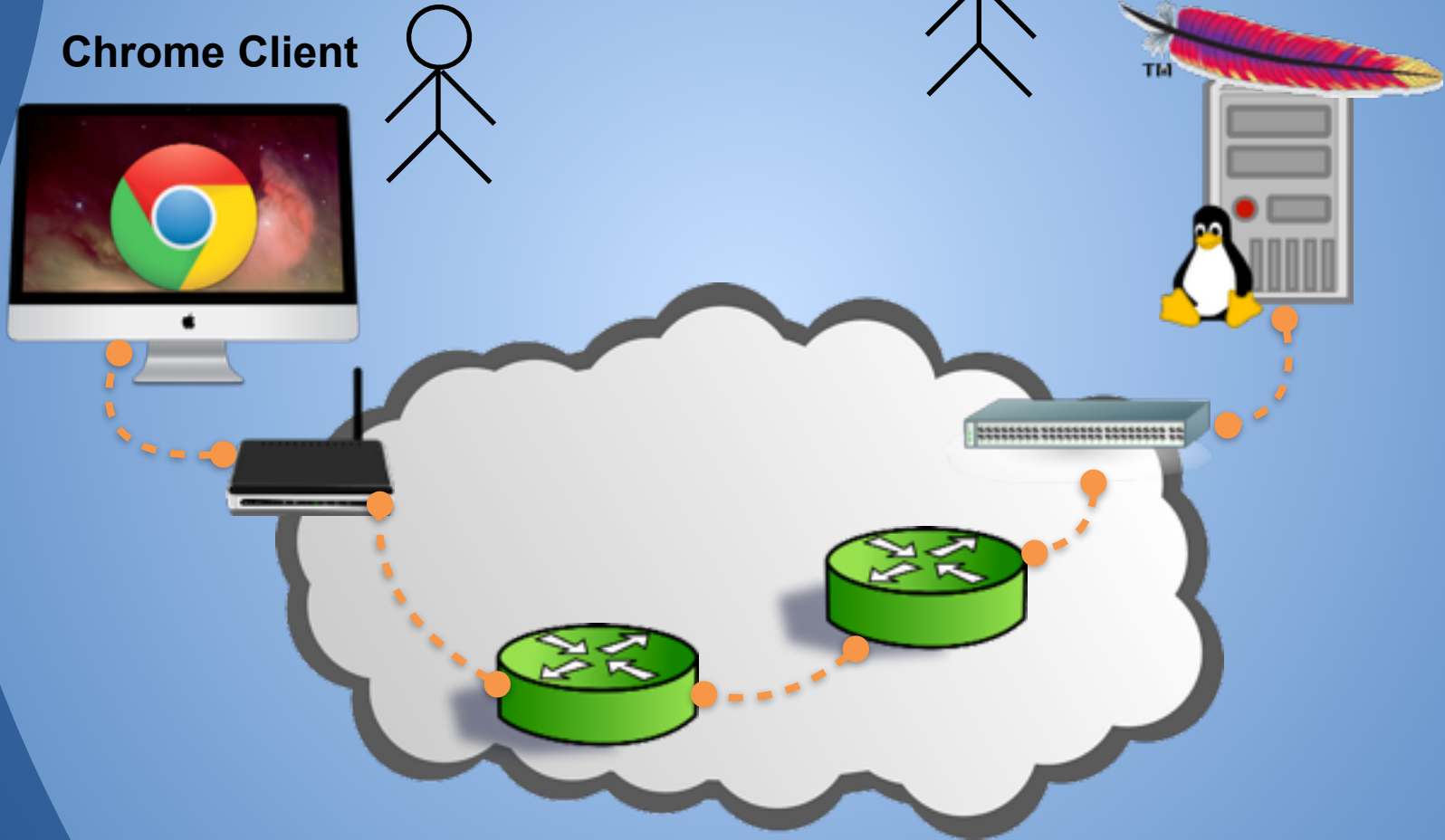
# NetCheck

programmer

programmer

Chrome Client

Apache Server





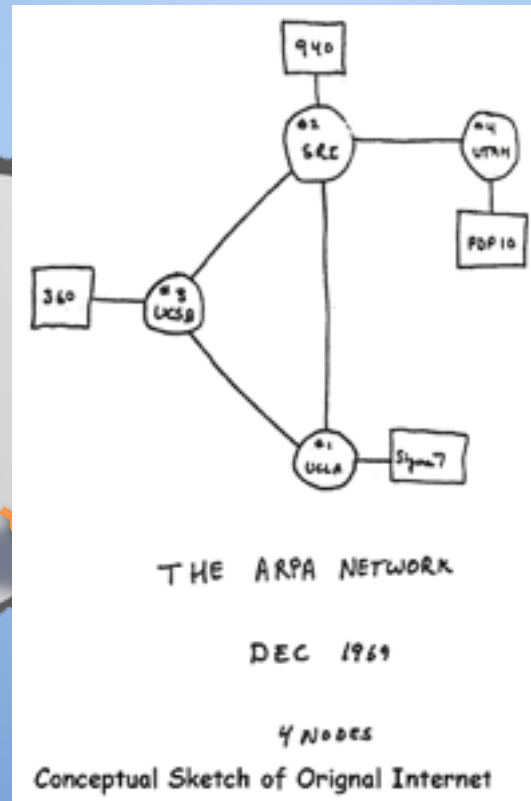
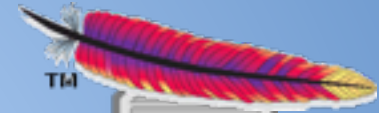
# NetCheck

programmer

programmer

Chrome Client

Apache Server



# NetCheck

programmer

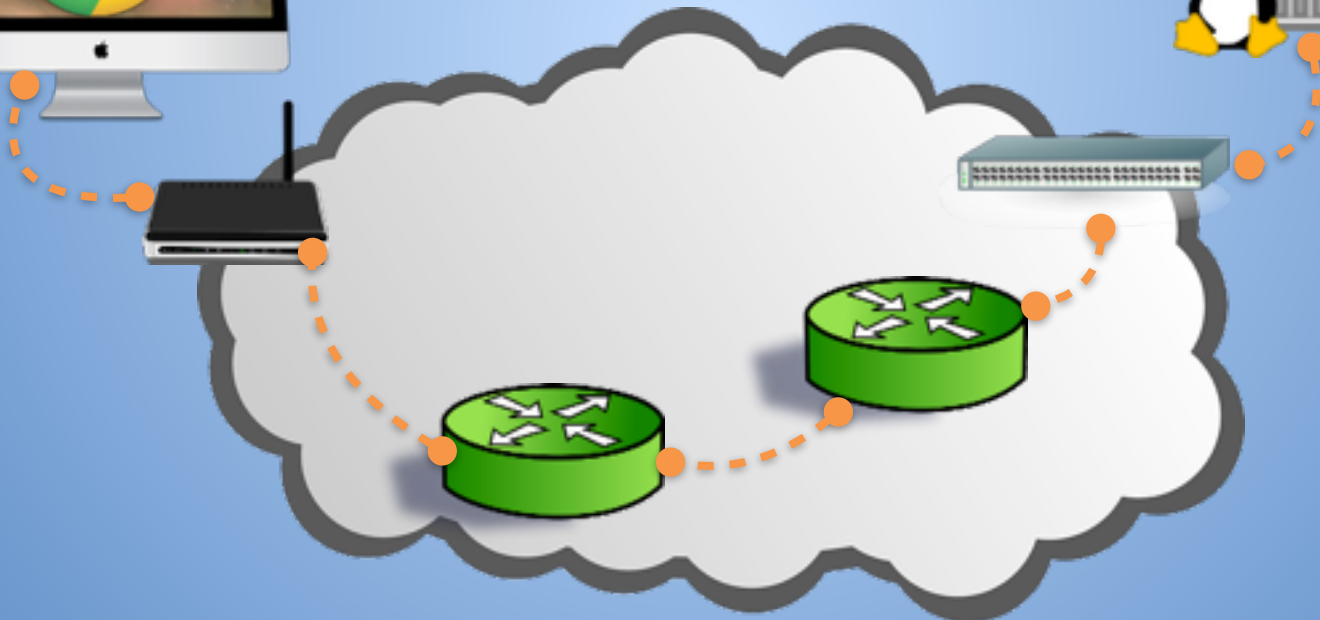
programmer

Chrome Client

Apache Server



Model Programmer's  
Understanding  
Deutsch's Fallacies



# Outline

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

# NetCheck overview

Application



Traces



NetCheck



Likely Faults



**Fail**



# NetCheck overview

Application



Traces



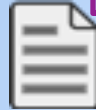
NetCheck



Likely Faults



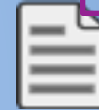
ktrace



**Fail**



strace



# NetCheck overview

Application



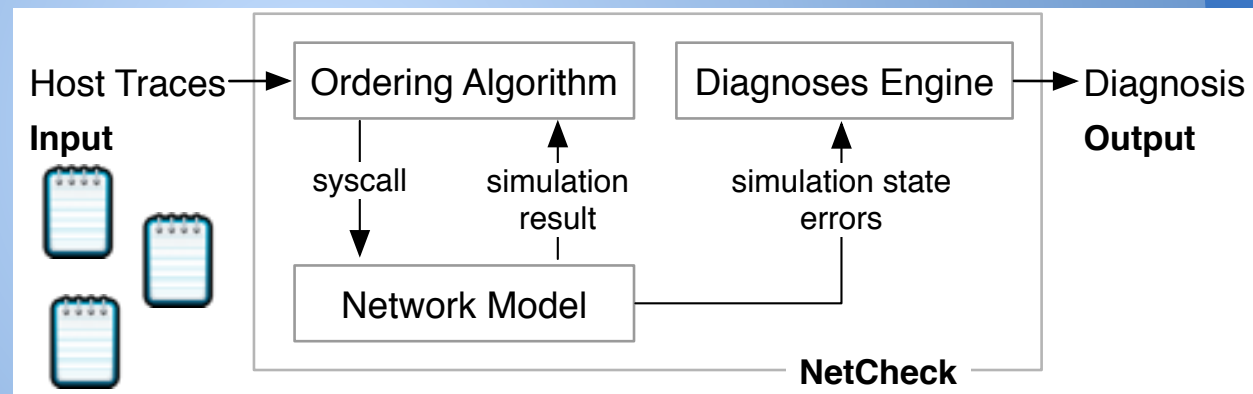
Traces



**NetCheck**



Likely Faults



# NetCheck overview

Application



Traces



NetCheck



Likely Faults

**Network Configuration Issues**

Network Configuration Issues

```
Trace B failed to connect to ::1 time(s) with an unknown error
This address matches server socket 819, which was bound to 8.8.8.8:51972
* [warning] Peer is an unspecified address '::', which is not portable
* [warning] Unable to verify port numbers, which should only happen for the client
* [warning] Addresses '::' and '8.8.8.8' are not the same IP version
This address matches server socket 10, which was bound to 8.8.8.8:44126
* [warning] Peer is an unspecified address '::', which is not portable
* [warning] Unable to verify port numbers, which should only happen for the client
* [warning] Addresses '::' and '8.8.8.8' are not the same IP version
```

**Traffic Statistics**

Traffic Statistics

```
Traffic from trace a to 239.255.255.258:1900
* 285 bytes sent, 0 bytes received, 285 bytes lost (100.00%)
Traffic from trace b to 239.255.255.258:1900
* 795 bytes sent, 0 bytes received, 795 bytes lost (100.00%)
Traffic from trace c to 239.255.255.258:1900
* 285 bytes sent, 0 bytes received, 285 bytes lost (100.00%)
```

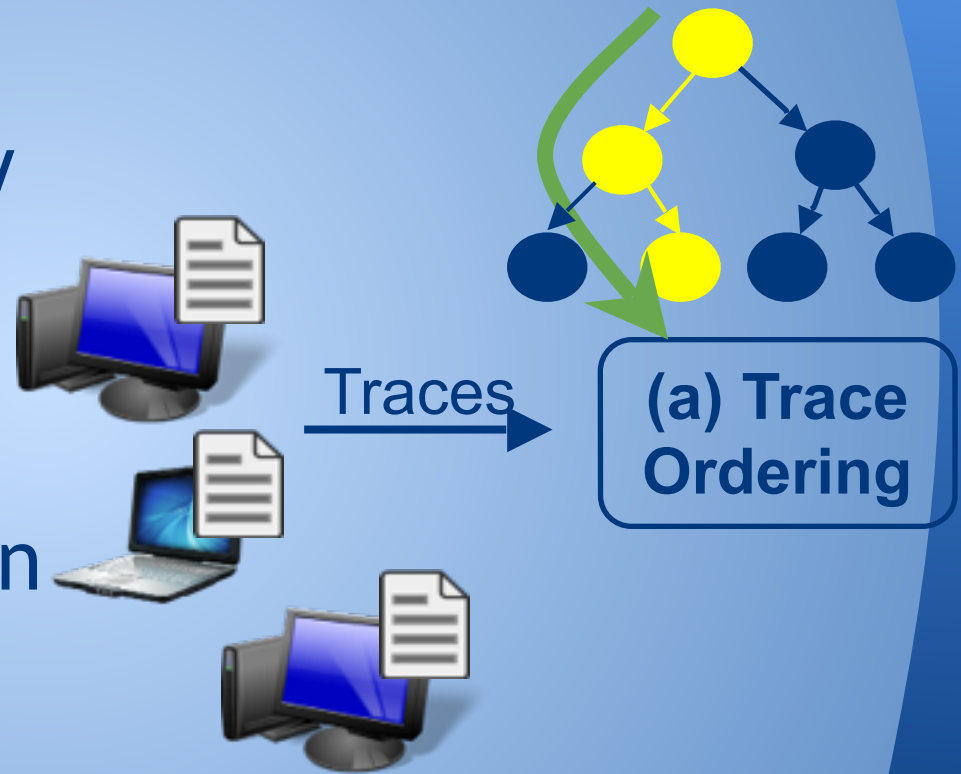
**Problem Detected**

Possible Problems Detected

```
* The tool failed to process the entire trace
* [Possible Network Misbehavior] One or more connects failed with an unknown error. This may be due to something filtering connection, for example a firewall.
* [Possible Network Misbehavior] All 3 datagrams sent from trace b to 239.255.255.258:1900 were lost
* [Possible Network Misbehavior] All 6 datagrams sent from trace B to 239.255.255.258:1900 were lost
* [Possible Network Misbehavior] All 2 datagrams sent from trace C to 239.255.255.258:1900 were lost
* 11 call(s) to getsockopt, setsockopt, fcntl, or ioctl used options which are not currently handled
```

# 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”

```
socket() = 3
bind(3, ...) Call arguments = 0
listen(3, 1) = 0
accept(3, ...) = 4
recv(4, "HTTP", ...) = 4 Return values
close(4) Return buffer = 0
```

# 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

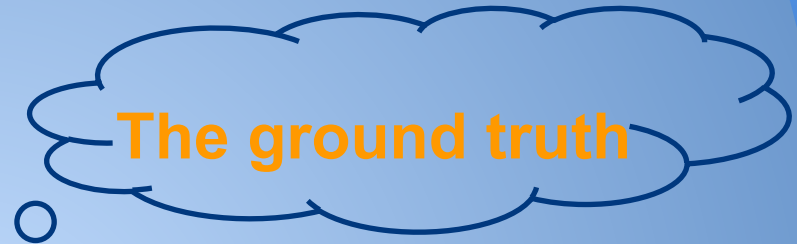
- one trace per host

- local order but no global order

Q: how do we reconstruct what really happened?

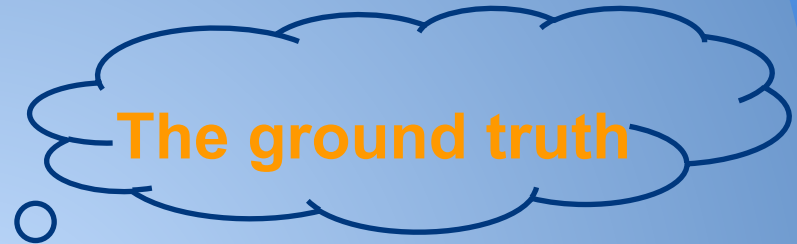
# What we want is this

A1. socket()	= 3
B1. socket()	= 3
A2. bind(3, .. .)	o = 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



# 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



**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

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

One valid ordering: all syscalls returned successfully.

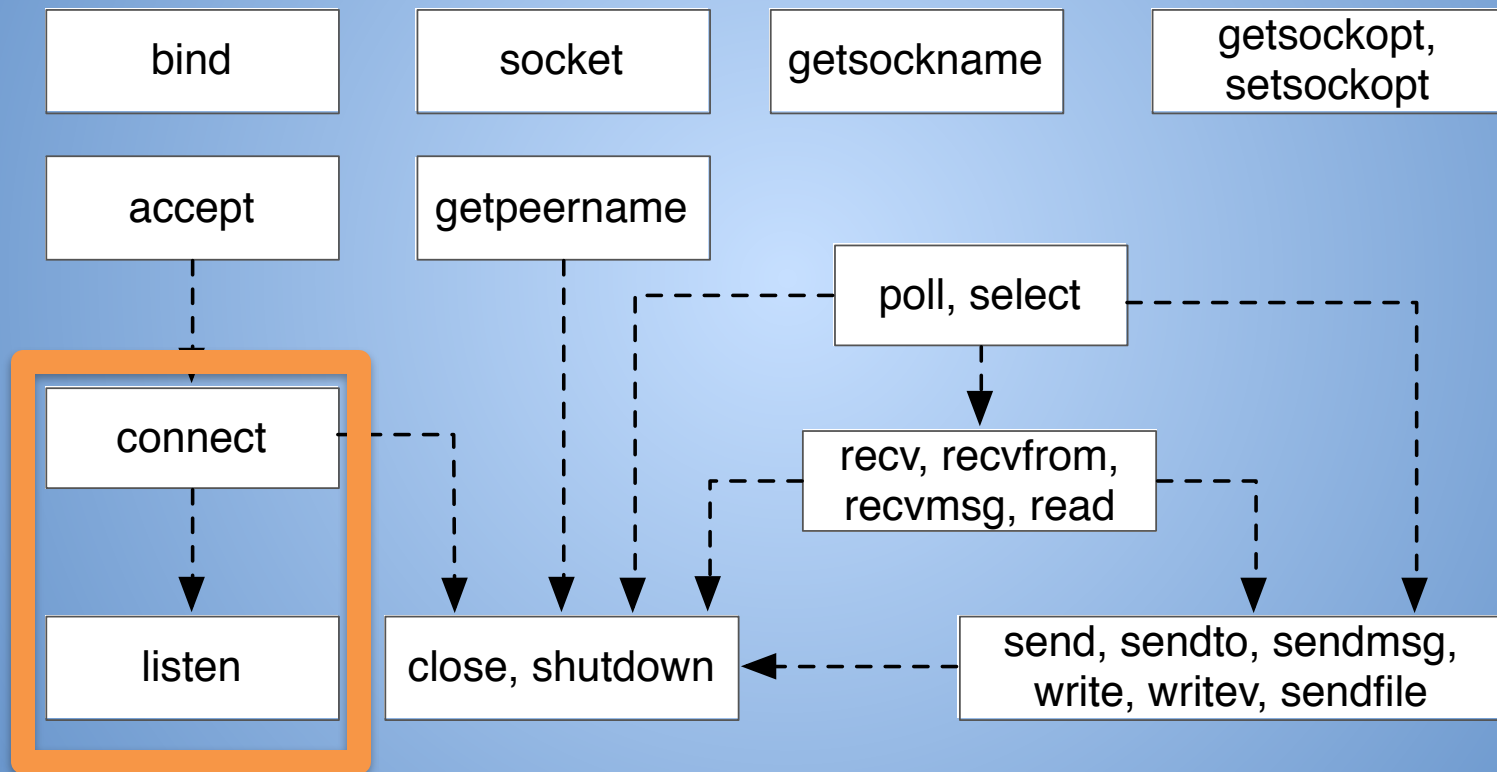
```
A2. bind(3, ...)    = 0
B2. connect(3, ...) = -1, ECONNREFUSED
A3. listen(3, 1)   = 0
```

A second valid ordering: connect failed with ECONNREFUSED.

A call's return value **may-depend-on** a remote call's action

Result indicates order of calls

# Deciding call order

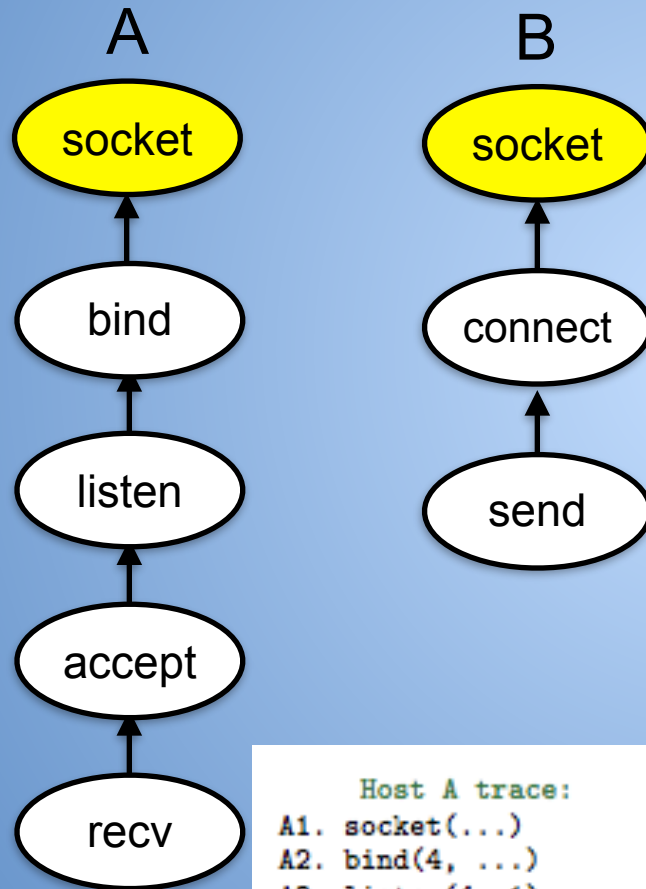


full set of **may-depend-on** relations



# Ordering Algorithm

## Input traces



## Algorithm process

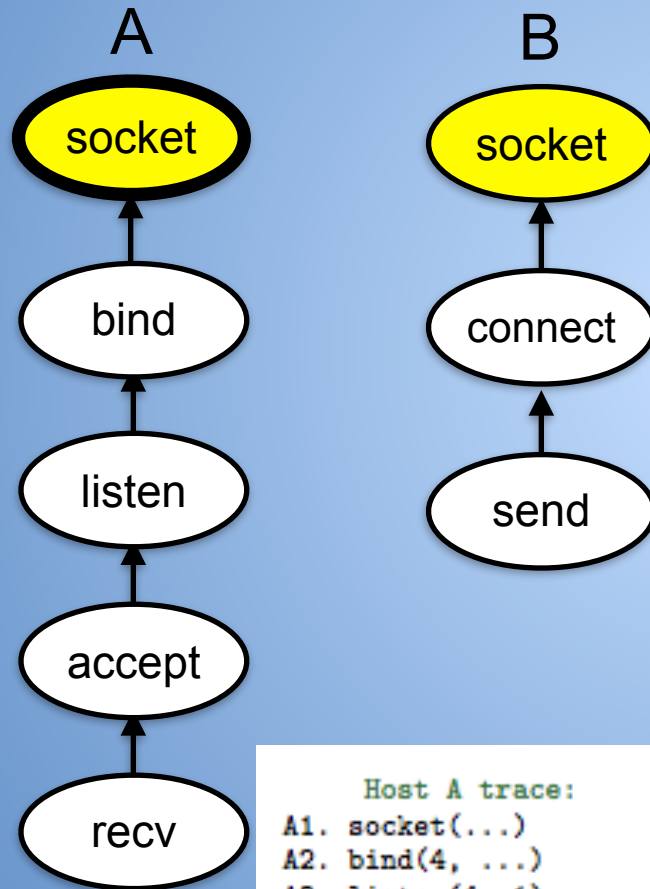
## Output Ordering

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

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

# Ordering Algorithm

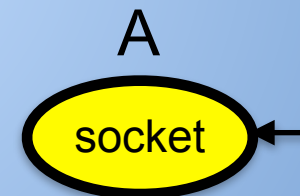
## Input traces



## Algorithm process

Try socket on host A: **accepted**

## Output Ordering

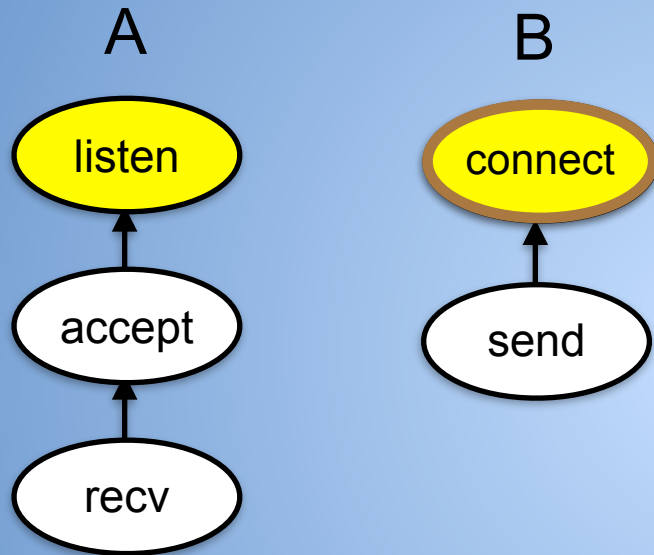


```
Host A trace:
A1. socket(...)      = 4
A2. bind(4, ...)    = 0
A3. listen(4, 1)    = 0
A4. accept(4, ...)  = 6
A5. recv(6, "Hola!", ...) = 5
```

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

# Ordering Algorithm

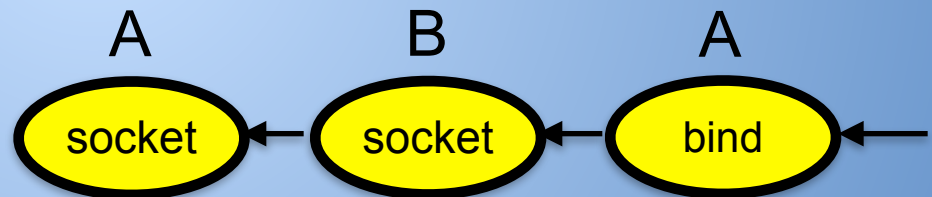
## Input traces



## Algorithm process

Try connect on host B: **rejected**

## Output Ordering

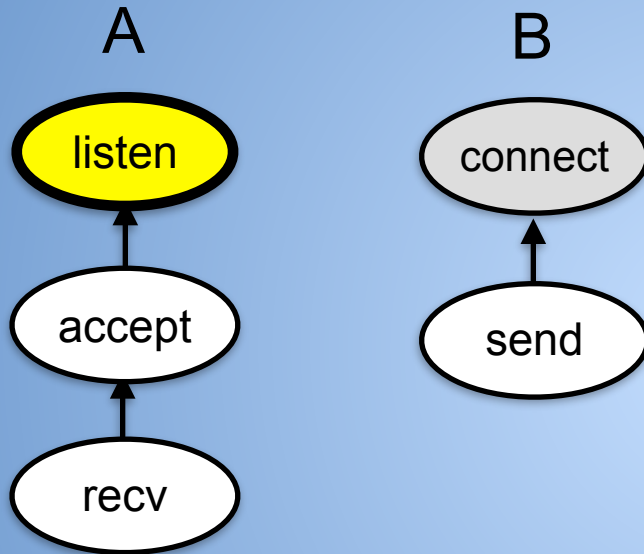


```
Host A trace:  
A1. socket(...) = 4  
A2. bind(4, ...) = 0  
A3. listen(4, 1) = 0  
A4. accept(4, ...) = 6  
A5. recv(6, "Hola!", ...) = 5
```

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

# Ordering Algorithm

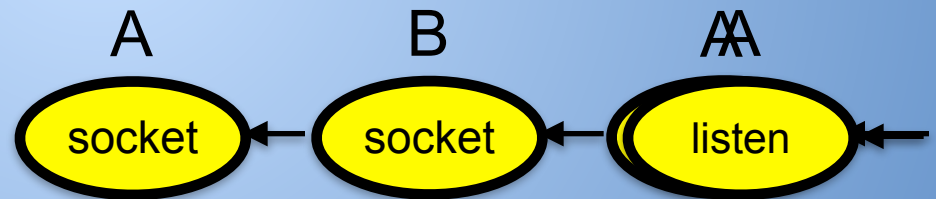
## Input traces



## Algorithm process

Try listen on host A: **accepted**

## Output Ordering



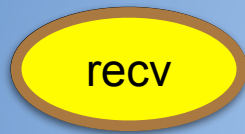
```
Host A trace:
A1. socket(...) = 4
A2. bind(4, ...) = 0
A3. listen(4, 1) = 0
A4. accept(4, ...) = 6
A5. recv(6, "Hola!", ...) = 5
```

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

# Ordering Algorithm

## Input traces

A



B

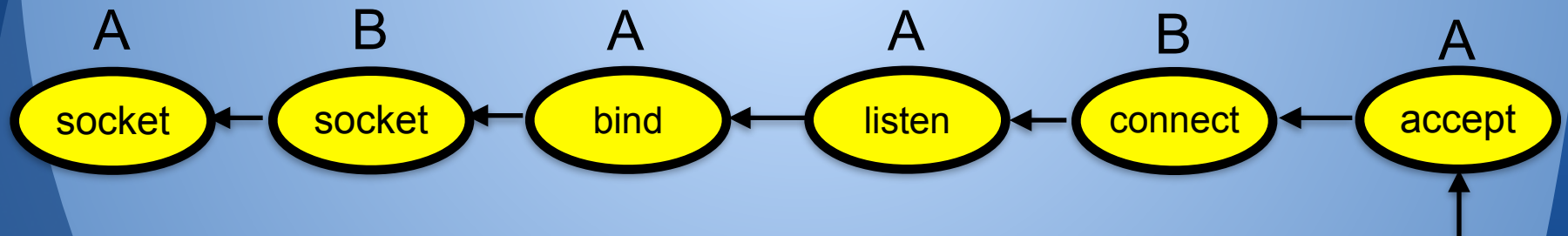


## Algorithm process

Try recv on host A: **rejected**

TCP BUFFER: ""

## 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

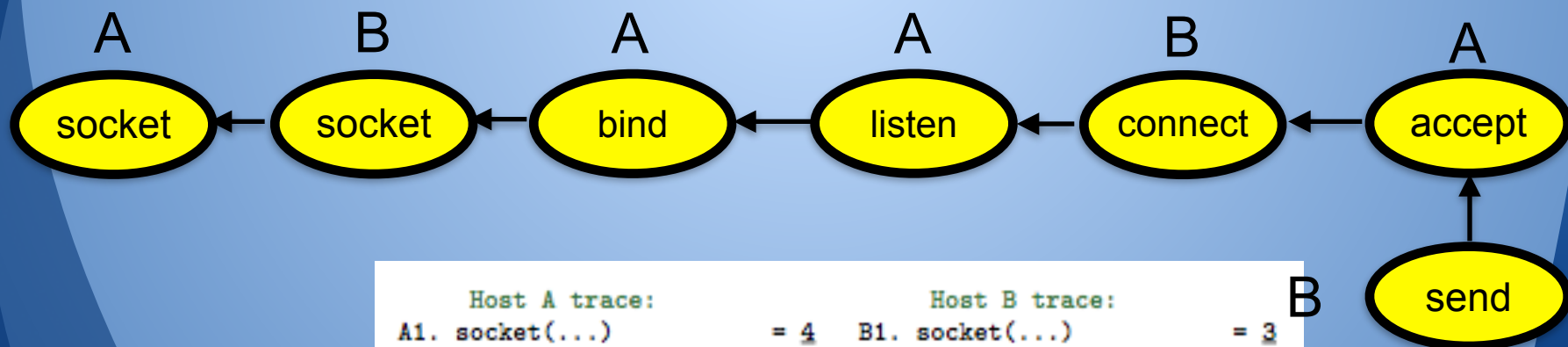


## Algorithm process

Try send on host B: **accepted**

TCP BUFFER: ""

## 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



B

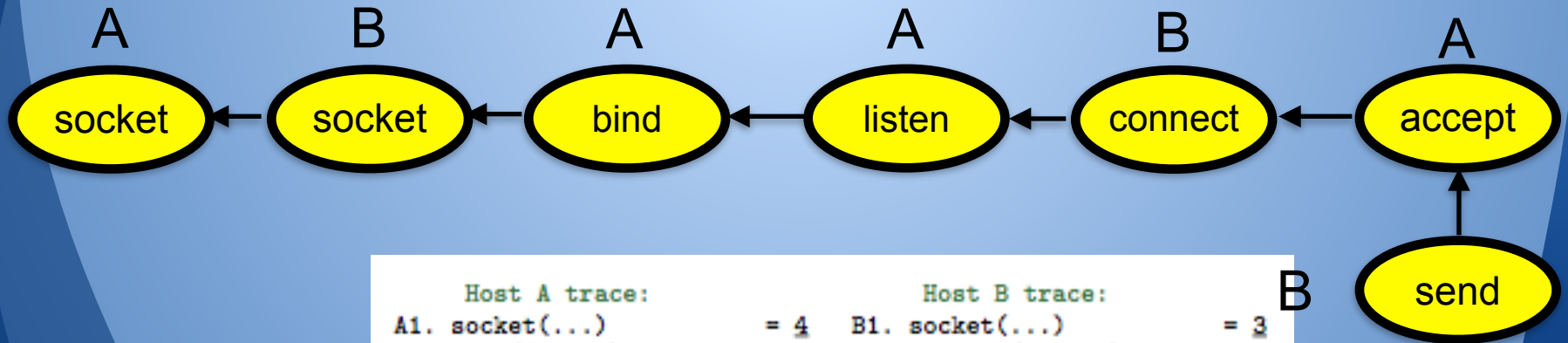
None

## Algorithm process

Try send on host B: **accepted**

TCP BUFFER: "Hello"

## 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
```

B

# Ordering Algorithm

## Input traces

A

recv

B

None

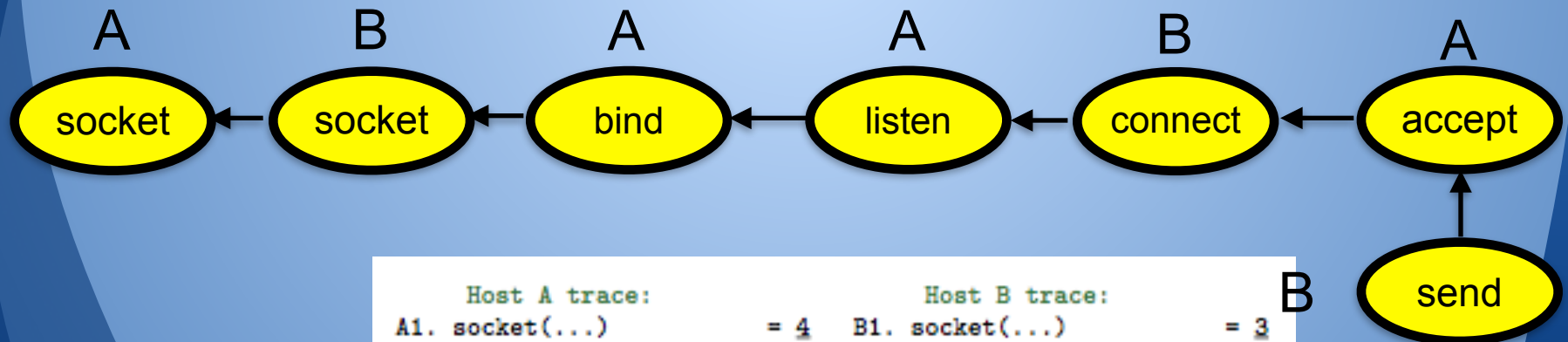
TCP BUFFER: "Hello"

## Algorithm process

Try recv on host A:

**Fatal Error**

## 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:

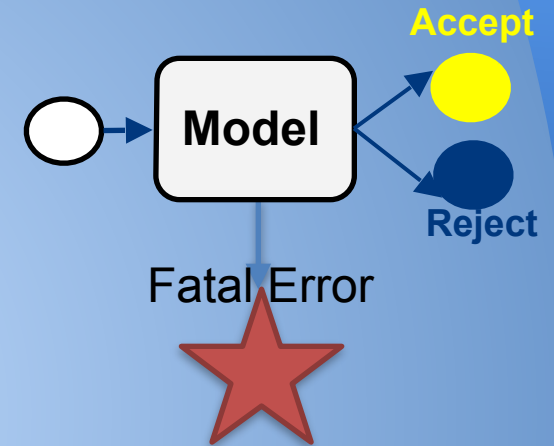
```
B1. socket(...) = 3  
B2. connect(3, ...) = 0  
B3. send(3, "Hello", ...) = 5
```

B



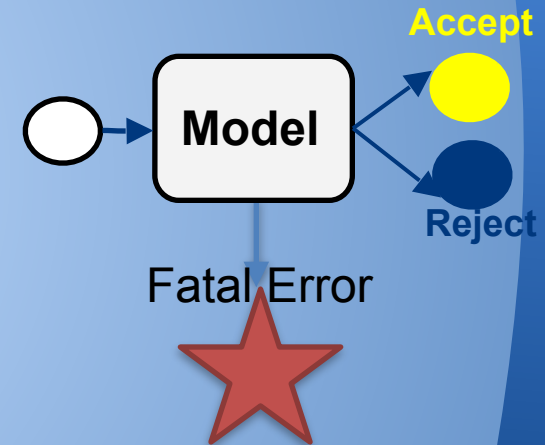
# Outline

- 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
  - *Accept* → can process (correct buffer)
  - *Reject* → wrong order (incomplete buffer)
  - *Permanent reject* → abnormal behavior (incorrect buffer)



# Network Model

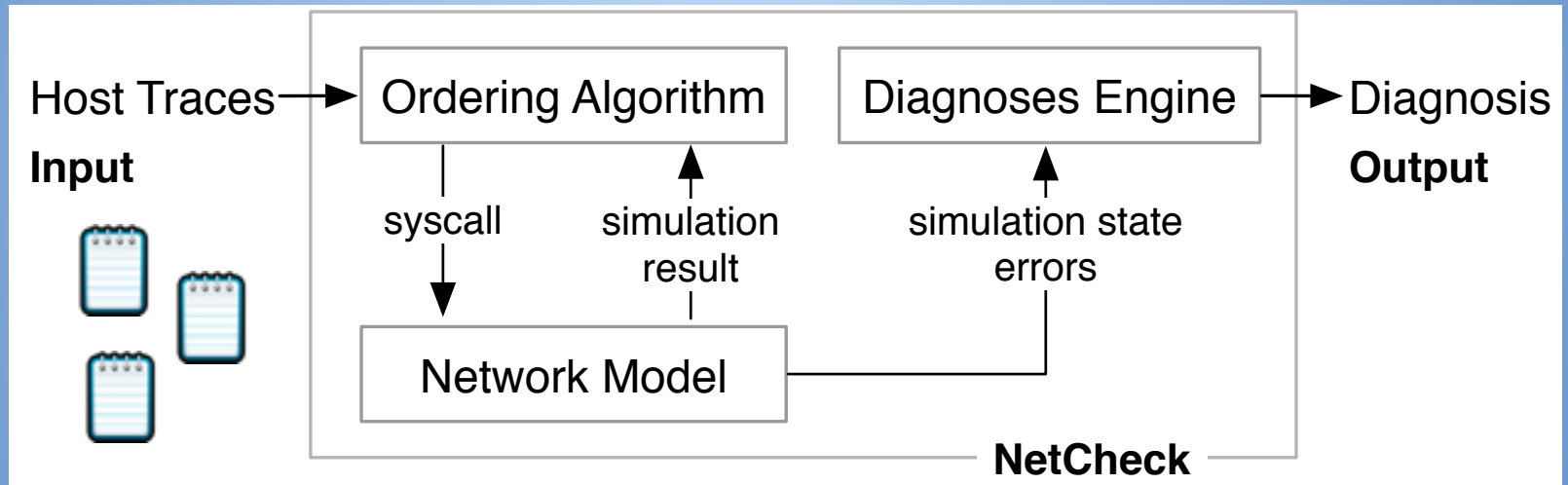
- Simulates invocation of a syscall
- Capture programmer assumptions
  - Assumes a simplified network view
    - Assume transitive connectivity
    - Little, random loss
    - No middle boxes
  - Assume uniform platform
    - Flag OS differences



THE ARPA NETWORK

# 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

## (c) Fault Classifier

```
Network Configuration Issues
-----
Trace A failed to connect to 10.1.1.1 (times) with an unknown error
This address receives no requests, which was found to R & B 0 (100%)
Learning failed to verify your network, which should only happen for the client
Learning addresses: ... and '2.0.0.0' are with the same IP network
This address matches server socket IP, which was found to R & B 0 (0%)
Learning failed to verify your network, which is not verifiable
Learning failed to verify your network, which should only happen for the client
Learning addresses: ... and '0.0.0.0' are not the same IP network

Trace A failed to connect to 10.1.1.1 (times) because the connection was refused
Several duplicating contracts may have failed to connect
+ 1 duplicating connects from Trace B to 10.0.0.1:10000 were never observed to connect
+ 1 duplicating connects from Trace C to 10.0.0.1:10000 were never observed to connect
-----
ADP Traffic Statistics
-----
Traffic from Trace B to 10.1.1.1:10100
+ 100 bytes sent, 0 bytes received, 26 Bytes lost (100.0%)
Traffic from Trace B to 10.1.1.1:10200
+ 100 bytes sent, 0 bytes received, 26 Bytes lost (100.0%)
Traffic from Trace C to 10.1.1.1:10100
+ 200 bytes sent, 0 bytes received, 54 Bytes lost (100.0%)
Traffic from Trace C to 10.1.1.1:10200
+ 200 bytes sent, 0 bytes received, 54 Bytes lost (100.0%)
-----
Possible problems detected
-----
+ the model failed to process the entire trace
+ Possible Network Misbehavior: one or more connects failed with an unknown error. This may be due to something filtering connections, for example a firewall.
+ Possible Network Misbehavior: all 1 datagram sent from trace B to 10.1.1.1:10100 were lost
+ Possible Network Misbehavior: all 1 datagram sent from trace C to 10.1.1.1:10100 were lost
+ Possible Network Misbehavior: all 1 datagram sent from Trace C to 10.1.1.1:10200 were lost
+ 1 call(s) to getsockopt, setsockopt, bind, or listen option which are not currently handled
```

Output

# 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

### Network Configuration Issues

```
.....
Network Configuration Issues
.....
Trace B failed to connect to :: 1 time(s) with an unknown error
This address matches server socket B19, which was bound to 8.8.8.8:52072
* [warning] Peer is an unspecified address '::', which is not portable
* [warning] Unable to verify port numbers, which should only happen for the client
* [warning] Addresses '::' and '8.8.8.8' are not the same IP version
This address matches server socket B0, which was bound to 8.8.8.8:44126
* [warning] Peer is an unspecified address '::', which is not portable
* [warning] Unable to verify port numbers, which should only happen for the client
* [warning] Addresses '::' and '8.8.8.8' are not the same IP version

Trace A failed to connect to 127.0.0.1:55988 1 time(s) because the connection was refused

Several nonblocking connects may have failed to connect
* 1 nonblocking connects from trace D to 127.0.1.1:7777 were never observed to connect
* 1 nonblocking connects from trace C to 127.0.1.1:7777 were never observed to connect
.....
```

## Traffic Statistics

### Traffic Statistics

```
.....
UDP Traffic Statistics
.....
Traffic from trace D to 239.255.255.258:1988
* 285 bytes sent, 0 bytes received, 285 bytes lost (100.00%)
Traffic from trace B to 239.255.255.258:1988
* 795 bytes sent, 0 bytes received, 795 bytes lost (100.00%)
Traffic from trace C to 239.255.255.258:1988
.....
```

## Problem Detected

### Problem Detected

```
.....
Possible Problems Detected
.....
* The hook failed to process the entire trace
* [Possible Network Misbehavior] One or more connects failed with an unknown error. This may be due to something filtering connections, for example a firewall.
* [Possible Network Misbehavior] All 2 datagrams sent from trace D to 239.255.255.258:1988 were lost
* [Possible Network Misbehavior] All 8 datagrams sent from trace B to 239.255.255.258:1988 were lost
* [Possible Network Misbehavior] All 2 datagrams sent from trace C to 239.255.255.258:1988 were lost
* If call(s) to getssockopt, setsockopt,fcntl, or ioctl used options which are not currently handled
.....
```

# 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

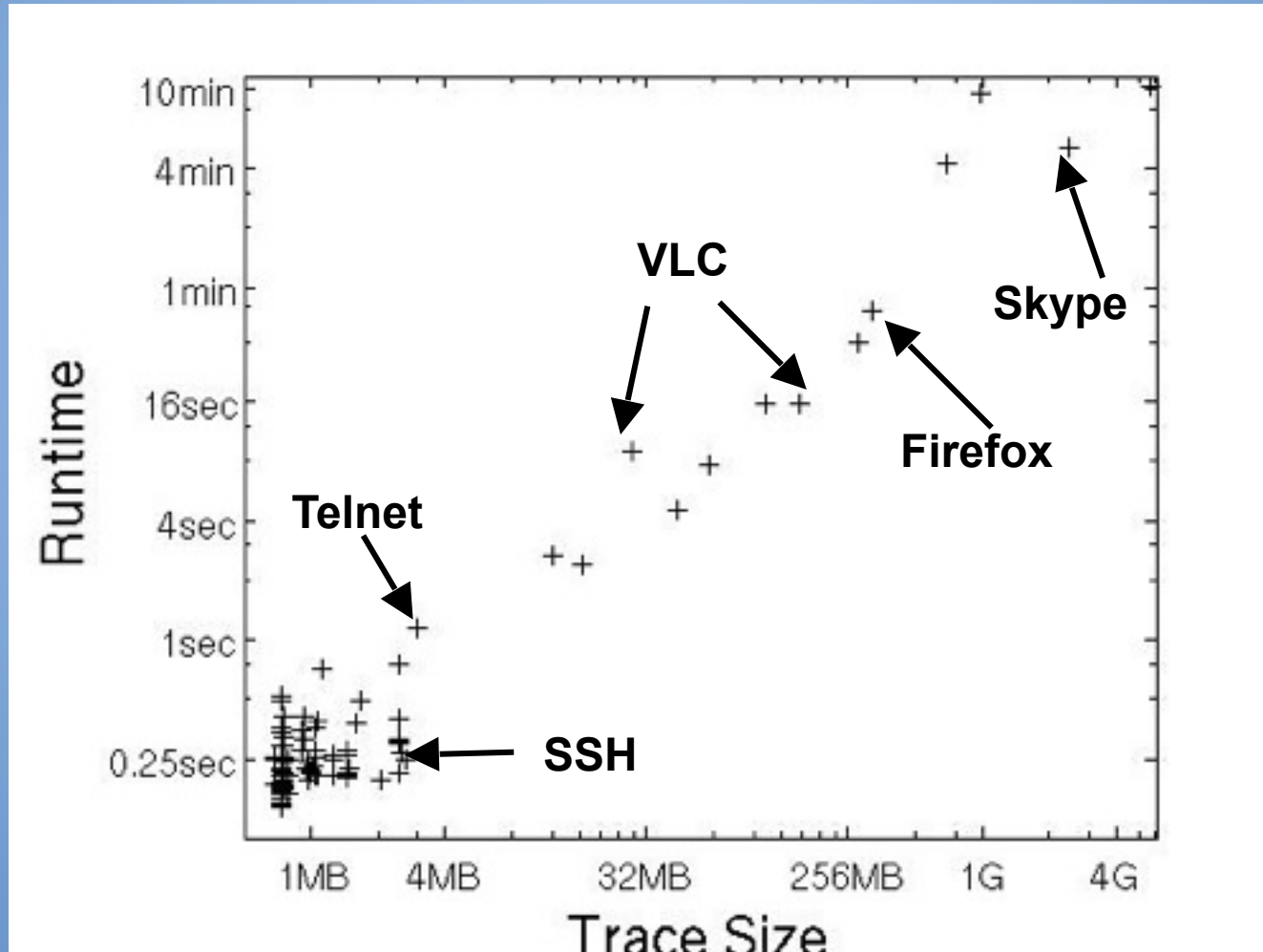
- Twenty faults observed in practice on a live network
  - MTU bug
    - Intermediary device
  - Port forward
    - Traffic sent to non-relevant addresses
  - Provide supplemental info
    - packet loss
    - buffers being closed with data in
  - 90% of cases correctly detected

# 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](http://seattle.poly.edu)



# NetCheck Performance Overhead



# Conclusion

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

- Key insights:
  - model the programmer's misconceptions
  - relation between calls → 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/>

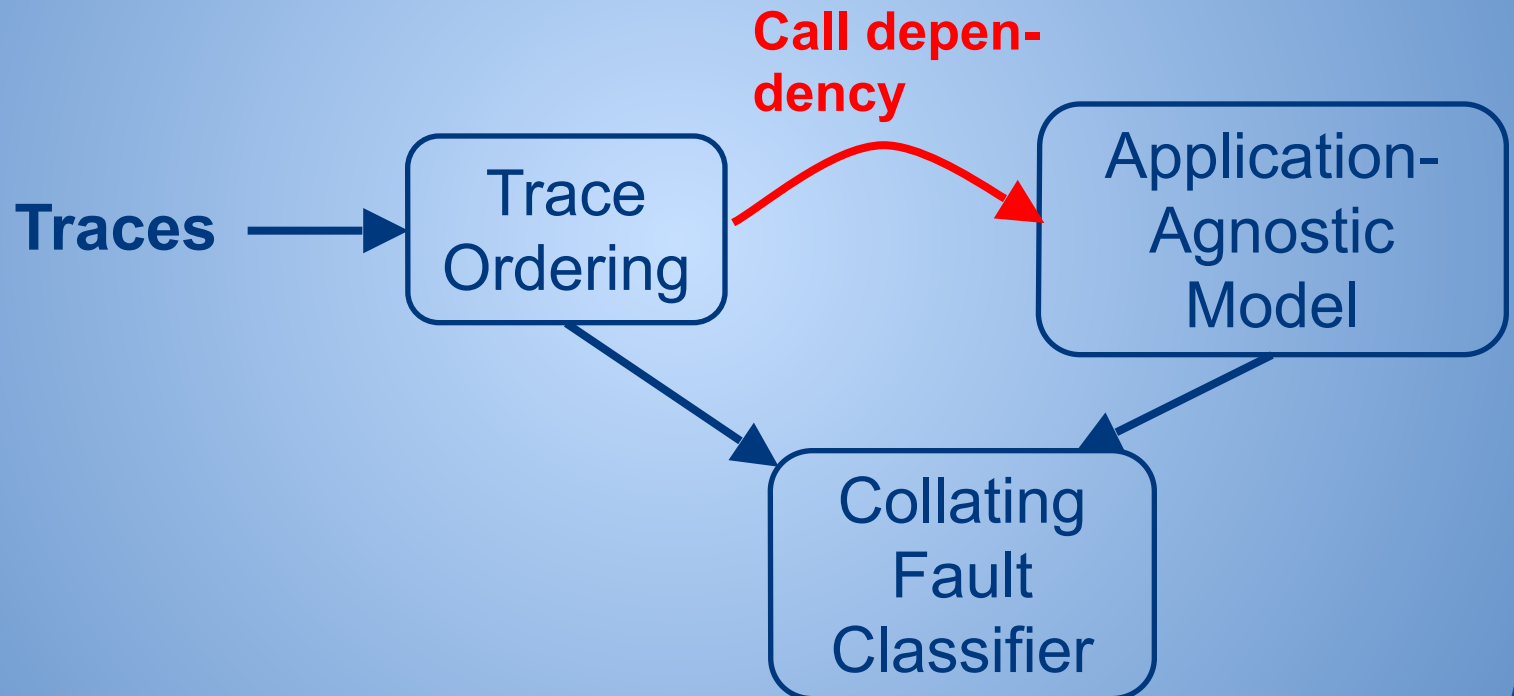
**Backup slides.**

# What is NetCheck?

- No app- or network-specific knowledge
- No modification to apps/infrastructure
- No synchronized global clock
- **Blackbox Tracing mechanism (eg, strace)**
  - **Reconstruct** a plausible total ordering of syscall traces from multiple hosts
  - Uses simulation and captured state to **identify** network related issues
  - **Map** low-level issues to higher-level characterizations of failure

# Diagnosis Model

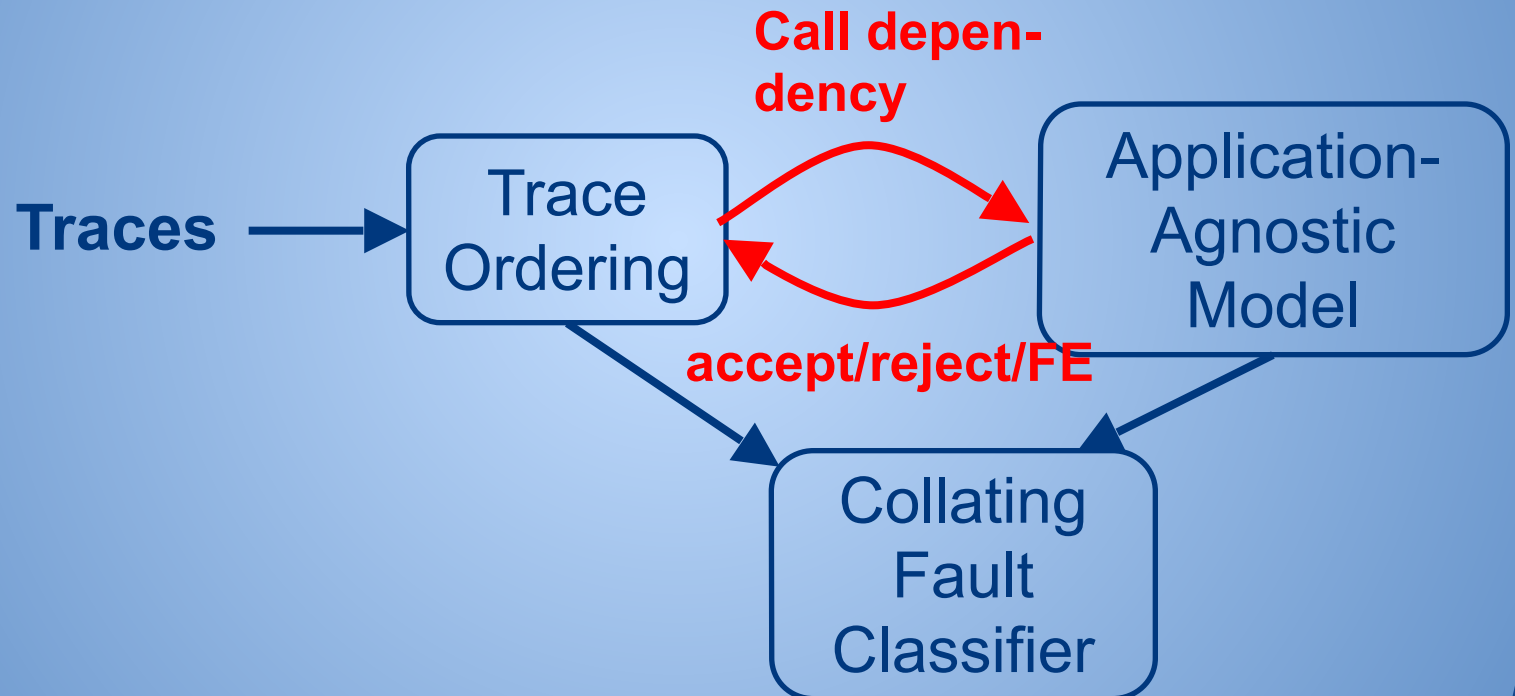
- Blackbox Tracing mechanism





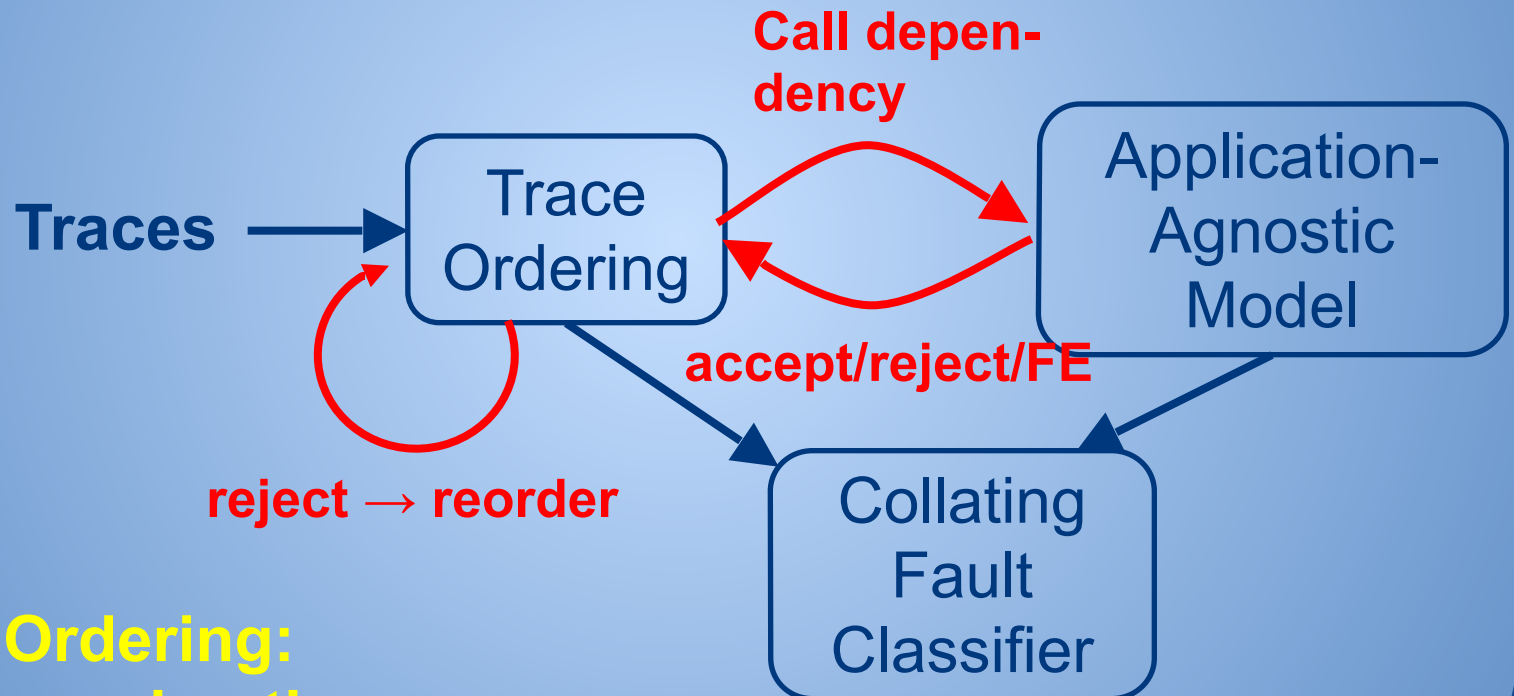
# Diagnosis Model

- Blackbox Tracing mechanism



# Diagnosis Model

- Blackbox Tracing mechanism



**Trace Ordering:  
linear running time**

# Pseudocode and Analysis

1. push trace  $t_0$  in stack  $s_0$ , ..., trace  $t_{n-1}$  in stack  $s_{n-1}$

2. while  $(s_0, \dots, s_{n-1})$  not empty:  **$O(L)$**

3.  $q = \text{peek\_stack}(s_0, \dots, s_{n-1});$   **$q.\text{sort}(\text{priority})$**

4. while True:

5. if  $q$  empty: raise FatalError

6.  $i_j = q.\text{dequeue}();$

7. **outcome = model\_simulate( $i_j$ )**

8. if outcome == ACCEPT:

9.  $\text{ordered\_trace.push}(s_j.\text{pop}());$  break

10. elif outcome == REJECT: pass

11. elif outcome == FatalError: raise FatalError

**Best case:  $O(1)$**   
**Worst case:  $O(n)$**

**Overall:**  
**Best case  $O(L)$**   
**Worst Case  $O(n*L)$**

# Pseudocode and Analysis

1. push trace  $t_0$  in list  $s_0$ , ..., trace  $t_{n-1}$  in list  $s_{n-1}$
2. while  $(s_0, \dots, s_{n-1})$  not empty:
3.  $q = \text{peek\_stack}(s_0, \dots, s_{n-1});$  **q.sort(priority)**
4. while True:
5. if q empty: raise FatalError
6.  $i_j = q.\text{dequeue}();$
7. **outcome = model\_simulate( $i_j$ )**
8. if **outcome == ACCEPT.** **Accept → Traverse**
9.  $\text{ordered\_trace.push}(s_j.\text{pop}());$  break
10. elif **outcome == REJECT:** continue **Reject → Backtrack**
11. elif **outcome == FatalError:** raise FatalError

# NetCheck input

**Syscall**

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

# NetCheck input

**Syscall**

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

# connect depends on listen

Order 1    A1    `bind(3, ...) = 0`  
            A2    `listen(3, 5) = 0`  
            B1    `connect(3, ...) = 0`

Order 2    A1    `bind(3, ...) = 0`  
            B1    `connect(3, ...) = -1 ECONNREFUSED`  
            A2    `listen(3, 5) = 0`

Order 3    B1    `connect(3, ...) = -1 ECONNREFUSED`  
            A1    `bind(3, ...) = 0`  
            A2    `listen(3, 5) = 0`

# Example Rules

- Middle boxes
  - Multiple unaccepted connections  
⇒ client behind NAT in FTP
  - Missing connect on accepted connections → server behind NAT or port forwarding
  - Multiple connect non-standard failure → firewall filtering connections
  - Multiple connect to listening address get refused
  - Multiple non-blocking connect failure
  - Traffic sent to non-relevant addresses → NAT or 3rd party proxy/traffic forwarding



# Example fault classifier rules

- Middle boxes
  - Multiple unaccepted connections  
⇒ *client behind NAT in FTP*
  - Missing connect on accepted connections → server behind NAT or port forwarding
  - Traffic sent to non-relevant addresses → NAT or 3rd party proxy/traffic forwarding
- TCP
  - select/poll timeout
  - send data after connection closed

# Example rules (cont.)

- UDP

- datagram sent/lost per connection
- high datagram loss rate

⇒ *non-transitive connectivity in VLC*

- Misc

- apps send data larger than default OS buffer size  
⇒ *bug report from VirtualBox bug tracker*

- returned IP different from bind  
⇒ *simultaneous net disconnect/reconnect in Pidgin*

- Skype attempted to close socket from a different thread

# Evaluation: Everyday Applications

- FTP
  - All reverse connections from server lost
    - Client behind NAT
- Pidgin
  - getsockname returns different IP
    - Client poor connection results in IP changes
- Skype
  - Poor call quality, msg drop
    - Network delay, NAT
    - Skype closes socket from different thread
- VLC
  - Packet loss
    - Non-transitive connectivity issue