Inferring Models of Concurrent Systems from Logs of Their Behavior with **CSight**

Ivan Beschastnikh  
Yuriy Brun  
Michael D. Ernst  
Arvind Krishnamurthy

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,0</td>
<td>2,1 M?m-0</td>
</tr>
<tr>
<td>2,0</td>
<td>2,2 recv(m)</td>
</tr>
<tr>
<td>3,3</td>
<td>2,3 A!a-0</td>
</tr>
<tr>
<td>4,3</td>
<td>5,4 M?m-1</td>
</tr>
<tr>
<td>5,3</td>
<td>5,5 recv(m)</td>
</tr>
<tr>
<td>6,6</td>
<td>5,6 A!a-1</td>
</tr>
<tr>
<td>7,6</td>
<td>8,7 M?m-0</td>
</tr>
<tr>
<td>8,6</td>
<td>8,8 recv(m)</td>
</tr>
<tr>
<td>9,9</td>
<td>8,9 A!a-0</td>
</tr>
<tr>
<td>10,9</td>
<td>11,10 M?m-1</td>
</tr>
<tr>
<td>11,9</td>
<td>11,11 recv(m)</td>
</tr>
<tr>
<td>12,12</td>
<td>11,12 A!a-1</td>
</tr>
</tbody>
</table>

(a) Sender  
(b) Receiver  

University of British Columbia  
UMass Amherst  
University of Washington
<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,0</td>
<td>2,1 M?m-0</td>
</tr>
<tr>
<td>2,0</td>
<td>2,2 recv(m)</td>
</tr>
<tr>
<td>3,3</td>
<td>2,3 A!a-0</td>
</tr>
<tr>
<td>4,3</td>
<td>5,4 M?m-1</td>
</tr>
<tr>
<td>5,3</td>
<td>5,5 recv(m)</td>
</tr>
<tr>
<td>6,6</td>
<td>5,6 A!a-1</td>
</tr>
<tr>
<td>7,6</td>
<td>8,7 M?m-0</td>
</tr>
<tr>
<td>8,6</td>
<td>8,8 recv(m)</td>
</tr>
<tr>
<td>9,9</td>
<td>8,9 A!a-0</td>
</tr>
<tr>
<td>10,9</td>
<td>11,10 M?m-1</td>
</tr>
<tr>
<td>11,9</td>
<td>11,11 recv(m)</td>
</tr>
<tr>
<td>12,12</td>
<td>11,12 A!a-1</td>
</tr>
</tbody>
</table>

(a) Sender

(b) Receiver

Input log

Output model

CSight
Challenges of logging for debugging

- Logs present a low-level view of the system
- Developers are not sure what to look for, or where
- Reasoning about logged concurrency is a nightmare
- Must connect clues across host and execution logs
Models from observations

Model inference

Input

...
Models from observations

Prior work:

- Cook et al. TSE 1998
- Lorenzoli et al. ICSE 2008
- Lo et al. ASE 2010
- Beschastnikh et al. FSE 2011
- Ghezzi et al. ICSE 2014

Model inference

- Specification mining
- Process discovery

Invalid

Read Only

Write

Input

Output
Using inferred models

Applications:

- Mental model validation
- Test case generation
- Evaluating test suites
- Anomaly detection
- Log summarization
- Verifying a code fix

Model inference

Input

Output

Invalid

Write

Read Only
### Challenges:
- Modeling concurrency
- Accuracy
- Efficiency

### Contributions:
- Communicating FSM inference
- Refinement
- Leveraging mined invariants

---

**Input**

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>src: 0, dst: 2, timestamp: 22, type: ack</td>
<td>src: 0, dst: 2, timestamp: 22, type: ack</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 19, type: commit</td>
<td>src: 1, dst: 2, timestamp: 19, type: commit</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 18, type: commit</td>
<td>src: 0, dst: 2, timestamp: 18, type: commit</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 17, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 17, type: prepare</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 15, type: ack</td>
<td>src: 1, dst: 2, timestamp: 15, type: ack</td>
</tr>
<tr>
<td>src: 2, dst: 0, timestamp: 8, type: prepare</td>
<td>src: 2, dst: 0, timestamp: 8, type: prepare</td>
</tr>
<tr>
<td>src: 2, dst: 0, timestamp: 4, type: tx_commit</td>
<td>src: 2, dst: 0, timestamp: 4, type: tx_commit</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 1, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 1, type: prepare</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 19, type: commit</td>
<td>src: 1, dst: 2, timestamp: 19, type: commit</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 18, type: commit</td>
<td>src: 0, dst: 2, timestamp: 18, type: commit</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 17, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 17, type: prepare</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 15, type: ack</td>
<td>src: 1, dst: 2, timestamp: 15, type: ack</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 13, type: tx_commit</td>
<td>src: 2, dst: 1, timestamp: 13, type: tx_commit</td>
</tr>
<tr>
<td>src: 2, dst: 0, timestamp: 12, type: tx_commit</td>
<td>src: 2, dst: 0, timestamp: 12, type: tx_commit</td>
</tr>
<tr>
<td>src: 2, dst: 0, timestamp: 8, type: prepare</td>
<td>src: 2, dst: 0, timestamp: 8, type: prepare</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 5, type: tx_commit</td>
<td>src: 2, dst: 1, timestamp: 5, type: tx_commit</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 2, type: commit</td>
<td>src: 0, dst: 2, timestamp: 2, type: commit</td>
</tr>
<tr>
<td>src: 2, dst: 0, timestamp: 0, type: prepare</td>
<td>src: 2, dst: 0, timestamp: 0, type: prepare</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 23, type: ack</td>
<td>src: 1, dst: 2, timestamp: 23, type: ack</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 21, type: tx_commit</td>
<td>src: 2, dst: 1, timestamp: 21, type: tx_commit</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 18, type: commit</td>
<td>src: 0, dst: 2, timestamp: 18, type: commit</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 17, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 17, type: prepare</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 15, type: ack</td>
<td>src: 1, dst: 2, timestamp: 15, type: ack</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 9, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 9, type: prepare</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 6, type: ack</td>
<td>src: 0, dst: 2, timestamp: 6, type: ack</td>
</tr>
<tr>
<td>src: 2, dst: 0, timestamp: 4, type: tx_commit</td>
<td>src: 2, dst: 0, timestamp: 4, type: tx_commit</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 2, type: commit</td>
<td>src: 0, dst: 2, timestamp: 2, type: commit</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 1, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 1, type: prepare</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 5, type: tx_commit</td>
<td>src: 2, dst: 1, timestamp: 5, type: tx_commit</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 3, type: commit</td>
<td>src: 1, dst: 2, timestamp: 3, type: commit</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 1, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 1, type: prepare</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 22, type: ack</td>
<td>src: 0, dst: 2, timestamp: 22, type: ack</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 21, type: tx_commit</td>
<td>src: 2, dst: 1, timestamp: 21, type: tx_commit</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 18, type: commit</td>
<td>src: 0, dst: 2, timestamp: 18, type: commit</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 17, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 17, type: prepare</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 15, type: ack</td>
<td>src: 1, dst: 2, timestamp: 15, type: ack</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 14, type: ack</td>
<td>src: 0, dst: 2, timestamp: 14, type: ack</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 13, type: tx_commit</td>
<td>src: 2, dst: 1, timestamp: 13, type: tx_commit</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 10, type: commit</td>
<td>src: 0, dst: 2, timestamp: 10, type: commit</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 9, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 9, type: prepare</td>
</tr>
<tr>
<td>src: 2, dst: 0, timestamp: 8, type: prepare</td>
<td>src: 2, dst: 0, timestamp: 8, type: prepare</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 7, type: ack</td>
<td>src: 1, dst: 2, timestamp: 7, type: ack</td>
</tr>
<tr>
<td>src: 0, dst: 2, timestamp: 6, type: ack</td>
<td>src: 0, dst: 2, timestamp: 6, type: ack</td>
</tr>
<tr>
<td>src: 2, dst: 0, timestamp: 4, type: tx_commit</td>
<td>src: 2, dst: 0, timestamp: 4, type: tx_commit</td>
</tr>
<tr>
<td>src: 1, dst: 2, timestamp: 3, type: commit</td>
<td>src: 1, dst: 2, timestamp: 3, type: commit</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 1, type: prepare</td>
<td>src: 2, dst: 1, timestamp: 1, type: prepare</td>
</tr>
<tr>
<td>src: 2, dst: 1, timestamp: 5, type: tx_commit</td>
<td>src: 2, dst: 1, timestamp: 5, type: tx_commit</td>
</tr>
</tbody>
</table>

---

**Output**

- **Process 1**
  - 0 → tx m → 1,
  - rx ack

- **Process 2**
  - 0 → rx m → 1,
  - tx ack

---

**Contributors**

- Cook et al. FSE 1998
- Kumar et al. ICSE 2012

---

*University of British Columbia*

---

Ivan Beschastnikh
Challenges:
- Modeling concurrency
- Accuracy
- Efficiency

Contributions:
- Communicating FSM inference
- Refinement
- Leveraging mined invariants

Key results:
- CSight infers concise models of distributed systems: distributed hash table, and TCP
- User study with 39 students shows that CFM models are helpful in findings bugs
- Proved termination/progress properties
Motivation

Background

• Logging: partial order and vector clocks
• Modeling: communicating FSMs
• CSight approach
• Evaluation

http://synoptic.googlecode.com
Limitations of total order

- A system with two threads: T1, T2
  - T1 generates event \( a \), T2 generates event \( b \)

Logging pipeline:

Generated log file:

Which of these two systems generated the log?

Two possible systems:
Limitations of total order

- A system with two threads: T1, T2
  - T1 generates event \( a \), T2 generates event \( b \)

- Logging pipeline:

- Generated log file:

A totally ordered log is insufficient.
Logging the partial order

- We know how to do this
  - Lamport defined the happens-before relation in 1978
  - Operationalized with vector clocks in Fidge, ACSS 1988 Mattern, PDA 1989

- Vector clocks capture the partial order of an execution
  - Track dependencies between processes
  - Tell us if two events are concurrent, or if one occurs before the other.
Logging the partial order

- We know how to do this
  - Lamport defined the happens-before relation in 1978
  - Operationalized with vector clocks in Fidge, ACSS 1988 and Mattern, PDA 1989

Diagram:

- T1:
  - a
  - b

- T2:
  - a
  - b
Vector clock instrumentation

- Automatically instruments a distributed system logging with vector clock information
- Transparently interposes on all socket communication and logging (via AspectJ)

Network

https://bitbucket.org/bestchai/shivector
Vector clock instrumentation

- **Automatically instruments** a distributed system logging with vector clock information
- **Transparency** interposes on all socket communication and logging (via AspectJ)

```
System + ShiVector
```

```
INFO Timeline request
GET /timeline
INFO New status
POST status="Lunch"
```

```
INFO Timeline request [ 3, 10, 7, 2 ]
GET /timeline [ 3, 11, 7, 2 ]
INFO New status [ 5, 12, 10, 2 ]
POST status="Lunch" [ 5, 13, 10, 2 ]
```

https://bitbucket.org/bestchai/shivector
CSight talk outline

• Motivation
• Background
  • Logging: partial order and vector clocks
  • Modeling: communicating FSMs
• CSight approach
• Evaluation

http://synoptic.googlecode.com
Extending finite state machines

Sequential:

Concurrent:
Extending finite state machines

Sequential:

Concurrent:

Process 1

Process 2
Communicating FSM formalism

Process 1

Q1

Q2

Process 2
Communicating FSM formalism

Process 1

- `s1` to `s2`:
  - Send `m` on `Q1`
  - Receive `ack` on `Q2`

Process 2

- `r1` to `r2`:
  - Receive `m` on `Q1`
  - Local event
- `r2` to `r3`:
  - Send `ack` on `Q2`

- `r3` to `r4`:
  - Send `ack` on `Q2`
Communicating FSM formalism

Process 1

- send m on $Q1$ = $Q1!m$
- receive ack on $Q2$ = $Q2?ack$

Process 2

- receive m on $Q1$ = $Q1?m$
- local event
- send ack on $Q2$ = $Q2!ack$

Diagram arrows:
- $m$
- $Q1$
- $Q2$
Communicating FSM formalism

Process 1

send m on Q1 = Q1!m
receive ack on Q2 = Q2?ack

Process 2

local event
receive m on Q1 = Q1?m
send ack on Q2 = Q2!ack
Communicating FSM formalism

Process 1

send m on Q1 = Q1!m
receive ack on Q2 = Q2?ack

Process 2

r1
Q1?m = receive m on Q1
local event
r2
Q2!ack = send ack on Q2
r3
r4
Communicating FSM formalism

Process 1

send m on Q1 = Q1!m
receive ack on Q2 = Q2?ack

Process 2

Q1
receive m on Q1 = Q1?m
local event
Q2
send ack on Q2 = Q2!ack
Communicating FSM formalism

Process 1

send m on Q1 = Q1!m
receive ack on Q2 = Q2?ack

Process 2

local event
receive m on Q1 = r1
send ack on Q2 = r4
CSight talk outline

• Motivation

• Background
  • Logging: partial order and vector clocks
  • Modeling: communicating FSMs

• CSight approach

• Evaluation

http://synoptic.googlecode.com
Goal

CSight

Input

Output
CSight approach

1. Parse the log with user-defined reg. expressions

2. Build a compact model

3. Mine salient log invariants

4. Refine model to satisfy invariants

e.g.,
“send() at Process 1 is always followed by receive() at Process 2”

Input

Output

Initial model

Process 1

Process 2

0

1

tx m

rx ack

0

1

tx ack
1. Parse the log with user-defined reg. expressions

2. Build a compact model

3. Mine salient log invariants

4. Refine model to satisfy invariants

Initial model

Invariants

"send() at Process 1 is always followed by receive() at Process 2"
From logs to execution **DAGs**

Input

Process 0

Process 1

Time-space DAGs

Fidge, ACSS 1988 | Mattern, PDA 1989
From logs to execution DAGs

Input

regular expressions

Time-space DAGs

Fidge, ACSS 1988  Mattern, PDA 1989
From logs to execution DAGs

[1,0] Q1!m
[1,1] Q1?m
[1,2] local event
[1,3] Q2!ack
[2,3] Q2?ack

Input

regular expressions

Time-space DAGs

Fidge, ACSS 1988  Mattern, PDA 1989
From logs to execution DAGs

[1,0] Q1!m
[1,1] Q1?q
[1,2] local event
[1,3] Q2!ack
[2,3] Q2?q

Process 0
Process 1

regular expressions

Input

Time-space DAGs

Fidge, ACSS 1988
Mattern, PDA 1989
CSight approach

1. Parse the log with user-defined reg. expressions

Input

Initial model

2. Build a compact model

3. Mine salient log invariants

e.g.,

“send() at Process 1 is always followed by receive() at Process 2”

4. Refine model to satisfy invariants

Output
**CSight approach**

1. Parse the log with user-defined regex expressions

2. Build a compact model
   - 2.1 Recover concrete states
   - 2.2 Abstract states

```
Process 1
0 -> tx m -> 1
    rx ack

Process 2
0 -> rx m -> 1
    tx ack
```

**e.g.,**

"send() at Process 1 is always followed by receive() at Process 2"
Recover concrete queue states

Time-Space DAG

Process 1

Q1!m

Q1?m

local event

Q2!ack

Q2?ack

Process 2

State-based DAG
Recover concrete queue states

Time-Space DAG

State-based DAG
Recover concrete queue states

Time-Space DAG

State-based DAG

Q1!m
Q1?m
local event
Q2!ack
Q2?ack

Q1: [ ] Q2: [ ]
Q1!m
Q1: [m] Q2: [ ]
Recover concrete queue states

Time-Space DAG

State-based DAG

Process 1  Process 2

Q1!m

Q1?m

local event

Q2!ack

Q2?ack

Q1: [ ] Q2: [ ]

Q1!m

Q1: [m] Q2: [ ]

Q1?m

Q1: [ ] Q2: [ ]
Recover concrete queue states

Time-Space DAG

State-based DAG
Recover concrete queue states
Abstract concrete queue states

State-based DAGs
Abstract concrete queue states

State-based DAGs

Initial global model
“Merge” concrete states into abstract states

State-based DAGs

Initial global model
“Merge” concrete states into abstract states

State-based DAGs

Initial global model
“Merge” concrete states into abstract states

State-based DAGs

Initial global model

Q1: [m]
Q2: [ ]

Q1: [ ]
Q2: [ ]

Local event

Q1: [ ]
Q2: [ ]

Q2!ack

Q1: [ ]
Q2: [ack]

Q2?ack

Q1: [ ]
Q2: [ ]

Q2?ack

Q1: [m]
Q2: [ ]

Q1: [ ]
Q2: [ ]
Generate abstract edges

State-based DAGs

Initial global model
Generate abstract edges

State-based DAGs

Initial global model
Generate abstract edges

State-based DAGs

Initial global model
CSight approach

1. Parse the log with user-defined reg. expressions

2. Build a compact model
   2.1 Recover concrete states
   2.2 Abstract states

Refine model to satisfy invariants

Input

Output

```
<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>tx m 1</td>
<td>rx ack</td>
</tr>
<tr>
<td>rx m 1</td>
<td>tx ack</td>
</tr>
</tbody>
</table>
```

e.g.,

“send() at Process 1 is always followed by receive() at Process 2”
CSight approach

1. Parse the log with user-defined regular expressions

2. Build a compact model

3. Mine salient log invariants

4. Refine model to satisfy invariants

Input

Process 1

Q1!m

Q1?m

local event

Q2!ack

Q2?ack

Process 2

Q1!m \rightarrow Q1 ?m
always followed by

Q1?m \leftarrow Q2 ?ack
always precedes

Q1?m \rightarrow Q1 !m
never followed by

Dwyer et al. ICSE 1999
Beschastnikh et al. OSR 2011
CSight approach

1. Parse the log with user-defined reg. expressions
2. Build a compact model
3. Mine salient log invariants
4. Refine model to satisfy invariants

Input

Output

Process 1

Process 2

Q1!m → Q1?m
always followed by
Q1?m ← Q2?ack
always precedes
Q1?m → Q1!m
never followed by
Deriving a Communicating FSM

- Decompose the global model into per-process FSMs

Compact global model

CFSM model
Making the compact model accurate

Check an invariant in the CFSM

Q1!m → Q1?m

Process 1

Process 2

Heußner et al. TACAS 2012. (McScM)
Holzmann TSE 1997. (SPIN)
Making the compact model accurate

Refine global model to remove invariant counter-example

Check an invariant in the CFSM

1. Q1!m
2. local event

Q1!m → Q1?m

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>t0</td>
</tr>
<tr>
<td>Q1!m</td>
<td>Q1?m</td>
</tr>
<tr>
<td>Q2?ack</td>
<td>Q2!ack</td>
</tr>
</tbody>
</table>

Local event

1. Q1!m
2. local event
Making the compact model accurate

Refine global model to remove invariant counter-example

Check an invariant in the CFSM

1. Q1!m
2. local event

Q1!m $\rightarrow$ Q1?m

Process 1

Process 2
Making the compact model accurate

Refine global model to remove invariant counter-example

Check an invariant in the CFSM

Process 1

Process 2

local event
Making the compact model accurate

Refine global model to remove invariant counter-example

Check invariants in the CFSM

Counter-example

All invariants satisfied

Final model
Making the compact model accurate

Refine global model to remove invariant counter-example

Check invariants in the CFSM

Counter-example

See the paper for more details!
CSight talk outline

- Motivation
- Background
  - Logging: partial order and vector clocks
  - Modeling: communicating FSMs
- CSight approach
- Evaluation

http://synoptic.googlecode.com
CSight evaluation

- Efficiency and correctness (proof)

- Utility (case studies and user study)
  - Voldemort DHT replication protocol
  - TCP opening/closing handshakes
  - User study: students found bugs using CFSM models

---

![Diagram of CSight processes and input/output]

Process 1

0 -> tx m -> 1

rx ack

Process 2

0 -> rx m -> 1

tx ack

Input

Output
Voldemort Distributed Hash Table

- Voldemort  
  DeCandia et al. SOSP 2007
- Implements a distributed hash table
  - put(key, value), get(key)
- Deployed at LinkedIn
- Code is the best protocol documentation
- Ran CSight on logs generated with unit tests
  - Targeted replication protocol messages
  - No modifications to Voldemort
Voldemort replication protocol

Replica 1:

Client:

Replica 2:
Voldemort replication protocol

Replica 1:

Client:

Replica 2:

put(key, value) execution path
Voldemort replication protocol

Replica 1:

get(key)
execution path

Client:

Replica 2:
Voldemort replication protocol

Replica 1:

- CSight uncovered the true model
- Model succinctly captures a 3-node distributed execution

Replica 2:
TCP opening/closing handshake

- Log generated using netcat/dummynet/tcpdump
- Semi-automatically annotated with vector timestamps
- CSight ran on traces with opening/closing handshakes

(a) TCP server

(b) TCP client
TCP opening/closing handshake

- Log generated using netcat/dummynet/tcpdump
- Semi-automatically annotated with vector timestamps
- CSight ran on traces with opening/closing handshakes

Server-initiated teardown
TCP opening/closing handshake

- Log generated using netcat/dummynet/tcpdump
- Semi-automatically annotated with vector timestamps
- CSight ran on traces with opening/closing handshakes

Client-initiated teardown
TCP opening/closing handshake

- TCP logs generated using netcat/dummynet/tcpdump
- Semi-automatic annotation with vector timestamps
- Ran CSight on traces with opening/closing handshakes

- Customized level of abstraction
- False positive transition, limited expressiveness of the model-checker
User study on CFSM models

- Compared efficacy of CFSM models and space-time diagrams in bug finding
- Two buggy systems, 39 student participants
- Each participant took a mini-tutorial before bug finding

Example model

Example space-time diagrams
User study on CFSM models

• Compared efficacy of CFSM models and space-time diagrams in bug finding

• Two buggy systems, 39 student participants

• Each participant took a mini-tutorial before bug finding

• Students found bugs as well with CFSM models as with space-time diagrams

• Oral feedback indicates that students preferred CFSM models:
  • Could follow process states
  • More concise and easier to follow than long space-time diagrams

See the paper for study details!
CSight limitations

- Quality and usefulness of the model depends on logged information
- May be difficult to interpret a log you did not generate
- Dynamic analysis: executions may not be representative of the system
- Limited property diversity (can add more)
- Focused on model conciseness, but small model size may be irrelevant
CSight contributions

Challenges:
- Modeling concurrency
- Accuracy
- Efficiency

Contributions:
- Communicating FSM inference
- Refinement
- Leveraging mined invariants
## CSight contributions

<table>
<thead>
<tr>
<th>Challenges:</th>
<th>Contributions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logs are rich with information, but inscrutable</td>
<td>- Communicating FSM inference</td>
</tr>
<tr>
<td>• Modeling concurrency</td>
<td>- Refinement</td>
</tr>
<tr>
<td>• Accuracy</td>
<td>- Leveraging mined invariants</td>
</tr>
<tr>
<td>• Efficiency</td>
<td></td>
</tr>
</tbody>
</table>

- Elucidates distributed protocols
- Beginner developers can find bugs with CFSMs
- Proved useful properties about the approach

- Open source: [synoptic.googlecode.com](http://synoptic.googlecode.com)