Who Goes First? Detecting Go Concurrency Bugs via Message Reordering

Ziheng Liu*, Shihao Xia*, Yu Liang, Linhai Song and Hong Hu

Pennsylvania State University

* Ziheng Liu and Shihao Xia are co-first authors
Go Programming Language

• A young but widely-used programming lang.

• Designed for efficient and *reliable* concurrency
  – Provide lightweight threads, called goroutines
  – Support both *message passing* and shared memory
Many concurrency bugs are in Go programs

Channel may be more error-prone than mutex

36% more blocking bugs are due to channels [1]

A Concurrency Bug in Docker

Parent Goroutine

```go
func parent() {
    ...
    ch, errCh := dis.Watch()
    select {
        case <- Fire(1 * T.Second):
            Log("Timeout!")
        case e := <- ch:
            ...
        case e := <- errCh:
            Log("Error!")
    }
    return
}
```

```go
func (s *Discover) Watch() (..., ch chan Entries, errCh chan error) {
    ch := make(chan Entries)
    errCh := make(chan error)
    go func() {
        entries, err := s.fetch()
        if err != nil {
            errCh <- err
        } else {
            ch <- entries
        }
    }()
    return ch, errCh
}
```
A Concurrency Bug in Docker

Parent Goroutine

```go
func parent() {
    ... 
    ch, errCh := dis.Watch()
    select {
        case e := <- ch:
            ... 
        case e := <- errCh:
            Log("Error!")
    }
    return 
}
```

Child Goroutine

```go
func (s *Discover) Watch() (...) {
    ch := make(chan Entries)
    errCh := make(chan error)
    go func() {
        entries, err := s.fetch()
        if err != nil {
            errCh <- err
        } else {
            ch <- entries
        }
    }()
    return ch, errCh
}
```

After 1 second

A msg from the child

Send a message
A Concurrency Bug in Docker

Parent Goroutine

```go
func parent() {
    ...
    ch, errCh := dis.Watch()
    select {
    case <- Fire(1 * T.Second):
        Log("Timeout!")
    case e := <- ch:
        ...
    case e := <- errCh:
        Log("Error!")
    }
    return
}
```

Child Goroutine

```go
func (s *Discover) Watch() (...) {
    ch := make(chan Entries)
    errCh := make(chan error)
    go func() {
        entries, err := s.fetch()
        if err != nil {
            errCh <- err
        } else {
            ch <- entries
            ...
        }
    }()
    return ch, errCh
}
```
Limitations of Existing Techniques

Parent Goroutine

```go
func parent() {
    ...
    ch, errCh := dis.Watch()
    select {
    case <- Fire(1 * T.Second):
        Log("Timeout!")
    case e := <- ch:
        ...
    case e := <- errCh:
        Log("Error!")
    }
    return
}
```

Child Goroutine

```go
func (s *Discover) Watch() (...) {
    ch := make(chan Entries)
    errCh := make(chan error)
    go func() {
        entries, err := s.fetch()
        if err != nil {
            errCh <- err
        } else {
            ch <- entries
        }
    }()
    return ch, errCh
}
```

- cannot resolve indirect function calls
- don’t increase the chance of exposing concurrency bugs
- do not analyze channel operations
Intuitions

• Building a dynamic technique
  – To avoid limitations of static analysis

• Focusing on concurrent messages
  – Their processing order is non-deterministic
    • Some orders may not be carefully implemented
  – Mutating their processing order to detect bugs

```javascript
select {
  case <- Fire(1 * T.Second):
    Log("Timeout!")
  case e := <- ch:
    ...
  case e := <- errCh:
    Log("Error!")
}
```
Challenges

• How to identify concurrent messages?

O : channel operation
Challenges

• How to identify concurrent messages?
• How to identify suspicious message orders?
• How to capture triggered channel-related bugs?

Goroutine1  Goroutine2  Goroutine3

O : channel operation

Order space

Suspicious orders
Contributions

• GFuzz: a dynamic Go concurrency bug detector
  – Use `select` to identify concurrent messages
  – Leverage fuzzing to pinpoint suspicious msg orders
  – Propose a novel sanitizer to capture triggered bugs

• Thorough experiments to evaluate GFuzz
  – Detect 184 previously unknown bugs
    • Developers have confirmed 124 bugs and fixed 67 bugs
  – Detect significantly more bugs than SOTA
Outline

• Introduction
• Reordering Concurrent Messages
• Favoring Propitious Orders
• Capturing Triggered Concurrency Bugs
• Implementation and Evaluation
• Conclusion
Outline

• Introduction
• Reordering Concurrent Messages
• Favoring Propitious Orders
• Capturing Triggered Concurrency Bugs
• Implementation and Evaluation
• Conclusion
Concurrent Channel Operations (Ops)

• Channel ops have no happens-before relation
  – Their processed messages are concurrent
  – Challenging to find all concurrent channel ops

• GFuzz focuses on select statements
  – Select allows a goroutine to wait for >1 channel ops
  – Channel ops within the same select are concurrent

```go
select {
    case <- Fire(1 * T.Second):
        Log("Timeout!")
    case e := <- ch:
        ...
    case e := <- errCh:
        Log("Error!")
}
```

*can happen simultaneously*
Encode Concurrent Message Orders

• Use selected cases to represent an order
  – Assign each select a unique ID
  – Allocate a local index to each case
  – An order $\rightarrow [(s_0, c_0, e_0) \ldots (s_n, c_n, e_n)]$

```
select {
  case Fire(1 * T.Second):
    Log("Timeout!")
  case e := <- ch:
    ...
  case e := <- errCh:
    Log("Error!")
}
```

- selected case with ID 0
- the number of cases
- chosen twice
Code Transformation

```java
switch FetchOrder(...) {
    // 3 cases

    select {
        case <- Fire(1 * T.Second):
            Log("Timeout!")
        case e := <- ch:
            ...  // 3 cases
        case e := <- errCh:
            Log("Error!")
    }

    // 3 cases
```
Code Transformation

```
select {
  case Fire(1 * T.Second):
    Log("Timeout!"")
  case e := ch:
    ...
  case e := errCh:
    Log("Error!")
}
```

```
switch FetchOrder(...) {
  case 0:
  case 1:
  case 2:
  default:
}
```

3 cases

3 cases + 1

default
switch FetchOrder(...) {
  case 0:
  
  select {
    case Fire(1 * T.Second):
      Log("Timeout!")
    case e := <- ch:
      ...
    case e := <- errCh:
      Log("Error!")
  }
  
  case 1:
  
  case 2:
  default:
    no order is specified
  }
}
```plaintext
switch FetchOrder(...) {
    case 0:
        select {
            case Fire(1 * T.Second):
                Log("Timeout!")
            case time.After(T):
                ..... 
        }
    case 1:
    case 2:
    default:
}
```
Code Transformation

```java
switch FetchOrder(...) {
    case 0:
        select {
            case <- Fire(1 * T.Second):
                Log(“Timeout!”)
            case <- time.After(T):
                ....
        }
    case 1:
    case 2:
    default:
}
```
switch FetchOrder(...) {
    case 0:
        select {
            case <- Fire(1 * T.Second):
                Log(“Timeout!”)
            case <- time.After(T):
                ......
        }
    case 1:
    case 2:
    default:
}
Code Transformation

```rust
define the message arrives within T
```

```rust
switch FetchOrder(...) {
  case 0:
    select {
      case fire(T.Second):
        Log("Timeout!")
      case time.After(T):
        ...
    }
  case 1:
  case 2:
  default:
}
```

```rust
define the message does not arrive within T
```
Code Transformation

```
select {
  case Fire(1 * T.Second):
    Log("Timeout!")
  case e := ch:
    ...
  case e := errCh:
    Log("Error!")
}

switch FetchOrder(...) {
  case 0:
    select {
      case Fire(1 * T.Second):
        Log("Timeout!")
      case time.After(T):
        ...
    }
  case 1:
    select {
      case e := ch:
        ...
      case time.After(T):
        ...
    }
  case 2:
    ...
  default:
    ...
}
```
Outline

- Introduction
- Reordering Concurrent Messages
- Favoring Propitious Orders
- Capturing Triggered Concurrency Bugs
- Implementation and Evaluation
- Conclusion
Fuzzing Message Orders

- Continuously mutating exercised orders
- Enforcing new orders to detect bugs

FIFO Order Queue -> Mutator -> New orders -> Instrumented Go program -> Execution feedback -> Bugs
Interesting Orders

• Triggering a new interleaving of channel ops
• Reaching a new channel state
  – New channel creation, new channel closing
  – Maximum fullness
Prioritizing Valuable Orders

\[ \text{score} = \sum \log_2 \text{CountChOpPair} + 10 \times \#\text{CreateCh} \]
\[ + 10 \times \#\text{CloseCh} + 10 \times \sum \text{MaxChBufFull} \]

- number of triggered channel interleavings
- number of distinct channels closed
- number of distinct channel created
- sum of max channel fullness

FIFO Order Queue → Mutator → New orders → Instrumented Go program → Execution feedback

Bugs
Outline

• Introduction
• Reordering Concurrent Messages
• Favoring Propitious Orders
• Capturing Triggered Concurrency Bugs
• Implementation and Evaluation
• Conclusion
Sanitizer Design

• Focus on channel-related blocking bugs
  – Go runtime captures non-blocking bugs

• Hybridize two methods to record dynamic info.
  – Source-to-source instrumentation
  – Modifying the Go runtime

```go
go func() {
    GainChRef(ch)
    GainChRef(errCh)
    entries, err := s.fetch()
    if err != nil {
        errCh <- err
    } else {
        ch <- entries
    }
}
```
### Blocking Bug Detection

<table>
<thead>
<tr>
<th>goroutine</th>
<th>channel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>channel</th>
<th>goroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
func parent() {
    ... 
    ch, errCh := dis.Watch()
    select {
        case <- Fire(1 * T.Second):
            Log("Timeout!")
        case e := <- ch: 
            ... 
        case e := <- errCh: 
            Log("Error!")
    }
    return 
}

func (s *Discover) Watch() (...) {
    ch := make(chan Entries)
    errCh := make(chan error)
    go func() {
        entries, err := s.fetch()
        if err != nil {
            errCh <- err
        } else {
            ch <- entries
        }
    }()
    return ch, errCh
func parent() {
    ... 
    ch, errCh := dis.Watch()
    select {
        case <- Fire(1 * T.Second):
            Log("Timeout!")
        case e := <- ch:
            ...
        case e := <- errCh:
            Log("Error!")
    }
    return
}

func (s *Discover) Watch() (...) {
    ch := make(chan Entries)
    errCh := make(chan error)
    go func() {
        entries, err := s.fetch()
        if err != nil {
            errCh <- err
        } else {
            ch <- entries
        }
    }()
    return ch, errCh
}
Blocking Bug Detection

GoInfo

<table>
<thead>
<tr>
<th>goroutine</th>
<th>channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>ch, errCh</td>
</tr>
<tr>
<td>child</td>
<td>ch, errCh</td>
</tr>
</tbody>
</table>

ChInfo

<table>
<thead>
<tr>
<th>channel</th>
<th>goroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch</td>
<td>parent, child</td>
</tr>
<tr>
<td>errCh</td>
<td>parent, child</td>
</tr>
</tbody>
</table>

```
func parent() {
    ...
    ch, errCh := dis.Watch()
    select {
        case <- Fire(1 * T.Second):
            Log(“Timeout!”)
        case e := <- ch:
            ...
        case e := <- errCh:
            Log(“Error!”)
    }
    return
}
```

```
func (s *Discover) Watch() (...) {
    ch := make(chan Entries)
    errCh := make(chan error)
    go func() {
        entries, err := s.fetch()
        if err != nil {
            errCh <- err
        } else {
            ch <- entries
        }
    }()
    return ch, errCh
}
### Blocking Bug Detection

<table>
<thead>
<tr>
<th>goroutine</th>
<th>channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>child</td>
<td>ch, errCh</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>channel</th>
<th>goroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch</td>
<td>child</td>
</tr>
<tr>
<td>errCh</td>
<td>child</td>
</tr>
</tbody>
</table>

```go
func parent() {
    ...
    ch, errCh := dis.Watch()
    select {
        case <- Fire(1 * T.Second):
            Log("Timeout!")
        case e := <- ch:
            ...
        case e := <- errCh:
            Log("Error!")
    }
    return
}
```

```go
func (s *Discover) Watch() (...) { 
    ch := make(chan Entries)
    errCh := make(chan error)
    go func() {
        entries, err := s.fetch()
        if err != nil {
            errCh <- err
        } else {
            ch <- entries
        }
    }()
    return ch, errCh
}
```

*blocking bug!!*
Outline

• Introduction
• Reordering Concurrent Messages
• Favoring Propitious Orders
• Capturing Triggered Concurrency Bugs
• Implementation and Evaluation
• Conclusion
Implementation & Evaluation

• Implementing GFuzz using Go-1.16
  – Leveraging the SSA and AST packages
  – Modifying the Go runtime

• Applying to the recent versions of 7 Go apps

<table>
<thead>
<tr>
<th>App</th>
<th>Star</th>
<th>LoC</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes</td>
<td>74K</td>
<td>3453K</td>
<td>3176</td>
</tr>
<tr>
<td>Docker</td>
<td>60K</td>
<td>1105K</td>
<td>1227</td>
</tr>
<tr>
<td>Prometheus</td>
<td>35K</td>
<td>1186K</td>
<td>570</td>
</tr>
<tr>
<td>Etcd</td>
<td>35K</td>
<td>181K</td>
<td>452</td>
</tr>
<tr>
<td>Go-Ethereum</td>
<td>28K</td>
<td>368K</td>
<td>1622</td>
</tr>
<tr>
<td>TiDB</td>
<td>27K</td>
<td>476K</td>
<td>264</td>
</tr>
<tr>
<td>gRPC</td>
<td>13K</td>
<td>117K</td>
<td>888</td>
</tr>
</tbody>
</table>
### Effectiveness of Bug Detection

- 184 previously unknown bugs

<table>
<thead>
<tr>
<th>App</th>
<th>$\text{chan}_b$</th>
<th>$\text{select}_b$</th>
<th>$\text{range}_b$</th>
<th>NBK</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes</td>
<td>28</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>Docker</td>
<td>17</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>19</td>
</tr>
<tr>
<td>Prometheus</td>
<td>14</td>
<td>-</td>
<td>1</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>Etcd</td>
<td>7</td>
<td>12</td>
<td>-</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Go-Ethereum</td>
<td>11</td>
<td>43</td>
<td>6</td>
<td>2</td>
<td>62</td>
</tr>
<tr>
<td>TiDB</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>gRPC</td>
<td>15</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>22</td>
</tr>
</tbody>
</table>
Effectiveness of Bug Detection

- 184 previously unknown bugs
  - 170 blocking bugs

<table>
<thead>
<tr>
<th>App</th>
<th>(\text{chan}_b)</th>
<th>(\text{select}_b)</th>
<th>(\text{range}_b)</th>
<th>NBK</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes</td>
<td>28</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>Docker</td>
<td>17</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>19</td>
</tr>
<tr>
<td>Prometheus</td>
<td>14</td>
<td>-</td>
<td>1</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>Etcd</td>
<td>7</td>
<td>12</td>
<td>-</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Go-Ethereum</td>
<td>11</td>
<td>43</td>
<td>6</td>
<td>2</td>
<td>62</td>
</tr>
<tr>
<td>TiDB</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>gRPC</td>
<td>15</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>22</td>
</tr>
</tbody>
</table>
Effectiveness of Bug Detection

• 184 previously unknown bugs
  – 170 blocking bugs

<table>
<thead>
<tr>
<th>App</th>
<th>chan\textsubscript{b}</th>
<th>select\textsubscript{b}</th>
<th>range\textsubscript{b}</th>
<th>NBK</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes</td>
<td>28</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>Docker</td>
<td>17</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>19</td>
</tr>
<tr>
<td>Prometheus</td>
<td>14</td>
<td>-</td>
<td>1</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>Etcd</td>
<td>7</td>
<td>12</td>
<td>-</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Go-Ethereum</td>
<td>11</td>
<td>43</td>
<td>6</td>
<td>2</td>
<td>62</td>
</tr>
<tr>
<td>TiDB</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>gRPC</td>
<td>15</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>22</td>
</tr>
</tbody>
</table>
184 previously unknown bugs
  – 170 blocking bugs

<table>
<thead>
<tr>
<th>App</th>
<th>$\text{chan}_b$</th>
<th>$\text{select}_b$</th>
<th>$\text{range}_b$</th>
<th>NBK</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes</td>
<td>28</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>Docker</td>
<td>17</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>19</td>
</tr>
<tr>
<td>Prometheus</td>
<td>14</td>
<td>-</td>
<td>1</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>Etdc</td>
<td>7</td>
<td>12</td>
<td>-</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Go-Ethereum</td>
<td>11</td>
<td>43</td>
<td>6</td>
<td>2</td>
<td>62</td>
</tr>
<tr>
<td>TiDB</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>gRPC</td>
<td>15</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>22</td>
</tr>
</tbody>
</table>

wait for multiple channel operation
Effectiveness of Bug Detection

- 184 previously unknown bugs
  - 170 blocking bugs

<table>
<thead>
<tr>
<th>App</th>
<th>chan&lt;sub&gt;b&lt;/sub&gt;</th>
<th>select&lt;sub&gt;b&lt;/sub&gt;</th>
<th>range&lt;sub&gt;b&lt;/sub&gt;</th>
<th>NBK</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes</td>
<td>28</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>Docker</td>
<td>17</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>19</td>
</tr>
<tr>
<td>Prometheus</td>
<td>14</td>
<td>-</td>
<td>1</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>Etcd</td>
<td>7</td>
<td>12</td>
<td>-</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Go-Etherum</td>
<td>11</td>
<td>43</td>
<td>6</td>
<td>2</td>
<td>62</td>
</tr>
<tr>
<td>TiDB</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>gRPC</td>
<td>15</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>22</td>
</tr>
</tbody>
</table>

drain a channel in a loop
Effectiveness of Bug Detection

- 184 previously unknown bugs
  - 170 blocking bugs
  - 14 non-blocking bugs
- 12 FPs due to imprecise static analysis

<table>
<thead>
<tr>
<th>App</th>
<th>$\text{chan}_b$</th>
<th>$\text{select}_b$</th>
<th>$\text{range}_b$</th>
<th>NBK</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes</td>
<td>28</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>Docker</td>
<td>17</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>19</td>
</tr>
<tr>
<td>Prometheus</td>
<td>14</td>
<td>-</td>
<td>1</td>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>Etcd</td>
<td>7</td>
<td>12</td>
<td>-</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Go-Ethereum</td>
<td>11</td>
<td>43</td>
<td>6</td>
<td>2</td>
<td>62</td>
</tr>
<tr>
<td>TiDB</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>gRPC</td>
<td>15</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>22</td>
</tr>
</tbody>
</table>
• Compare GFuzz with static detector GCatch
  – Run GFuzz on each application for three hours
  – Apply GCatch on all packages that can be compiled

\[
\begin{array}{|c|c|c|}
\hline
\text{App} & \text{GFuzz}_3 & \text{GCatch} \\
\hline
\text{Kubernetes} & 18 & 3 \\
\text{Docker} & 5 & 4 \\
\text{Prometheus} & 8 & - \\
\text{Etcd} & 7 & 5 \\
\text{Go-Ethereum} & 40 & 5 \\
\text{TiDB} & - & - \\
\text{gRPC} & 7 & 8 \\
\hline
\end{array}
\]

\[
85 > 25
\]
GFuzz detects 5 bugs captured by GCatch

GFuzz misses 20 bugs for four reasons:

- 2 bugs are missed due to the implementation
- 5 bugs detected by GFuzz
- 8 bugs do not have good unit tests
- 6 bugs need a longer fuzzing time
- 4 bugs cannot be helped by message reordering
The average overhead of GFuzz is **3.0X**

The average overhead of the sanitizer is **32.3%**

<table>
<thead>
<tr>
<th>App</th>
<th>GFuzz&lt;sub&gt;o&lt;/sub&gt;</th>
<th>Sanitizer&lt;sub&gt;o&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kubernetes</td>
<td>8.7X</td>
<td>36.65%</td>
</tr>
<tr>
<td>Docker</td>
<td>22.7X</td>
<td>44.53%</td>
</tr>
<tr>
<td>Prometheus</td>
<td>3.0X</td>
<td>18.08%</td>
</tr>
<tr>
<td>Etcld</td>
<td>0.9X</td>
<td>14.43%</td>
</tr>
<tr>
<td>Go-Ethereum</td>
<td>20.1X</td>
<td>75.18%</td>
</tr>
<tr>
<td>TiDB</td>
<td>1.6X</td>
<td>17.65%</td>
</tr>
<tr>
<td>gRPC</td>
<td>8.5X</td>
<td>20.00%</td>
</tr>
</tbody>
</table>

Comparable with ASAN and TSAN
Contribution of Gfuzz’s Components

- Apply GFuzz on gRPC using 4 different settings:
  - Enable all components
  - Disable one of the three components
Contribution of Gfuzz’s Components

- Apply GFuzz on gRPC using 4 different settings:
  - Enable all components
  - Disable one of the three components
Contribution of Gfuzz’s Components

- Apply GFuzz on gRPC using 4 different settings:
  - Enable all components
  - Disable one of the three components

![Graph showing contributions](image)
Contribution of Gfuzz’s Components

• Apply GFuzz on gRPC using 4 different settings:
  – Enable all components
  – Disable one of the three components

![Graph showing the contribution of Gfuzz’s Components.](chart.png)
Conclusion

• GFuzz: an effective dynamic bug detector
  – Change message order to explore program states
  – Use feedback to prioritize suspicious orders
  – Propose a sanitizer to capture blocking bugs

*Detected 184 previously unknown bugs in real Go apps*

• Future work
  – Integrate other mutation mechanisms
  – Identify more concurrent messages
Thanks a lot!

fuzzing message orders for concurrency bugs

We just built a tool, named GFuzz, to fuzz (randomly mutate) message orders to expose concurrency bugs in Go programs. Our paper is published in this year’s ASPLOS. In total, we find 184 previously unknown concurrency bugs in famous open-source Go software. 67 of the detected bugs have already been fixed based on our reporting.

Our paper can be found here: [https://songlh.github.io/paper/gfuzz.pdf](https://songlh.github.io/paper/gfuzz.pdf)

We also release our tool on GitHub: [https://github.com/system-pclub/GFuzz](https://github.com/system-pclub/GFuzz)

Feel free to try our tools and comments are welcome.

Contact: Shihao Xia (szx5097@psu.edu)
Questions?

- How to identify concurrent messages?
- How to identify suspicious message orders?
- How to capture triggered channel-related bugs?