Fine-Grained Fault Tolerance using Device Checkpoints

Asim Kadav
with Matthew Renzelmann and Michael M. Swift
University of Wisconsin-Madison
The (old) elephant in the room

3rd party developers

+ device drivers

OS kernel
The (old) elephant in the room

3rd party developers + device drivers

OS kernel
The (old) elephant in the room

3rd party developers + device drivers → OS kernel

Recipe for disaster

Thursday, March 7, 13
### Improvement

<table>
<thead>
<tr>
<th>System</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shadow driver migration</td>
<td>1</td>
</tr>
<tr>
<td>RevNIC</td>
<td>1</td>
</tr>
</tbody>
</table>

### Reliability

<table>
<thead>
<tr>
<th>System</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nooks</td>
<td>6</td>
</tr>
<tr>
<td>XFI</td>
<td>2</td>
</tr>
<tr>
<td>CuriOS</td>
<td>2</td>
</tr>
</tbody>
</table>

### Type Safety

<table>
<thead>
<tr>
<th>System</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SafeDrive</td>
<td>6</td>
</tr>
<tr>
<td>Singularity</td>
<td>1</td>
</tr>
</tbody>
</table>

### Specification

<table>
<thead>
<tr>
<th>System</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nexus</td>
<td>2</td>
</tr>
<tr>
<td>Termite</td>
<td>2</td>
</tr>
</tbody>
</table>

### Static analysis tools

<table>
<thead>
<tr>
<th>Static analysis tools</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows SDV</td>
<td>All</td>
</tr>
<tr>
<td>Coverity</td>
<td>All</td>
</tr>
<tr>
<td>Cocinelle</td>
<td>All</td>
</tr>
</tbody>
</table>
### Improvement System Validation

<table>
<thead>
<tr>
<th>Improvement</th>
<th>System</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Drivers</td>
</tr>
<tr>
<td>New functionality</td>
<td>Shadow driver migration [OSR09]</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>RevNIC [Eurosys 10]</td>
<td>1</td>
</tr>
<tr>
<td>Reliability</td>
<td>Nooks [SOSP 03]</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>XFI [OSDI 06]</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>CuriOS [OSDI 08]</td>
<td>2</td>
</tr>
<tr>
<td>Type Safety</td>
<td>SafeDrive [OSDI 06]</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Singularity [Eurosys 06]</td>
<td>1</td>
</tr>
<tr>
<td>Specification</td>
<td>Nexus [OSDI 08]</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Termite [SOSP 09]</td>
<td>2</td>
</tr>
</tbody>
</table>

Past work mostly looks at detection and isolation of security threats. However, the limited adoption of research solutions is largely due to the complexity of large kernel subsystems and the validity of few device types.

Large kernel subsystems and validity of few device types result in limited adoption of research solutions.
Past work mostly looks at detection and isolation.

<table>
<thead>
<tr>
<th>Improvement</th>
<th>System</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Drivers</td>
</tr>
<tr>
<td>New functionality</td>
<td>Shadow driver migration [OSR09]</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>RevNIC [Eurosys 10]</td>
<td>1</td>
</tr>
<tr>
<td>Reliability</td>
<td>Nooks [SOSP 03]</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>XFI [OSDI 06]</td>
<td>2</td>
</tr>
<tr>
<td>Specification</td>
<td>Nexus [OSDI 08]</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Termite [SOSP 09]</td>
<td>2</td>
</tr>
<tr>
<td>Static analysis tools</td>
<td>Windows SDV [Eurosys 06]</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td>Coverity [CACM 10]</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td>Cocinelle [Eurosys 08]</td>
<td>All</td>
</tr>
</tbody>
</table>

Limited kernel changes + Applicable to lots of drivers => Real Impact
### Improvement System Validation

<table>
<thead>
<tr>
<th>Improvement</th>
<th>System</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Drivers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Classes</td>
</tr>
<tr>
<td>New functionality</td>
<td>Shadow driver migration [OSR09]</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>RevNIC [Eurosys 10]</td>
<td>1</td>
</tr>
<tr>
<td>Reliability</td>
<td>Nooks [SOSP 03]</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>XFI [OSDI 06]</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Singularity [Eurosys 06]</td>
<td>1</td>
</tr>
<tr>
<td>Specification</td>
<td>Nexus [OSDI 08]</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Termite [SOSP 09]</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Static analysis tools</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td>Windows SDV [Eurosys 06]</td>
<td>All</td>
</tr>
</tbody>
</table>

**Limited kernel changes + Applicable to lots of drivers => Real Impact**

**Goal: Improve recovery with complete solutions that can be applied to many drivers**

Thursday, March 7, 13
State of the art in recovery: Shadow drivers

- Carburizer calls generic recovery service if check fails
- Low cost transparent recovery
  - Based on shadow drivers
  - Records state of driver at all times
  - Transparently restarts and replays recorded state on failure

Swift [OSDI ’04]
Recovery Performance: Device initialization is slow

★ Multi-second device probe
  ★ Identify device
  ★ Cold boot device
  ★ Setup device/driver structures
  ★ Configuration/Self-test
Recovery Performance: Device initialization is slow

★ Multi-second device probe
  ★ Identify device
  ★ Cold boot device
  ★ Setup device/driver structures
  ★ Configuration/Self-test

★ What does it hurt?
  ★ Fault tolerance: Driver recovery
  ★ Virtualization: Live migration, cloning, consolidation
  ★ OS functions: Boot, upgrade, NVM checkpoints

Thursday, March 7, 13
Driver Code Characteristics
<table>
<thead>
<tr>
<th>Driver Code Characteristics</th>
<th>init</th>
<th>cleanup</th>
<th>ioctl</th>
<th>config</th>
<th>power</th>
<th>error</th>
<th>proc</th>
<th>core</th>
<th>intr</th>
</tr>
</thead>
<tbody>
<tr>
<td>acpi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bluetooth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdrom</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>crypto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>edac</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>firewire</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gpio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gpu</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hwmon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isdn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>leds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>media</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>message</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>misc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parport</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>platform</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pnp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>serial</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sound</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>video</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>watchdog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ata</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>block</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>md</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mtd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scsi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>atm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>infiniband</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>net</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uwb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*“Understanding Modern Device Drivers”*  
**ASPLOS 2012**
“Understanding Modern Device Drivers”  

- Initialization/cleanup – 36%
- Core I/O & interrupts – 23%
- Device configuration – 15%
- Power management – 7.4%
- Device ioctl – 6.2%
Initialization code dominates driver LOC and adds to complexity

★ Initialization/cleanup – 36%
★ Core I/O & interrupts – 23%
★ Device configuration – 15%
★ Power management – 7.4%
★ Device ioctl – 6.2%
Recovery works by interposing class defined entry points

Class definition includes:

- Callbacks registered with the bus, device and kernel subsystem
Recovery works by interposing class defined entry points

Class definition includes:

- Callbacks registered with the bus, device and kernel subsystem

How many drivers follow class behavior?
 Restart/replay doesn’t work with all drivers

★ Non-class behavior stems from:
- Load time parameters, procfs and sysfs interactions, unique ioctl

```c
... qlcnic_sysfs_write_esw_config (...) {
    ...
    switch (esw_cfg[i].op_mode) {
      case QLCNIC_PORT_DEFAULTS:
        qlcnic_set_eswitch(...,...,&esw_cfg[i]);
        ...
      case QLCNIC_ADD_VLAN:
        qlcnic_set_vlan_config(...,...,&esw_cfg[i]);
        ...
      case QLCNIC_DEL_VLAN:
        esw_cfg[i].vlan_id = 0;
        qlcnic_set_vlan_config(...,...,&esw_cfg[i]);
        ...
    }
...```

Drivers/net/qlcnic/qlcnic_main.c: Qlogic driver(network class)

★ “Understanding Modern Device Drivers” *ASPLOS 2012*
Restart/replay doesn’t work with all drivers

- Non-class behavior stems from:
  - Load time parameters, procfs and sysfs interactions, unique ioctls

Results as measured by our analyses:
- 36% of drivers use load time parameters
- 16% of drivers use proc /sysfs support

Overall, 44% of drivers do not conform to class behavior and recovery will not work correctly for these drivers
Limitations of restart/replay recovery

- **Device save/restore limited to restart/replay**
  - **Slow**: Device initialization is complex (multiple seconds)
  - **Incomplete**: Unique device semantics not captured
  - **Hard**: Need to be written for every class of drivers
  - **Large changes**: Introduces new large kernel subsystems
Limitations of restart/replay recovery

- Device save/restore limited to restart/replay
  - Slow: Device initialization is complex (multiple seconds)
  - Incomplete: Unique device semantics not captured
  - Hard: Need to be written for every class of drivers
  - Large changes: Introduces new large kernel subsystems

Checkpoint/restore of device and driver state removes the need to reboot device and replay state
Fine-Grained Fault Tolerance (FGFT)

Goal: Fault isolation and recovery system based on “pay as you go” failure model

**Fine-Grained Isolation**
- Ability to run select entry points as transactions

**Checkpoint based recovery**
- Provides fast and correct recovery semantics
- Requires incremental changes to drivers and has low overhead
Outline

- Introduction
- Fine-grained isolation
- Checkpoint based recovery
- Conclusion
FGFT overview

Driver with checkpoint support

Static modifications
FGFT overview

- **Source transformation** (adds driver transactions)
- **Driver with checkpoint support**
- **User supplied annotations**

Static modifications
FGFT overview

Static modifications

Driver with checkpoint support

Source transformation (adds driver transactions)

User supplied annotations

Main driver module

SFI driver module

SFI = software fault isolated
FGFT overview

Source transformation (adds driver transactions)

Driver with checkpoint support

User supplied annotations

Main driver module

SFI driver module

SFI = software fault isolated

Static modifications

Run-time support

If (c==0) {
  .
  print ("Driver init");
  .
}
FGFT overview

Source transformation (adds driver transactions)

Driver with checkpoint support

User supplied annotations

Main driver module

SFI driver module

Object tracking

Marshaling/Demarshaling

Kernel undo log

Communication and recovery support

SFI = software fault isolated

1200 LOC

Static modifications

Run-time support

If (c==0) {
  print ("Driver init");
}
Fault model in FGFT

- Can be applied to untested code, statically and dynamically detected suspicious entry points

- Detect and recover from:
  - Memory errors like NULL pointer accesses
  - Structural errors like malformed structures
  - Processor exceptions like divide by zero, stack corruption
Fault model in FGFT

- Provide fault tolerance to specific driver entry points
- Can be applied to untested code, statically and dynamically detected suspicious entry points
- Detect and recover from:
  - Memory errors like NULL pointer accesses
  - Structural errors like malformed structures
  - Processor exceptions like divide by zero, stack corruption
Transactional support through code generation

- Generate code to run driver invocations on a separate stack with a copy of parameters
- Reduce copy overhead by copying only referenced fields in driver and kernel structures to a range table
- Instrument all memory references in SFI module to compare accesses against copied fields in range table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>
Resource access during isolated execution

★ Device registers and I/O memory
  ★ Grant drivers full access to devices
  ★ Restore device checkpoint in case of failure

★ Locks: Spinlocks and semaphores
  ★ Grants read access to locks
  ★ Maintain kernel log of locks acquired
  ★ Release locks at the end of entry point/failures

★ Kernel resources like memory
  ★ All allocations generate range table entry
  ★ Maintain kernel log of all acquired resources
  ★ Free resources on failures

Range Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>
Checkpointing drivers is hard

★ Existing mechanisms limited to capturing memory state
Checkpointing drivers is hard

Existing mechanisms limited to capturing memory state
Checkpointing drivers is hard

★ Existing mechanisms limited to capturing memory state

★ Device state is not captured
  ★ Device configuration space
Checkpointing drivers is hard

- Existing mechanisms limited to capturing memory state

- Device state is not captured
  - Device configuration space
  - Internal device registers and counters
Checkpointing drivers is hard

- Existing mechanisms limited to capturing memory state

- Device state is not captured
  - Device configuration space
  - Internal device registers and counters
  - Memory buffer addresses used for DMA
Checkpointing drivers is hard

- Existing mechanisms limited to capturing memory state

- Device state is not captured
  - Device configuration space
  - Internal device registers and counters
  - Memory buffer addresses used for DMA
  - Unique for every class, bus and vendor
Device checkpoint/restore from PM code

**Suspend**
- Save config state
- Save register state
- Disable device
- Copy-out s/w state
- Suspend device

**Resume**
- Restore config state
- Restore register state
- Restore s/w state & reset
- Re-attach/Enable device
- Device Ready
Device checkpoint/restore from PM code

**Suspend**
- Save config state
- Save register state
- Copy-out s/w state
- Suspend device

**Resume**
- Restore config state
- Restore register state
- Restore s/w state & reset
- Re-attach/Enable device
- Device Ready
Device checkpoint/restore from PM code

Suspend

- Save config state
- Save register state
- Copy-out s/w state

Resume

- Restore config state
- Restore register state
- Restore s/w state & reset
- Re-attach/Enable device
- Device Ready

Thursday, March 7, 13
Device checkpoint/restore from PM code

**Suspend**
- Save config state
- Save register state
- Copy-out s/w state

**Resume**
- Restore config state
- Restore register state
- Restore s/w state & reset
- Re-attach/Enable device
- Device Ready
Device checkpoint/restore from PM code

**Checkpoint**

- Save config state
- Save register state
- Copy-out s/w state

**Resume**

- Restore config state
- Restore register state
- Restore s/w state & reset
- Re-attach/Enable device
- Device Ready
Device checkpoint/restore from PM code

Checkpoint

- Save config state
- Save register state
- Copy-out s/w state

Resume

- Restore config state
- Restore register state
- Restore s/w state & reset
- Re-attach/Enable device

Thursday, March 7, 13
Device checkpoint/restore from PM code

**Checkpoint**

- Save config state
- Save register state
- Copy-out s/w state

**Resume**

- Restore config state
- Restore register state
- Restore s/w state & reset
Device checkpoint/restore from PM code

**Checkpoint**
- Save config state
- Save register state
- Copy-out s/w state

**Restore**
- Restore config state
- Restore register state
- Restore s/w state & reset
Device checkpoint/restore from PM code

Checkpoint
- Save config state
- Save register state
- Copy-out s/w state

Restore
- Restore config state
- Restore register state
- Restore s/w state & reset

Suspend/resume code provides device checkpoint functionality
Intuition with power management
Intuition with power management

- **Intuition:** Power management code captures device specific state for every driver
  - **Our study:** Present in 76% of all common classes
Intuition with power management

★ Intuition: Power management code captures device specific state for every driver
★ Our study: Present in 76% of all common classes
Intuition with power management

- Intuition: Power management code captures device specific state for every driver
- **Our study:** Present in 76% of all common classes
Intuition with power management

- Intuition: Power management code captures device specific state for every driver
- Our study: Present in 76% of all common classes
Intuition: Power management code captures device specific state for every driver

Our study: Present in 76% of all common classes
Intuition with power management

- Intuition: Power management code captures device specific state for every driver
  - Our study: Present in 76% of all common classes

- Refactor power management code for device checkpoints
  - Correct: Developer captures unique device semantics
  - Fast: Avoids probe and latency critical for applications
Synergy of isolation and recovery

★ Goal: Improve driver recovery with minor changes to drivers
★ Solution: Run drivers as transactions using device checkpoints
Synergy of isolation and recovery

★ Goal: Improve driver recovery with minor changes to drivers
★ Solution: Run drivers as transactions using device checkpoints

Device state

★ Developers export checkpoint/restore in drivers
Synergy of isolation and recovery

★ Goal: Improve driver recovery with minor changes to drivers
★ Solution: Run drivers as transactions using device checkpoints

Device state

★ Developers export checkpoint/restore in drivers

Driver state

★ Run drivers invocations as memory transactions
★ Use source transformation to copy parameters and run on separate stack

C R

network driver

SFI network driver

Thursday, March 7, 13
**Synergy of isolation and recovery**

* Goal: Improve driver recovery with minor changes to drivers
* Solution: Run drivers as **transactions** using device checkpoints

**Device state**
- Developers export checkpoint/restore in drivers

**Driver state**
- Run drivers invocations as memory transactions
- Use source transformation to copy parameters and run on separate stack

**Execution model**
- Checkpoint device
- Execute driver code as memory transactions
- On failure, rollback and restore device
- Re-use existing device locks in the driver
Example transactional execution

- network driver
- SFI network driver

get ringparam
probe
xmit
get config
Example transactional execution

get ringparam
probe
xmit
get config

netdev

network driver

SFI
network
driver
Example transactional execution

- C
- network driver
- SFI network driver

Transactions:
- get ringparam
- probe
- xmit
- get config
Example transactional execution

```
netdev->priv->tx_ring
netdev->priv->rx_ring
```

```
netdev
network
driver

SFI
network
driver
```

**get ringparam**

**probe**

**xmit**

**get config**
Example transactional execution

- **netdev->priv->rx_ring**
- **netdev->priv->tx_ring**

**Range Table**

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

**Network Driver**

- get ringparam
- probe
- xmit
- get config

**SFI Network Driver**

- netdev
- netdev->rx_ring
- netdev->tx_ring

*Thursday, March 7, 13*
Example transactional execution

**C**

```
netdev->priv->rx_ring
netdev->priv->tx_ring
```

**Range Table**

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

**netdev**

- get ringparam
- probe
- xmit
- get config

**network driver**

**SFI network driver**
Example transactional execution

![Diagram showing network driver and SFI network driver with address access rights table]

- **Address**
  - 0xffffa000: Read
  - 0xffffa008: Write
  - 0xffffa00a: Read

**Range Table**

---

**netdev->priv->rx_ring**

**netdev->priv->tx_ring**
Example transactional execution

get ringparam
probe
xmit
get config

netdev

network driver

netdev->priv->rx_ring
netdev->priv->tx_ring

result

netdev->priv->rx_ring
netdev->priv->tx_ring

SFI network driver

Range Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

Kernel Log
alloc

Thursday, March 7, 13
Example failed transaction

- get ringparam
- probe
- xmit
- get config

SFI network driver

network driver

Thursday, March 7, 13
Example failed transaction
Example failed transaction

get ringparam
probe
xmit
get config

netdev
network driver

SFI network driver

Thursday, March 7, 13
Example failed transaction

```c
netdev->priv->rx_ring
netdev->priv->tx_ring
```

**netdev**

**network driver**

**SFI network driver**
Example failed transaction

netdev->priv->rx_ring
netdev->priv->tx_ring

Range Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

Thursday, March 7, 13
Example failed transaction

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

Range Table

C

netdev

netdev->priv->tx_ring

netdev->priv->rx_ring

SFI

network

driver

network

driver

get config

get ringparam

probe

xmit
Example failed transaction

```
netdev->priv->rx_ring
netdev->priv->tx_ring
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

C

探

netdev

network
driver

get config

get ringparam

xmit

probe

SFI

network
driver

Range Table
Example failed transaction

- `netdev->priv->rx_ring`
- `netdev->priv->tx_ring`

Range Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

Kernel Log alloc
Example failed transaction

```
netdev->priv->rx_ring
netdev->priv->tx_ring
```

Range Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

Kernel Log alloc

netdev

network driver

get ringparam

probe

xmit

get config

SFI

network driver

Thursday, March 7, 13
Example failed transaction

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

Range Table

get ringparam
probe
xmit
get config

netdev->priv->rx_ring
netdev->priv->tx_ring

SFI network driver

network driver

Kernel Log alloc

Thursday, March 7, 13
Example failed transaction

Range Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

get ringparam
probe
xmit
get config

netdev

network driver

netdev->priv->rx_ring
netdev->priv->tx_ring

SFI network driver
Example failed transaction

```c
netdev->priv->rx_ring
netdev->priv->tx_ring
```

Range Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

get ringparam

probe

xmit

get config

c

SFI

network

driver

err

netdev

network
driver

Range Table

Thursday, March 7, 13
Example failed transaction

C

netdev->priv->rx_ring
netdev->priv->tx_ring

Range Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

network driver

SFI

network driver

get config
get ringparam
probe
xmit
Example failed transaction

FGFT provides transactional execution of driver entry points

Range Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffa000</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffa008</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffa00a</td>
<td>Read</td>
</tr>
</tbody>
</table>

netdev->priv->rx_ring
netdev->priv->tx_ring

C

netdev
network driver

R

get ringparam
get config
probe
xmit

err
# Recovery speedup

<table>
<thead>
<tr>
<th>Driver</th>
<th>Class</th>
<th>Bus</th>
<th>Restart recovery</th>
<th>FGFT recovery</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>8139too</td>
<td>net</td>
<td>PCI</td>
<td>0.31s</td>
<td>70μs</td>
<td>4400</td>
</tr>
<tr>
<td>e1000</td>
<td>net</td>
<td>PCI</td>
<td>1.80s</td>
<td>295ms</td>
<td>6</td>
</tr>
<tr>
<td>r8169</td>
<td>net</td>
<td>PCI</td>
<td>0.12s</td>
<td>40μs</td>
<td>3000</td>
</tr>
<tr>
<td>pegasus</td>
<td>net</td>
<td>USB</td>
<td>0.15s</td>
<td>5ms</td>
<td>30</td>
</tr>
<tr>
<td>ens1371</td>
<td>sound</td>
<td>PCI</td>
<td>1.03s</td>
<td>115ms</td>
<td>9</td>
</tr>
<tr>
<td>psmouse</td>
<td>input</td>
<td>serio</td>
<td>0.68s</td>
<td>410ms</td>
<td>1.65</td>
</tr>
</tbody>
</table>

FGFT provides significant speedup in driver recovery
### Static and dynamic fault injection

<table>
<thead>
<tr>
<th>Driver</th>
<th>Injected Faults</th>
<th>Benign Faults</th>
<th>Native Crashes</th>
<th>FGFT Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8139too</td>
<td>43</td>
<td>0</td>
<td>43</td>
<td>NONE</td>
</tr>
<tr>
<td>e1000</td>
<td>47</td>
<td>0</td>
<td>47</td>
<td>NONE</td>
</tr>
<tr>
<td>r8169</td>
<td>36</td>
<td>0</td>
<td>36</td>
<td>NONE</td>
</tr>
<tr>
<td>pegasus</td>
<td>34</td>
<td>1</td>
<td>33</td>
<td>NONE</td>
</tr>
<tr>
<td>ens1371</td>
<td>22</td>
<td>1</td>
<td>21</td>
<td>NONE</td>
</tr>
<tr>
<td>psmouse</td>
<td>46</td>
<td>0</td>
<td>46</td>
<td>NONE</td>
</tr>
<tr>
<td>TOTAL</td>
<td>258</td>
<td>2</td>
<td>256</td>
<td>NONE</td>
</tr>
</tbody>
</table>

**FGFT survives multiple static and dynamic faults**
FGFT requires limited programmer effort and needs only 38 lines of new kernel code.
Throughput with isolation and recovery

- Native
- FGFT-off-I/O
- FGFT-I/O-1/2
- FGFT-I/O-all

*netperf on Intel quad-core machines*
Throughput with isolation and recovery

- Native
- FGFT-off-I/O
- FGFT-I/O-1/2
- FGFT-I/O-all

netperf on Intel quad-core machines
Throughput with isolation and recovery

Throughput %age (Baseline 844 Mbps)

- Native
- FGFT-off-I/O
- FGFT-I/O-1/2
- FGFT-I/O-all

**CPU**: 2.4%

*netperf* on Intel quad-core machines
Throughput with isolation and recovery

CPU: 2.4%  2.4%

Throughput %age (Baseline 844 Mbps)

Native
FGFT-off-I/O
FGFT-I/O-1/2
FGFT-I/O-all

e1000 Network Card

netperf on Intel quad-core machines
Throughput with isolation and recovery

Throughput %age (Baseline 844 Mbps)

CPU: 2.4% 2.4% 2.9%

e1000 Network Card

netperf on Intel quad-core machines

Native
FGFT-off-I/O
FGFT-I/O-1/2
FGFT-I/O-all

Thursday, March 7, 13
Throughput with isolation and recovery

Throughput %age (Baseline 844 Mbps)

CPU: 2.4%  2.4%  2.9%  3.4%

Native  FGFT-off-I/O  FGFT-I/O-1/2  FGFT-I/O-all

netperf on Intel quad-core machines
FGFT can isolate and recover high bandwidth devices at low overhead without adding kernel subsystems

netperf on Intel quad-core machines
Summary

- Fine-Grained Fault tolerance based on a pay-as-you go model
  - Provides fault tolerance at incremental performance costs and programmer efforts

- Introduces fast checkpointing for drivers
  - Device checkpoints average ~20micros
  - Reduces recovery time significantly
  - Should be explored in other domains apart from fault tolerance like fast reboot, upgrade etc.
Questions

Asim Kadav
★ http://cs.wisc.edu/~kadav
★ kadav@cs.wisc.edu