Understanding and Improving Device Access Complexity

Asim Kadav
(with Prof. Michael M. Swift)
University of Wisconsin-Madison
Devices enrich computers

- Keyboard
- Sound
- Printer
- Network
- Storage
Devices enrich computers

- Keyboard
- Sound
- Printer
- Network
- Storage

- Keyboard
- Flash storage
- Graphics
- WIFI
- Headphones
- SD card
- Camera
- Accelerometers
- GPS
- Touch display
- NFC
Huge growth in number of devices

New I/O devices: accelerometers, GPUs, GPS, touch
Huge growth in number of devices

New I/O devices: accelerometers, GPUs, GPS, touch

Many buses: USB, PCI-e, thunderbolt
Huge growth in number of devices

New I/O devices: accelerometers, GPUs, GPS, touch

Many buses: USB, PCI-e, thunderbolt

Heterogeneous O/S support: 10G ethernet vs card readers
Device drivers: OS interface to devices

- applications
- OS
- device drivers
- buses
- devices
Device drivers: OS interface to devices

Exposure device abstractions and hide device complexity
Device drivers: OS interface to devices

Expose kernel abstractions and hide OS complexity

Expose device abstractions and hide device complexity

Allow diverse set of applications and OS services to access diverse set of devices
Complexity hurts efficient device access

Evolution of devices

Efficient device support in OS
Complexity hurts efficient device access

Evolution of devices

- Simplicity
- Reliability
- Low latency
- Cost effective

Efficient device support in OS
Complexity hurts efficient device access

- Simplicity
- Low latency
- Reliability
- Cost effective

Growth in number and diversity
Run in challenging environments
Hardware failures (like CMOS issues)
Complex firmware and configuration modes

Evolution of devices
Efficient device support in OS
Complexity hurts efficient device access

Tools and mechanisms to address increasing device complexity

- Growth in number and diversity
- Run in challenging environments
- Hardware failures (like CMOS issues)
- Complex firmware and configuration modes

Efficient device support in OS

- Simplicity
- Reliability
- Low latency
- Cost effective

Evolution of devices
Complexity hurts understanding of drivers
Complexity hurts understanding of drivers

- device drivers
- buses
- devices

Contribute 60% of Linux kernel code
and more than 90% in Windows
Complexity hurts understanding of drivers

Lines of code in Linux 3.8

memory mgmt: 60,000
kernel: 66,000
file systems: 760,000
drivers: 6,700,000
Complexity hurts understanding of drivers

Lines of code in Linux 3.8

- memory mgmt: 60,000
- kernel: 66,000
- file systems: 760,000
- drivers: 6,700,000

Understand and improve this rapidly growing body of code
Last decade: Reliability of the driver-kernel interface

3rd party developers + device drivers = OS kernel
Last decade: Reliability of the driver-kernel interface

3rd party developers + device drivers → OS kernel
Last decade: Reliability of the driver-kernel interface

3rd party developers + device drivers = OS kernel

Recipe for disaster
Re-use lessons from existing driver research

<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>
<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>
## Re-use lessons from existing driver research

<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</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>[OSR09]</td>
<td></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>

Large kernel subsystems and validity of few device types result in limited adoption of research solutions.
Re-use lessons from existing driver research

<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>
<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>
Re-use lessons from existing driver research

<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</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>[OSR09]</td>
<td></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

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

★ Make device access efficient and reliable in the face of rising hardware and software complexity
Goal

★ Make device access efficient and reliable in the face of rising hardware and software complexity

Increasing hardware complexity

Reliability against hardware failures
Goal

★ Make device access efficient and reliable in the face of rising hardware and software complexity

- Increasing hardware complexity
  - Reliability against hardware failures
- Increasing hardware complexity
  - Low latency device availability
Goal

- Make device access efficient and reliable in the face of rising hardware and software complexity

- Increasing hardware complexity
  - Reliability against hardware failures

- Increasing hardware complexity
  - Low latency device availability

- Increasing software complexity
  - Better understanding of driver code
My approach
My approach

Take a narrow view and solve specific problems in all drivers

Tolerate device failures
My approach

- Take a narrow view and solve specific problems in all drivers
- Tolerate device failures

- Take a broad approach and have a holistic view of all drivers
- Understand drivers and potential opportunities
My approach

- Take a narrow view and solve specific problems in all drivers
- Tolerate device failures

- Take a broad approach and have a holistic view of all drivers
- Understand drivers and potential opportunities

- Take a known approach and apply to all drivers
- Transactional approach for low latency recovery
My approach

Take a narrow view and solve specific problems in all drivers

Tolerate device failures

Take a broad approach and have a holistic view of all drivers

Understand drivers and potential opportunities

Take a known approach and apply to all drivers

Transactional approach for low latency recovery

Minimize kernel changes and apply to all drivers
Contributions/Outline

**SOSP ’09**

First research consideration of hardware failures in drivers

Tolerate device failures

**ASPLOS ’12**

Largest study of drivers to understand their behavior and verify research assumptions

Understand drivers and potential opportunities

**ASPLOS ’13**

Introduce checkpoint/restore in drivers for low latency fault tolerance

Transactional approach for low latency recovery
What happens when devices misbehave?
What happens when devices misbehave?

★ Drivers make it better
What happens when devices misbehave?

- Drivers make it better
- Drivers make it worse
What happens when devices misbehave?

★ Drivers make it better
★ Drivers make it worse

Early example: Apollo 11 1969

★ Hardware design bug almost aborted the landing
★ Assumptions about antenna in driver led to extra CPU
★ Scientists on-board had to manually prioritize critical tasks
Current state of OS-hardware interaction

2013
Current state of OS-hardware interaction

2013

- Many device drivers often assume device perfection
  - Common Linux network driver: 3c59x.c
Current state of OS-hardware interaction

2013

★ Many device drivers often assume device perfection
- Common Linux network driver: 3c59x.c

while (ioread16(ioaddr + Wn7_MasterStatus)) & 0x8000);
Current state of OS-hardware interaction

2013

★ Many device drivers often assume device perfection
  - Common Linux network driver: 3c59x.c

while (ioread16(ioaddr + Wn7_MasterStatus)) & 0x8000);
Current state of OS-hardware interaction

2013

★ Many device drivers often assume device perfection
  - Common Linux network driver: 3c59x.c

while (ioread16(ioaddr + Wn7_MasterStatus))
  & 0x8000);

Hardware dependence bug: Device malfunction can crash the system
Sources of hardware misbehavior

- Sources of hardware misbehavior

Diagram:
- Driver
- Bus
- Cache
- Firmware
- Electrical
- Mechanical
- Device
Sources of hardware misbehavior

- Firmware/Design bugs
Sources of hardware misbehavior

- Firmware/Design bugs
- Device wear-out, insufficient burn-in
- Bridging faults
Sources of hardware misbehavior

- Firmware/Design bugs
- Device wear-out, insufficient burn-in
- Bridging faults
- Electromagnetic radiation
Sources of hardware misbehavior

★ Firmware/Design bugs
★ Device wear-out, insufficient burn-in
★ Bridging faults
★ Electromagnetic radiation

★ Results of misbehavior
★ Corrupted/stuck-at inputs
★ Timing errors/incorrect memory access
★ Interrupt storms/missing interrupts
An evidence: Windows Server
An evidence:

Transient hardware failures caused 8% of all crashes and 9% of all unplanned reboots [1]

Transient hardware failures caused 8% of all crashes and 9% of all unplanned reboots [1]

★ Systems work fine after reboots
★ Vendors report returned device was faultless

Transient hardware failures caused 8% of all crashes and 9% of all unplanned reboots [1]

- Systems work fine after reboots
- Vendors report returned device was faultless

Existing solution is **hand-coded** hardened drivers

- Crashes reduce from 8% to 3%

How do hardware dependence bugs manifest?
How do hardware dependence bugs manifest?

Drivers use device data in critical control and data paths.

\[\text{printk("\%s", msg[inb(regA)]);}\]
How do hardware dependence bugs manifest?

1. Drivers use device data in critical control and data paths
   
   ```c
   printk("%s", msg[inb(regA)]);
   ```

2. Drivers do not report device malfunction to system log
   
   ```c
   if (inb(regA)!= 5) {
       return; //do nothing
   }
   ```
How do hardware dependence bugs manifest?

1. Drivers use device data in critical control and data paths
   ```c
   printk("%s",msg[inb(regA)]);
   ```

2. Drivers do not report device malfunction to system log
   ```c
   if (inb(regA)!=5) {
     return; //do nothing
   }
   ```

3. Drivers do not detect or recover from device failures
   ```c
   if (inb(regA)!=5) {
     panic();
   }
   ```
# Vendor recommendations for driver developers

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Summary</th>
<th>Intel</th>
<th>Sun</th>
<th>MS</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>Input validation</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Read once &amp; CRC data</td>
<td>•</td>
<td>•</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>DMA protection</td>
<td>•</td>
<td>•</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Timing</td>
<td>Infinite polling</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Stuck interrupt</td>
<td></td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Lost request</td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Avoid excess delay in OS</td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Unexpected events</td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Reporting</td>
<td>Report all failures</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Recovery</td>
<td>Handle all failures</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Cleanup correctly</td>
<td>•</td>
<td>•</td>
<td></td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Do not crash on failure</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Wrap I/O memory access</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>
Vendor recommendations for driver developers

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Summary</th>
<th>Recommended by</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Intel</td>
</tr>
<tr>
<td>Validation</td>
<td>Input validation</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Read once &amp; CRC data</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>DMA protection</td>
<td>•</td>
</tr>
<tr>
<td>Timing</td>
<td>Infinite polling</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Stuck interrupt</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Lost request</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Avoid excess delay in OS</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Unexpected events</td>
<td>•</td>
</tr>
<tr>
<td>Reporting</td>
<td>Report all failures</td>
<td>•</td>
</tr>
<tr>
<td>Recovery</td>
<td>Handle all failures</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Cleanup correctly</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Do not crash on failure</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Wrap I/O memory access</td>
<td>•</td>
</tr>
</tbody>
</table>
## Vendor recommendations for driver developers

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Summary</th>
<th>Recommended by</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Intel</td>
</tr>
<tr>
<td><strong>Validation</strong></td>
<td>Input validation</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Read once &amp; CRC data</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>DMA protection</td>
<td>✔</td>
</tr>
<tr>
<td><strong>Timing</strong></td>
<td>Infinite polling</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Stuck interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lost request</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avoid excess delay in OS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unexpected events</td>
<td></td>
</tr>
<tr>
<td><strong>Reporting</strong></td>
<td>Report all failures</td>
<td>✔</td>
</tr>
<tr>
<td><strong>Recovery</strong></td>
<td>Handle all failures</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Cleanup correctly</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Do not crash on failure</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td>Wrap I/O memory access</td>
<td>✔</td>
</tr>
</tbody>
</table>
Vendor recommendations for driver developers

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Summary</th>
<th>Intel</th>
<th>Sun</th>
<th>MS</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Validation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input validation</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Read once &amp; CRC data</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>DMA protection</td>
<td></td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td><strong>Timing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infinite polling</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Stuck interrupt</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lost request</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Avoid excess delay in OS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>Unexpected events</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td><strong>Reporting</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report all failures</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td><strong>Recovery</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handle all failures</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Cleanup correctly</td>
<td></td>
<td></td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Do not crash on failure</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Wrap I/O memory access</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>
## Vendor recommendations for driver developers

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Summary</th>
<th>Intel</th>
<th>Sun</th>
<th>MS</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>Input validation</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read once &amp; CRC data</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>DMA protection</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing</td>
<td>Infinite polling</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stuck interrupt</td>
<td></td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lost request</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>Avoid excess delay in OS</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>Unexpected events</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reporting</td>
<td>Report all failures</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td>Recovery</td>
<td>Handle all failures</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>Cleanup correctly</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Do not crash on failure</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
</tr>
<tr>
<td></td>
<td>Wrap I/O memory access</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
</tbody>
</table>
## Vendor recommendations for driver developers

**Goal:** Automatically implement as many recommendations as possible in commodity drivers

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Summary</th>
<th>Intel</th>
<th>Sun</th>
<th>MS</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>Input validation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Read once &amp; CRC data</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>DMA protection</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timing</td>
<td>Infinite polling</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Reporting</td>
<td>Report all failures</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Handle all failures</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Cleanup correctly</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Do not crash on failure</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Wrap I/O memory access</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Goal: Tolerate hardware device failures in software through hardware failure detection and recovery
Goal: Tolerate hardware device failures in software through hardware failure detection and recovery

**Static analysis component**

- Detect and fix hardware dependence bugs
- Detect and generate missing error reporting information
Goal: Tolerate hardware device failures in software through hardware failure detection and recovery

**Static analysis component**
- Detect and fix hardware dependence bugs
- Detect and generate missing error reporting information

**Runtime component**
- Detect interrupt failures
- Provide automatic recovery
Bug detection and automatic fix generation

If \( c = 0 \) {
    print("Driver init");
}

Driver
Carburizer architecture

Bug detection and automatic fix generation

```java
if (c==0) {
  print("Driver init");
}
```
Carburizer architecture

Bug detection and automatic fix generation

Carburizer

Compiler

Driver

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

Code snippet:
Bug detection and automatic fix generation

Carburizer architecture

Recovery and interrupt watchdog

OS Kernel

Carburizer

Compiler

Driver

Hardened Driver Binary

Original code:
```c
if (c==0) {
  print ("Driver Init");
}
```

Modified code:
```c
if (c==0) {
  print ("Driver Init");
}
```
Bug detection and automatic fix generation

Recovery and interrupt watchdog

OS Kernel

Carburizer architecture

Carburizer
Compiler
Driver

Compiler

Kernel Interface

Hardened Driver Binary

Carburizer Runtime

Faulty Hardware

Driver

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

If (c==0) {
  print ("Driver Init");
}
Hardening drivers
Hardening drivers

- **Goal: Remove hardware dependence bugs**
  - Find driver code that uses data from device
  - Ensure driver performs validity checks
Hardening drivers

• **Goal: Remove hardware dependence bugs**
  ★ Find driver code that uses data from device
  ★ Ensure driver performs validity checks

• **Carburizer detects and fixes hardware bugs:**
Hardening drivers

• **Goal: Remove hardware dependence bugs**
  ★ Find driver code that uses data from device
  ★ Ensure driver performs validity checks

• **Carburizer detects and fixes hardware bugs:**

  **Infinite polling**
Hardening drivers

• **Goal: Remove hardware dependence bugs**
  - Find driver code that uses data from device
  - Ensure driver performs validity checks

• **Carburizer detects and fixes hardware bugs:**
  - Infinite polling
  - Unsafe pointer reference
Hardening drivers

- **Goal: Remove hardware dependence bugs**
  - Find driver code that uses data from device
  - Ensure driver performs validity checks

- **Carburizer detects and fixes hardware bugs:**
  - Infinite polling
  - Unsafe pointer reference
  - Unsafe array reference
Hardening drivers

- **Goal:** Remove hardware dependence bugs
  - Find driver code that uses data from device
  - Ensure driver performs validity checks

- **Carburizer detects and fixes hardware bugs:**
  - Infinite polling
  - Unsafe pointer reference
  - Unsafe array reference
  - System panic calls
Finding sensitive code

★ First pass: Identify tainted variables that contain data from device

Types of device I/O

★ Port I/O: inb/outb
★ Memory-mapped I/O: readl/writel
★ DMA buffers
★ Data from USB packets

OS

network card
Finding sensitive code

* First pass: Identify tainted variables that contain data from device

```c
int test () { 
```
Finding sensitive code

* First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = read1();
}
```
Finding sensitive code

* First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = read1();
}
```

Tainted Variables
- a
Finding sensitive code

★ First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = readl();
    b = inb();
}
```

Tainted Variables

a
Finding sensitive code

* First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = read1();
    b = inb();
}
```

Tainted Variables

- a
- b
Finding sensitive code

★ First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = readl();
    b = inb();
    c = b;
}
```

Tainted Variables

- a
- b
Finding sensitive code

* First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = read1();
    b = inb();
    c = b;
}
```

Tainted Variables

- a
- b
- c
Finding sensitive code

* First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = readl();
    b = inb();
    c = b;
    d = c + 2;
}
```

Tainted Variables

- a
- b
- c
Finding sensitive code

★ First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = readl();
    b = inb();
    c = b;
    d = c + 2;
}
```

Tainted Variables

- a
- b
- c
- d
Finding sensitive code

First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = read1();
    b = inb();
    c = b;
    d = c + 2;
    return d;
}
```

Tainted Variables

- a
- b
- c
- d
Finding sensitive code

* First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = readl();
    b = inb();
    c = b;
    d = c + 2;
    return d;
}
```

Tainted Variables:
- a
- b
- c
- d
- test()
Finding sensitive code

★ First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = readl();
    b = inb();
    c = b;
    d = c + 2;
    return d;
}
```

Tainted Variables

- a
- b
- c
- d
- test()
Finding sensitive code

★ First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = readl();
    b = inb();
    c = b;
    d = c + 2;
    return d;
}

int set() {
```

**Tainted Variables**

- a
- b
- c
- d
- test()
Finding sensitive code

★ First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = read1();
    b = inb();
    c = b;
    d = c + 2;
    return d;
}
int set() {
    e = test();
}
```

Tainted Variables

- a
- b
- c
- d
- test()
Finding sensitive code

* First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = readl();
    b = inb();
    c = b;
    d = c + 2;
    return d;
}
int set() {
    e = test();
}
```

Tainted Variables
- a
- b
- c
- d
- test()
- e
Finding sensitive code

* First pass: Identify tainted variables that contain data from device

```c
int test () {
    a = read1();
    b = inb();
    c = b;
    d = c + 2;
    return d;
}
int set() {
    e = test();
}
```

Tainted Variables

- a
- b
- c
- d
- test()
- e
Detecting risky uses of tainted variables

- **Finding sensitive code**
  - Second pass: Identify risky uses of tainted variables

- **Example: Infinite polling**
  - Driver waiting for device to enter particular state
  - Solution: Detect loops where all terminating conditions depend on tainted variables
Infinite polling

★ Infinite polling of devices can cause system lockups

```c
static int amd8111e_read_phy(………)
{
  ...
  reg_val = readl(mmio + PHY_ACCESS);
  while (reg_val & PHY_CMD_ACTIVE)
    reg_val = readl(mmio + PHY_ACCESS)
  ...
}
```

AMD 8111e network driver(amd8111e.c)
Hardware data used in array reference

* Tainted variables used as array indexes

```c
static void __init attach_pas_card(...)
{
    if ((pas_model = pas_read(0xFF88)))
    {
        ...  
        sprintf(temp, "%s rev %d",
               pas_model_names[(int) pas_model], pas_read(0x2789));
        ...
    }
}
```

Pro Audio Sound driver (pas2_card.c)
Experience with the Linux kernel
Experience with the Linux kernel

- Extra analyses to reduce false positives
  - Detect counters, range and not NULL checks
  - Detect taint lifetimes
Experience with the Linux kernel

- Extra analyses to reduce false positives
  - Detect counters, range and not NULL checks
  - Detect taint lifetimes
- Analyzed drivers in 2.6.18.8 Linux kernel
  - 6300 driver source files
  - 2.8 million lines of code
  - 37 minutes to analyze and compile code
Analysis results over the Linux kernel

<table>
<thead>
<tr>
<th>Driver class</th>
<th>Infinite polling</th>
<th>Static array</th>
<th>Dynamic array</th>
<th>Panic calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>net</td>
<td>117</td>
<td>2</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>scsi</td>
<td>298</td>
<td>31</td>
<td>22</td>
<td>121</td>
</tr>
<tr>
<td>sound</td>
<td>64</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>video</td>
<td>174</td>
<td>0</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>other</td>
<td>381</td>
<td>9</td>
<td>57</td>
<td>32</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>860</strong></td>
<td><strong>43</strong></td>
<td><strong>89</strong></td>
<td><strong>179</strong></td>
</tr>
</tbody>
</table>

- Found **992** hardware dependence bugs in driver code
- False positive rate: 7.4% (manual sampling of 190 bugs)
Analysis results over the Linux kernel

<table>
<thead>
<tr>
<th>Driver class</th>
<th>Infinite polling</th>
<th>Static array</th>
<th>Dynamic array</th>
<th>Panic calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>net</td>
<td>117</td>
<td>2</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>scsi</td>
<td>298</td>
<td>31</td>
<td>22</td>
<td>121</td>
</tr>
<tr>
<td>sound</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>video</td>
<td>64</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>other</td>
<td>174</td>
<td>0</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Total</td>
<td>860</td>
<td>43</td>
<td>89</td>
<td>179</td>
</tr>
</tbody>
</table>

Lightweight and usable technique to find hardware dependence bugs

- Found **992** hardware dependence bugs in driver code
- False positive rate: **7.4%** (manual sampling of 190 bugs)
Repairing drivers

- Carburizer automatically generates repair code
  - Inserts failure detection and recovery service callout
Repairing drivers

- Carburizer automatically generates repair code
- Inserts failure detection and recovery service callout

- Infinite polling
- Unsafe array reference
- Unsafe pointer reference
- System panic calls
Repairing drivers

- Carburizer automatically generates repair code
- Inserts failure detection and recovery service callout

Timeout checks

- Infinite polling
- Unsafe array reference
- Unsafe pointer reference
- System panic calls
Repairing drivers

- Carburizer automatically generates repair code
  - Inserts failure detection and recovery service callout

- Timeout checks
- Array bounds check
- Infinite polling
- Unsafe array reference
- Unsafe pointer reference
- System panic calls
Repairing drivers

★ Carburizer automatically generates repair code
★ Inserts failure detection and recovery service callout

Timeout checks
Array bounds check
Not null checks

Infinite polling
Unsafe array reference
Unsafe pointer reference
System panic calls
Repairing drivers

Call recovery service

- Timeout checks
- Array bounds check
- Not null checks
- Infinite polling
- Unsafe array reference
- Unsafe pointer reference
- System panic calls
Runtime fault recovery

- Carburizer calls generic recovery service if check fails
- Low cost transparent recovery
  - Based on shadow drivers
  - Records state of driver
  - Transparent restart and state replay on failure
- No isolation required (like Nooks)

Swift [OSDI ’04]
timeout = rdtsc1l(start) + (cpu/khz/HZ)*2;
reg_val = read1(mmio + PHY_ACCESS);
while (reg_val & PHY_CMD_ACTIVE) {
    reg_val = read1(mmio + PHY_ACCESS);
    if (_cur < timeout)
        rdtsc1l(_cur);
    else
        ___recover_driver();
}

Timeout code added

AMD 8111e network driver(amd8111e.c)

*Code simplified for presentation purposes
static void __init attach_pas_card(...) {

    if ((pas_model = pas_read(0xFF88))) {
        ...
        if ((pas_model< 0)) || (pas_model>= 5))
            __recover_driver();
        .
        sprintf(temp, "%s rev %d",
                pas_model_names[(int) pas_model],
                pas_read(0x2789));
    }

*Code simplified for presentation purposes
Fault injection validation

★ Synthetic fault injection on network drivers
★ Results
## Fault injection validation

- Synthetic fault injection on network drivers
- Results

<table>
<thead>
<tr>
<th>Device/Driver</th>
<th>Original Driver</th>
<th>Carburizer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Behavior</td>
<td>Detection</td>
</tr>
<tr>
<td>3COM 3C905</td>
<td>CRASH</td>
<td>None</td>
</tr>
<tr>
<td>DEC DC 21x4x</td>
<td>CRASH</td>
<td>None</td>
</tr>
</tbody>
</table>
Fault injection validation

- Synthetic fault injection on network drivers
- Results

<table>
<thead>
<tr>
<th>Device/Driver</th>
<th>Original Driver</th>
<th>Carburizer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Behavior</td>
<td>Detection</td>
</tr>
<tr>
<td>3COM 3C905</td>
<td>CRASH</td>
<td>None</td>
</tr>
<tr>
<td>DEC DC 21x4x</td>
<td>CRASH</td>
<td>None</td>
</tr>
</tbody>
</table>

Carburizer failure detection and transparent recovery work well for transient device failures
Throughput overhead

- Linux Kernel
- Carburizer Kernel

No CPU overhead

netperf on 2.2 GHz AMD machines
Throughput overhead

Netperf on 2.2 GHz AMD machines

Network Card Type

- nVIDIA MCP 55
- Intel Pro 1000

Throughput in Mbps

0

250

500

750

1000

No CPU overhead

Linux Kernel

Carburizer Kernel
Throughput overhead

Network Card Type
netperf on 2.2 GHz AMD machines
Throughput overhead

Throughput in Mbps

- nVIDIA MCP 55
  - Linux Kernel: 940
  - Carburizer Kernel: 935
- Intel Pro 1000
  - Linux Kernel: 721
  - Carburizer Kernel: 720

Network Card Type
netperf on 2.2 GHz AMD machines

No CPU overhead
Throughput overhead

Throughput in Mbps

<table>
<thead>
<tr>
<th>Network Card Type</th>
<th>Throughput in Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>nVIDIA MCP 55</td>
<td>940 935</td>
</tr>
<tr>
<td>Intel Pro 1000</td>
<td>721 720</td>
</tr>
</tbody>
</table>

Almost no overhead from hardened drivers and automatic recovery
Outline

- Tolerate device failures
- Reporting failures
- Hardening drivers
- Runtime Fault tolerance
- Results
- Transactional approach for cheap recovery
- Understand drivers and potential opportunities
Outline

- Tolerate device failures
- Hardening drivers
  Reporting failures
  Runtime Fault tolerance
  Results
- Understand drivers and potential opportunities
- Transactional approach for cheap recovery
Runtime failure detection

★ Static analysis cannot detect all device failures

- **Missing interrupts**
  - Interrupt expected but never arrives

- **Stuck interrupts**
  - Interrupt cleared but continues to assert
Missing interrupts

Diagram:
- OS
  - Driver
  - Hardware Device
Missing interrupts

- OS
- Driver
- Hardware Device
Missing interrupts

- Device polling on interrupt failures
  - Polling frequently has high overhead
  - Polling infrequently results in throughput loss
Missing interrupts

- **Device polling on interrupt failures**
  - Polling frequently has high overhead
  - Polling infrequently results in throughput loss

- **How frequently should we poll?**
  - Increase frequency if interrupt invocation did useful work
Missing interrupts

- **Device polling on interrupt failures**
  - Polling frequently has high overhead
  - Polling infrequently results in throughput loss

- **How frequently should we poll?**
  - Increase frequency if interrupt invocation did useful work

- **When are requests likely to come?**
  - Driver invocation: Use reference bits to detect driver activity
Stuck interrupts

- Driver interrupt handler is called too many times
- Convert the device from interrupts to polling
Stuck interrupts

- Driver interrupt handler is called too many times
- Convert the device from interrupts to polling

<table>
<thead>
<tr>
<th>Driver Type</th>
<th>Driver Name</th>
<th>Native</th>
<th>With Carburizer</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>ide-core,ide-disk, ide-generic</td>
<td>Hang</td>
<td>Reduced by 50%</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>e1000</td>
<td>Hang</td>
<td>Reduced from 750 Mb/s to 130 Mb/s</td>
<td></td>
</tr>
<tr>
<td>Sound</td>
<td>ens1371</td>
<td>Hang</td>
<td>Sounds plays with distortion</td>
<td></td>
</tr>
</tbody>
</table>
Stuck interrupts

- Driver interrupt handler is called too many times
- Convert the device from interrupts to polling

<table>
<thead>
<tr>
<th>Driver Type</th>
<th>Driver Name</th>
<th>Native</th>
<th>With Carburizer</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td>ide-core, ide-disk, ide-generic</td>
<td>Hang</td>
<td>Reduced by 50%</td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>e1000</td>
<td>Hang</td>
<td>Reduced from 750 Mb/s to 130 Mb/s</td>
<td></td>
</tr>
<tr>
<td>Sound</td>
<td>ens1371</td>
<td>Hang</td>
<td>Sounds plays with distortion</td>
<td></td>
</tr>
</tbody>
</table>

Carburizer ensures system makes forward progress
<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Summary</th>
<th>CPU</th>
<th>Sun</th>
<th>MS</th>
<th>Linux</th>
<th>Ensures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Validation</strong></td>
<td>Input validation</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td></td>
<td>Read once &amp; CRC data</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td></td>
<td>DMA protection</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td><strong>Timing</strong></td>
<td>Infinite polling</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td></td>
<td>Stuck interrupt</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td></td>
<td>Lost request</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td></td>
<td>Avoid excess delay in OS</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td></td>
<td>Unexpected events</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td><strong>Reporting</strong></td>
<td>Report all failures</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td><strong>Recovery</strong></td>
<td>Handle all failures</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td></td>
<td>Cleanup correctly</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td></td>
<td>Do not crash on failure</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
<tr>
<td></td>
<td>Wrap I/O memory access</td>
<td>![Intel]</td>
<td>![Sun]</td>
<td>![MS]</td>
<td>![Linux]</td>
<td>![Ensures]</td>
</tr>
</tbody>
</table>
## Summary

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Summary</th>
<th>Intel</th>
<th>Sun</th>
<th>MS</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Validation</strong></td>
<td>Input validation</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
</tr>
<tr>
<td></td>
<td>Read once &amp; CRC data</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
</tr>
<tr>
<td></td>
<td>DMA protection</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td></td>
</tr>
<tr>
<td><strong>Timing</strong></td>
<td>Infinite polling</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
</tr>
<tr>
<td></td>
<td>Stuck interrupt</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
</tr>
<tr>
<td></td>
<td>Lost request</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
</tr>
<tr>
<td></td>
<td>Avoid excess delay in OS</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
</tr>
<tr>
<td></td>
<td>Unexpected events</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
</tr>
<tr>
<td>Wrap I/O memory access</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
<td>![Recommendation]</td>
</tr>
</tbody>
</table>

**Carburizer** improves system reliability by automatically ensuring that hardware failures are tolerated in software.
Contributions beyond research

- Informed developers at Plumbers Conference [2011]
- LWN Article with paper & list of bugs [Feb ‘12]
- Released patches to the Linux kernel
- Tool + source available for download at:
Functionality: Recovery assumes drivers follow class behavior

- Record state by interposing class defined entry points
- Restart and replay state using class semantics when failure happens
Functionality: Recovery assumes drivers follow class behavior

- Record state by interposing class defined entry points
- Restart and replay state using class semantics when failure happens

Non-class behavior can lead to incomplete restore after failure
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
Outline

- Tolerate device failures
- Understand drivers and potential opportunities
- Transactional approach for cheap recovery
- Overview
  - Recovery specific results
Our view of drivers is narrow

Drivers
6.7 million LOC in Linux
Our view of drivers is narrow

Drivers
6.7 million LOC in Linux

Driver Research (avg. 2.2 drivers/system)
Our view of drivers is narrow

Drivers
6.7 million LOC in Linux

Driver Research (avg. 2.2 drivers/system)

Bugs
Our view of drivers is narrow

Drivers
6.7 million LOC in Linux

Necessary to review driver code in modern settings

Driver Research
(avg. 2.2 drivers/system)

Bugs
Study source of all Linux drivers for x86 (~3200 drivers)
Understanding Modern Device Drivers [ASPLOS 2012]

Study source of all Linux drivers for x86 (~3200 drivers)

- Code properties
- Verify research assumptions
Understanding Modern Device Drivers

Study source of all Linux drivers for x86 (~3200 drivers)

- **Driver properties**
  - Code properties
  - Verify research assumptions

- **Driver interaction**
  - Driver kernel & device interaction
  - Driver architecture
Understanding Modern Device Drivers

Study source of all Linux drivers for x86 (~3200 drivers)

- Code properties
- Verify research assumptions
- Driver kernel & device interaction
- Driver architecture
- 7 million lines of code needed?
Study methodology

- Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)
Study methodology

★ Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)

★ Identify driver entry points, kernel and bus callouts
Study methodology

★ Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)

★ Identify driver entry points, kernel and bus callouts
★ Device class, sub-class
Study methodology

★ Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)

★ Identify driver entry points, kernel and bus callouts
  ★ Device class, sub-class
  ★ Driver functions registered as entry points (purpose)
Study methodology

★ Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)

★ Identify driver entry points, kernel and bus callouts
  ★ Device class, sub-class
  ★ Driver functions registered as entry points (purpose)
  ★ Bus properties
Study methodology

★ Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)

★ Identify driver entry points, kernel and bus callouts
  ★ Device class, sub-class
  ★ Driver functions registered as entry points (purpose)
  ★ Bus properties
  ★ Other properties (module params)
Study methodology

- Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)

- Identify driver entry points, kernel and bus callouts
  - Device class, sub-class
  - Driver functions registered as entry points (purpose)
  - Bus properties
  - Other properties (module params)

For every merged driver
Study methodology

- Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)

- Identify driver entry points, kernel and bus callouts
  - Device class, sub-class
  - Driver functions registered as entry points (purpose)
  - Bus properties
  - Other properties (module params)

Driver properties

For every merged driver

Driver entry points

- xmit
- open
- close
- probe
Study methodology

★ Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)

★ Identify driver entry points, kernel and bus callouts

★ Reverse propagate information to aggregate bus, device and kernel behavior

Driver properties

Driver interactions

xmit
open
probe
xmit
open
probe
kmalloc
close
Study methodology

- Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)
- Identify driver entry points, kernel and bus callouts
- Reverse propagate information to aggregate bus, device and kernel behavior
Study methodology

- Static source analysis of 3200 drivers in Linux 2.6.37.6 (May 2011)

- Identify driver wide and function specific properties of all drivers

- Reverse propagate information to aggregate bus, device and kernel behavior

- Use statistical clustering techniques and static analysis to identify similar code
Some additional results

Driver properties

- Many assumptions made by driver research does not hold:
  - 15% drivers perform significant processing
  - 28% drivers support multiple chipsets

Driver interactions

- USB bus offers efficient access (as compared to PCI, Xen)
  - Supports high # devices/driver (standardized code)
  - Coarse-grained access

Driver similarity

- 400, 000 lines of code similar to code elsewhere and ripe for improvement via:
  - Procedural abstractions
  - Better multiple chipset support
  - Table driver programming
Contributions/Outline

- Tolerate device failures
- Understand drivers and potential opportunities
- Transactional approach for cheap recovery

Overview
Recovery specific results
Driver Code
Characteristics
## Driver Code Characteristics

- **Core I/O & interrupts** – 23%
- **Initialization/cleanup** – 36%
- **Device configuration** – 15%
- **Power management** – 7.4%
- **Device ioctl** – 6.2%

### Driver Code Characteristics

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Percentage of LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core I/O &amp; interrupts</td>
<td>23%</td>
</tr>
<tr>
<td>Initialization/cleanup</td>
<td>36%</td>
</tr>
<tr>
<td>Device configuration</td>
<td>15%</td>
</tr>
<tr>
<td>Power management</td>
<td>7.4%</td>
</tr>
<tr>
<td>Device ioctl</td>
<td>6.2%</td>
</tr>
</tbody>
</table>

### Percent- age of LOC

![Color scale for LOC percentage](image)
Driver Code Characteristics

- Core I/O & interrupts – 23%
- Initialization/cleanup – 36%
- Device configuration – 15%
- Power management – 7.4%
- Device ioctl – 6.2%

Initialization code dominates driver LOC and adds to complexity
Recovery assumes drivers follow class behavior

Class definition includes:
- Callbacks registered with the bus, device and kernel subsystem
- Exported APIs of the kernel to use kernel resources and services
Recovery assumes drivers follow class behavior

Does driver behavior belong to class definitions?

★ Class definition includes:

★ Callbacks registered with the bus, device and kernel subsystem

★ Exported APIs of the kernel to use kernel resources and services
Do drivers belong to classes?

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

```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)
Do drivers belong to classes?

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

Results as measured by our analyses:
- 16% of drivers use proc/sysfs support
- 36% of drivers use load time parameters
- 16% of drivers use ioctl that may include non-standard behavior

Overall, 44% of drivers do not conform to class behavior
Outline

- Tolerate device failures
- Understand drivers and potential opportunities
- Transactional approach for cheap recovery
- Checkpoint/restore
  FGFT
  Future work and conclude
Limitations of restart/replay recovery

- **Device save/restore limited to restart/replay**
  - Slow: Device initialization is complex (multiple seconds)
  - Not enough: Incomplete recovery due to unique semantics
  - Hard: Need to be written for every class of drivers
  - Expensive: Continuous logging of all driver operations
Limitations of restart/replay recovery

- Device save/restore limited to restart/replay
  - Slow: Device initialization is complex (multiple seconds)
  - Not enough: Incomplete recovery due to unique semantics
  - Hard: Need to be written for every class of drivers
  - Expensive: Continuous logging of all driver operations

Checkpoint/restore of device and driver state removes the need to reboot device and replay state
Checkpoint/Restore

* Checkpoints limited to capturing **memory** state

(network driver)

(network card)
★ Checkpoints limited to capturing memory state
Checkpoint/Restore

* Checkpoints limited to capturing memory state

* Device state is not captured
★ Checkpoints limited to capturing memory state

★ Device state is not captured
  ★ Device configuration space
Checkpoint/Restore

- Checkpoints limited to capturing memory state

- Device state is not captured
  - Device configuration space
  - Internal device registers and counters
★ Checkpoints limited to capturing memory state

★ Device state is not captured
  ★ Device configuration space
  ★ Internal device registers and counters
  ★ Memory buffer addresses used for DMA
Power management in drivers
Power management in drivers

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

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

★ Suspend to RAM: Save state and suspend processors and devices

★ Refactor power management code for checkpoint/restore
  ★ Correct: Driver developer captures unique semantics
  ★ Fast: Avoids probe and latency critical for applications
Checkpoint/Restore from PM code

**Suspend**
- Save config state
- Save device 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
# Checkpoint/Restore from PM code

## Suspend
- Save config state
- Save device 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
Checkpoint/Restore from PM code

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

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

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

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

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

**Resume**
- Restore config state
- Restore register state
- Restore s/w state & reset
- Re-attach/Enable device
Checkpoint/Restore from PM code

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

**Resume**
- Restore config state
- Restore register state
- Restore s/w state & reset
Checkpoint/Restore from PM code

Suspend

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

Resume

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

Suspend/resume code provides checkpoint functionality
Fine-Grained Fault Tolerance [ASPLOS 2013]

- **Use device checkpoints** to improve recovery
- **Execute driver entry points as transactions**
  - Take a device checkpoint, run driver as memory transaction
  - If the driver fails, we abort memory transaction and restore the checkpoint
- **Provide memory safety and trap processor exceptions**
- **Recovery is simple and fast**

- **Developers export checkpoint/restore in all drivers**
Fine-Grained Isolation

- get ringparam
- probe
- xmit
- get config

network driver

SFI network driver
Fine-Grained Isolation

- get ringparam
- probe
- xmit
- get config

★ Suspect entry point arrives

network driver

SFI network driver
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device
Fine-Grained Isolation

- **Suspect entry point arrives**
- **Checkpoint device**
- **Marshal required data in SFI**
Fine-Grained Isolation

network driver

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

★ Suspect entry point arrives
★ Checkpoint device
★ Marshal required data in SFI

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
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device
- Marshal required data in SFI

<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>
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device
- Marshal required data in SFI

**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**

**network driver**

**SFI network driver**

**C**

**get config**

**probe**

**xmit**

**get ringparam**
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device
- Marshal required data in SFI
- Populate 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
```

Kernel Log

```
C
```

```
probe
xmit
get config
get ringparam
```

```
SFI network driver
```

```
Range Table
```

```
network driver
```

```
★ Suspect entry point arrives
★ Checkpoint device
★ Marshal required data in SFI
★ Populate range table
```
Fine-Grained Isolation

- get ringparam
- probe
- xmit
- get config

- Suspect entry point arrives
- Checkpoint device
- Marshal required data in SFI
- Populate range table
- Execute & Populate compensation log

**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>

**SFI network driver**

**Kernel Log lock**
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device
- Marshal required data in SFI
- Populate range table
- Execute & Populate compensation log
- Success: Copy back written data

Address | Access rights
---|---
0xffffa000 | Read
0xffffa008 | Write
0xffffa00a | Read

get ringparam
probe
xmit
get config

SFI network driver

Range Table

Kernel Lock

lock

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

C

network driver

★ Suspect entry point arrives
★ Checkpoint device
★ Marshal required data in SFI
★ Populate range table
★ Execute & Populate compensation log
★ Success: Copy back written data
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device
- Marshal required data in SFI
- Populate range table
- Execute & Populate compensation log
- Success: Copy back written data
Fine-Grained Isolation

Network driver

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

Resource access
- I/O memory: Full access
- Locks: Read access & locks acquired via kernel
- Memory: Allocate & add to range table

- Suspect entry point arrives
- Checkpoint device
- Marshal required data in SFI
- Populate range table
- Execute & Populate compensation log
- Success: Copy back written data

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

- lock
Fine-Grained Isolation

get ringparam
probe
xmit
get config

network driver

SFI network driver
Fine-Grained Isolation

★ Suspect entry point arrives
Fine-Grained Isolation

- **Suspect entry point arrives**
- **Checkpoint device and processor state**
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device and processor state
- Marshal required data in SFI

```c
netdev->priv->tx_ring
netdev->priv->rx_ring
```
Fine-Grained Isolation

```
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>

- ★ Suspect entry point arrives
- ★ Checkpoint device and processor state
- ★ Marshal required data in SFI
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device and processor state
- Marshal required data in SFI

**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 config → xmit → probe

get ringparam

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

C

SFI network driver
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device and processor state
- Marshal required data in SFI

```c
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>
Fine-Grained Isolation

**Range Table**

<table>
<thead>
<tr>
<th>Address</th>
<th>Access rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffffff</td>
<td>Read</td>
</tr>
<tr>
<td>0xffffffff8</td>
<td>Write</td>
</tr>
<tr>
<td>0xffffffffa</td>
<td>Read</td>
</tr>
</tbody>
</table>

**Suspect entry point arrives**

**Checkpoint device and processor state**

**Marshal required data in SFI**

**Populate range table**

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

```
get config
get ringparam
probe
xmit
```

```
Kernel Log
```
Fine-Grained Isolation

network driver

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 lock

★ Suspect entry point arrives
★ Checkpoint device and processor state
★ Marshal required data in SFI
★ Populate range table
★ Execute & Populate compensation log
Fine-Grained Isolation

```
<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>
```

- Suspect entry point arrives
- Checkpoint device and processor state
- Marshal required data in SFI
- Populate range table
- Execute & Populate compensation log
Fine-Grained Isolation

**network driver**

- `netdev->priv->tx_ring`
- `netdev->priv->rx_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>

- Suspect entry point arrives
- Checkpoint device and processor state
- Marshal required data in SFI
- Populate range table
- Execute & Populate compensation log
Fine-Grained Isolation

**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**

- probe
- xmit
- get config
- get ringparam
- netdev->priv->rx_ring
- netdev->priv->tx_ring

**SFI network driver**

- Suspect entry point arrives
- Checkpoint device and processor state
- Marshal required data in SFI
- Populate range table
- Execute & Populate compensation log
- Fail: Restore processor and device state, release locks

**Kernel Log lock**

- ★ Suspect entry point arrives
- ★ Checkpoint device and processor state
- ★ Marshal required data in SFI
- ★ Populate range table
- ★ Execute & Populate compensation log
- ★ Fail: Restore processor and device state, release locks
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device and processor state
- Marshal required data in SFI
- Populate range table
- Execute & Populate compensation log
- Fail: Restore processor and device state, release locks

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>
Fine-Grained Isolation

- Suspect entry point arrives
- Checkpoint device and processor state
- Marshal required data in SFI
- Populate range table
- Execute & Populate compensation log
- Fail: Restore processor and device state, release locks

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>0xffffa0oa</td>
<td>Read</td>
</tr>
</tbody>
</table>

Network driver

C

network driver

netdev->priv->rx_ring

netdev->priv->tx_ring

get ringparam

probe

xmit

get config

err

SFI

network driver
Fine-Grained Isolation

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>

★ Suspect entry point arrives
★ Checkpoint device and processor state
★ Marshal required data in SFI
★ Populate range table
★ Execute & Populate compensation log
★ Fail: Restore processor and device state, release locks
Fine-Grained Isolation

**FGFT provides transactional execution of driver entry points**

- Suspect entry point arrives
- Checkpoint device and processor state
- Fail: Restore processor and device state, release locks

<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>
## 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 speedup in driver recovery
### Programming effort

<table>
<thead>
<tr>
<th>Driver</th>
<th>LOC</th>
<th>Recovery additions</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LOC Moved</td>
<td>LOC Added</td>
<td></td>
</tr>
<tr>
<td>8139too</td>
<td>1,904</td>
<td>26</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>e1000</td>
<td>13,973</td>
<td>32</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>r8169</td>
<td>2,993</td>
<td>17</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>pegasus</td>
<td>1,541</td>
<td>22</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>ens1371</td>
<td>2,110</td>
<td>16</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>psmouse</td>
<td>2,448</td>
<td>19</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

**FGFT requires limited annotation support and needs only 38 lines of new kernel code**
Throughput overhead

**CPU:** 2.4%  2.4%  2.9%  3.4%

*netperf on Intel quad-core machines*
Throughput overhead

CPU: 2.4%  2.4%  2.9%  3.4%

Native
FGFT-static
FGFT-dynamic-1/2
FGFT-dynamic-all

netperf on Intel quad-core machines
Throughput overhead

Throughput %age (Baseline 844 Mbps)

Throughput overhead

CPU: 2.4% 2.4% 2.9% 3.4%

Network Card Type

Native
FGFT-static
FGFT-dynamic-1/2
FGFT-dynamic-all

netperf on Intel quad-core machines
Throughput overhead

Network Card Type

Native
FGFT-static
FGFT-dynamic-1/2
FGFT-dynamic-all

Throughput % age (Baseline 844 Mbps)

Throughput overhead

CPU: 2.4% 2.4% 2.9% 3.4%

netperf on Intel quad-core machines
Throughput overhead

Throughput %age (Baseline 844 Mbps)

*CPU*: 2.4%  2.4%  2.9%  3.4%

- Native
- FGFT-static
- FGFT-dynamic-1/2
- FGFT-dynamic-all

**netperf on Intel quad-core machines**
Throughput overhead

CPU: 2.4% 2.4% 2.9% 3.4%

Network Card Type
- Native
- FGFT-static
- FGFT-dynamic-1/2
- FGFT-dynamic-all

netperf on Intel quad-core machines
Summary

★ Investigated the problem of device failures in OS

★ Developed static and runtime solutions, contributed patches and a talk to developer community

★ Took a holistic view of research solutions and identified new research opportunities

★ Addressed one of these findings, and introduced checkpoint/restore in modern drivers for fast recovery
Outline

Tolerate device failures

Understand drivers and potential opportunities

Transactional approach for cheap recovery

Checkpoint/restore
FGFT
Other/Future Work
<table>
<thead>
<tr>
<th>Storage</th>
<th>Differential RAID [Eurosys '10]</th>
<th>GPFS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SymDrive [OSDI '12]</td>
<td>ThinCloud [Under Submission]</td>
</tr>
<tr>
<td>Drivers</td>
<td>FGFT [ASPLOS '13]</td>
<td>Live Migration [OSR '09]</td>
</tr>
<tr>
<td></td>
<td>Carburizer [SOSP '09]</td>
<td></td>
</tr>
</tbody>
</table>

- Reliability
- Performance
- Measurement

Papers at http://cs.wisc.edu/~kadav
Future Work

- Use prior experience in
  - Operating Systems
  - Distributed Systems
  - Software Reliability
  - Program Analysis

- Solve new problems
- Build real systems
- Measure real impact
- Improve
Future Work: Lessons from reliability research

★ Distributed Systems: Identify and automatically fix cluster specific issues: expired leases, stale views, flooding (cascading failures)

★ Distributed Systems: How to create lightweight, broad and consistent checkpoints?

★ Automatically fix problems in other plugin based architectures like app stores, browsers
Future Work: Investigate OS-hardware co-design

- **Co-design**: Co-design OS and device abstractions
  - Integrating energy proportional DRAM in OS
  - Use special purpose workloads to accelerate cloud workloads
  - Re-design I/O in clusters for remote access

- **Co-verification**: Device protocol violations
  - Extend existing work on device failures to detect inconsistencies in software-device interaction
Example: Energy Proportional DRAM

★ Goal: Co-design virtual memory and newer low power DRAM (such as Partial Array Self-Refresh)
★ Evidence:
   ★ Workloads heterogenous show huge variance in memory demands (Google [SOCC ’12])
★ Problem: OS aggressively uses memory for performance
   ★ Consumes all memory as page cache
   ★ Fragments address space making consolidation difficult
★ How do we re-design OS and DRAM chips to save power?
   ★ Where?: Reliable last level cache interface
   ★ Virtual memory integration: Ensure transparency
   ★ De-fragmentation: Energy-aware page migration
Questions?

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

Thanks to
★ Michael Swift
★ Matt Renzelmann
★ Mahesh Balakrishnan
★ Dahlia Malkhi
★ Vijayan Prabhakaran
★ Ed Nightingale
★ Jeremy Elson
★ James Mickens
★ Rathijit Sen