Tolerating Hardware Device Failures in Software

Asim Kadav, Matthew J. Renzelmann, Michael M. Swift
University of Wisconsin-Madison

THE UNIVERSITY of WISCONSIN MADISON

Current state of OS-hardware interaction

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

```
While (ioread16(ioaddr + Wn7_MasterStatus))
            & 0x8000)
        ;
```

Hardware dependence bug: Device malfunction can crash the system

Current state of OS-hardware interaction

- Hardware dependence bugs across driver classes

```
void hptitop_iop_request_callback(...)     {

arg= readl(...);
  ...
if (readl(&req->result) == IOP_SUCCESS)  {
    arg->result = HPT_IOCTL_OK;
  }
}
```

Highpoint SCSI driver(hptiop.c)

*Code simplified for presentation purposes

How do the hardware bugs manifest?

- Drivers often trust hardware to always work correctly
  - Drivers use device data in critical control and data paths
  - Drivers do not report device malfunctions to system log
  - Drivers do not detect or recover from device failures

An example: Windows servers

- 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 driver:
  - » Crashes reduced from 8% to 3%
- Driver isolation systems not yet deployed

[1] Fault resilient drivers for Longhorn server, May 2004. Microsoft Corp.

Carburizer

- Goal: Tolerate hardware device failures in software through hardware failure detection and recovery

- Static analysis tool - analyze and insert code to:
  - » Detect and fix hardware dependence bugs
  - » Detect and generate missing error reporting information

- Runtime
  - » Handle interrupt failures
  - » Transparently recover from failures

Outline

- Background
- Hardening drivers
- Reporting errors
- Runtime fault tolerance
- Cost of carburizing
- Conclusion

Hardware unreliability

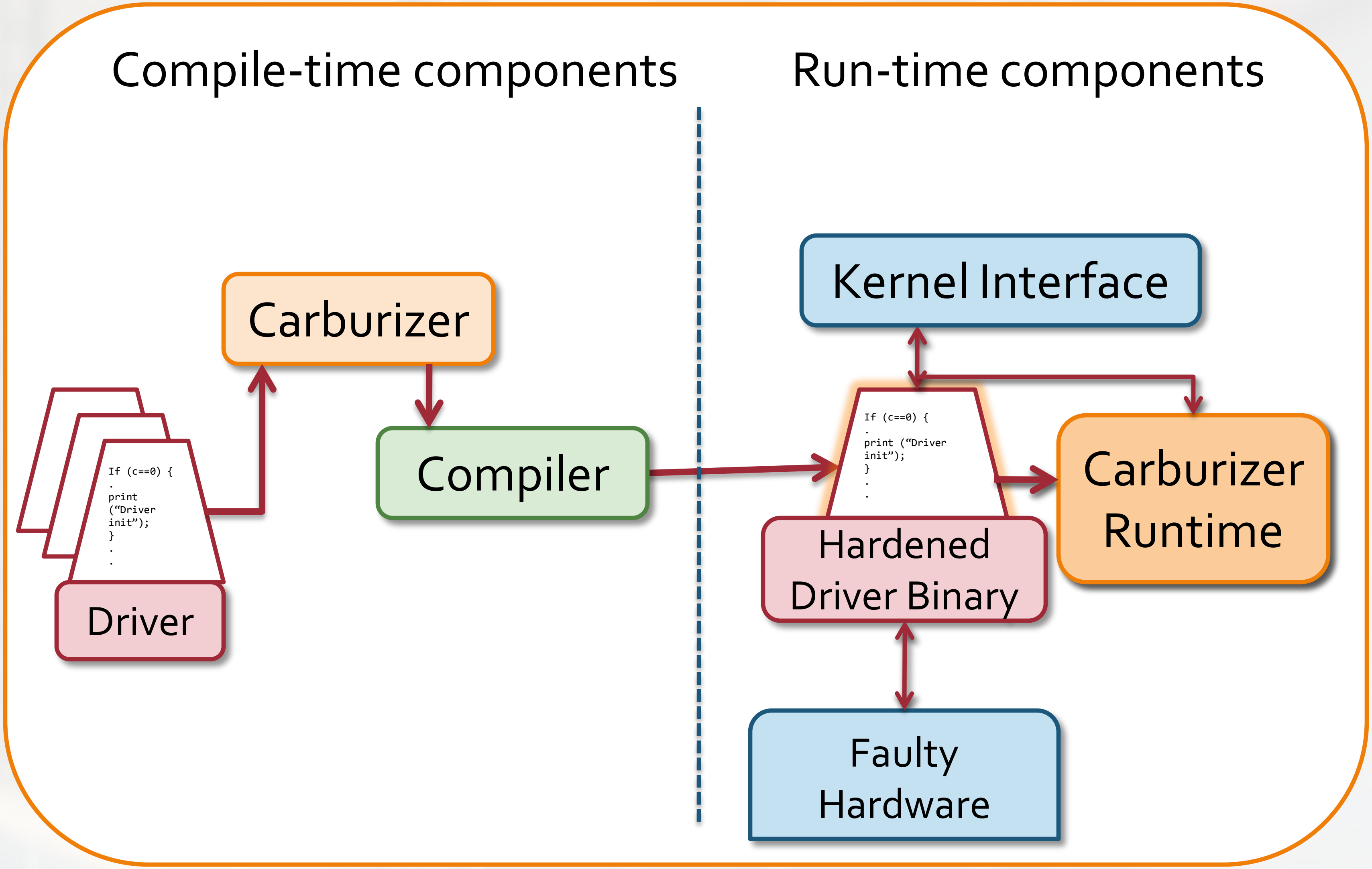
- Sources of hardware misbehavior:
  - » Device wear-out, insufficient burn-in
  - » Bridging faults
  - » Electromagnetic radiation
  - » Firmware bugs
- Result of misbehavior:
  - » Corrupted/stuck-at inputs
  - » Timing errors/unpredictable DMA
  - » Interrupt storms/missing interrupts

Vendor recommendations for driver developers

Recommendation	Summary	Recommended by			
		Intel	Sun	MS	Linux
Validation	Input validation	●	●	●	
	Read once& CRC data	●	●		●
	DMA protection	●	●		
Timing	Infinite polling	●	●	●	
Reporting	Report all failures	●	●	●	
Recovery	Handle all failures		●	●	
	Cleanup correctly	●	●		
	Do not crash on failure	●		●	●
	Wrap I/O memory access	●	●	●	●

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

Carburizer architecture



Outline

- Background
- Hardening drivers
  - Finding sensitive code
  - Repairing code
- Reporting errors
- Runtime fault tolerance
- Cost of carburizing
- Conclusion

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 from
  - » Infinite polling
  - » Unsafe static/dynamic array reference
  - » Unsafe pointer dereferences
  - » System panic calls

Hardening drivers

- Finding sensitive code
  - » First pass: Identify tainted variables

Finding sensitive code

First pass: Identify tainted variables

```
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
```

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

Example: Infinite polling

Finding sensitive code

```
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)

Not all bugs are obvious

```
while (DAC960_PD_StatusAvailableP(ControllerBaseAddress))
{
     DAC960_V1_CommandIdentifier_T CommandIdentifier= DAC960_PD_ReadStatusCommandIdentifier
                                                           (ControllerBaseAddress);
     DAC960_Command_T *Command = Controller ->Commands [CommandIdentifier-1];
     DAC960_V1_CommandMailbox_T *CommandMailbox = &Command->V1.CommandMailbox;
     DAC960_V1_CommandOpcode_T CommandOpcode=CommandMailbox->Common.CommandOpcode;
     Command->V1.CommandStatus =DAC960_PD_ReadStatusRegister(ControllerBaseAddress);
     DAC960_PD_AcknowledgeInterrupt(ControllerBaseAddress);
     DAC960_PD_AcknowledgeStatus(ControllerBaseAddress);
     switch (CommandOpcode)
       {
       case DAC960_V1_Enquiry_Old:
          DAC960_P_To_PD_TranslateReadWriteCommand(CommandMailbox);
          …
}
```

DAC960 Raid Controller(DAC960.c)

Detecting risky uses of tainted variables

- Example II: Unsafe array accesses
  - » Tainted variables used as array index into static or dynamic arrays
  - » Tainted variables used as pointers

Example: Unsafe array accesses

Unsafe array accesses

```
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)

Analysis results over the Linux kernel

- 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

- Additional analyses to detect existing validation code

Analysis results over the Linux kernel

Driver class	Infinite polling	Static array	Dynamic array	Panic calls
net	117	2	21	2
scsi	298	31	22	121
sound	64	1	0	2
video	174	0	22	22
other	381	9	57	32
Total	860	43	89	179

Many cases of poorly written drivers with hardware dependence bugs

Repairing drivers

- Hardware dependence bugs difficult to test
- Carburizer automatically generates repair code
  - » Inserts timeout code for infinite loops
  - » Inserts checks for unsafe array/pointer references
  - » Replaces calls to panic() with recovery service
  - » Triggers generic recovery service on device failure

Carburizer automatically fixes infinite loops

```
timeout = rdstcll(start) + (cpu/khz/HZ)*2;
reg_val = readl(mmio + PHY_ACCESS);
while (reg_val & PHY_CMD_ACTIVE)     {
       reg_val = readl(mmio + PHY_ACCESS);

       if (_cur < timeout)
           rdstcll(_cur);
       else
           __recover_driver();


}
```

Timeout code added

AMD 8111e network driver(amd8111e.c)

*Code simplified for presentation purposes

Carburizer automatically adds bounds checks

```
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));

}
```

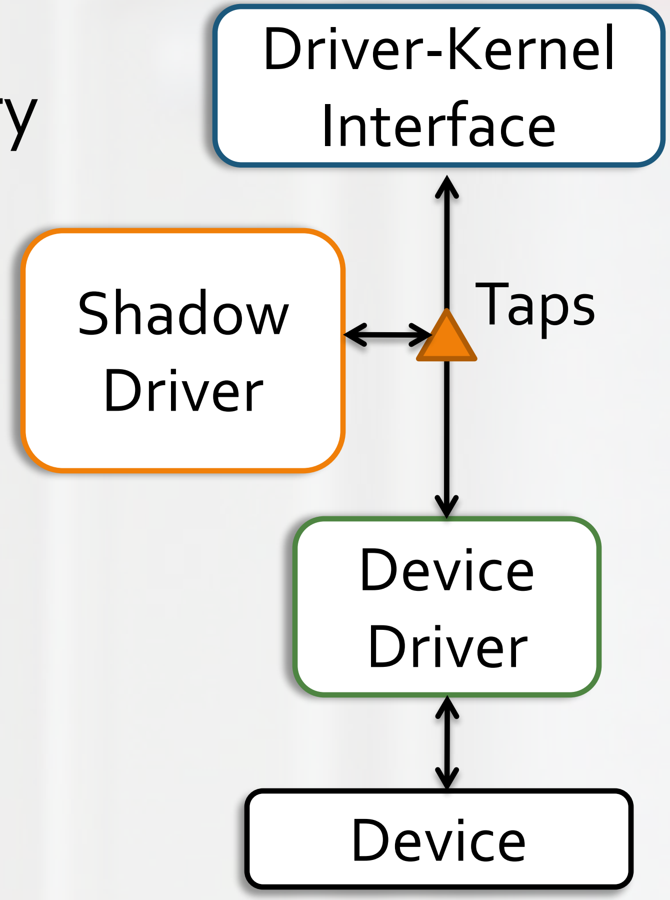
Array bounds check added

Pro Audio Sound driver (pas2_card.c)

*Code simplified for presentation purposes

Runtime fault recovery

- Low cost transparent recovery
  - » Based on shadow drivers
  - » Records state of driver
  - » Transparent restart and state replay on failure
- Independent of any isolation mechanism (like Nooks)



Experimental validation

- Synthetic fault injection on network drivers
- Results

Device/Driver	**Original Driver**		**Carburizer**		
	Behavior	Detection	Behavior	Detection	Recovery
3COM 3C905	CRASH	None	RUNNING	Yes	Yes

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

Outline

- Background
- Hardening drivers
- Reporting errors
- Runtime fault tolerance
- Cost of carburizing
- Conclusion

Reporting errors

- Drivers often fail silently and fail to report device errors
  - » Drivers should proactively report device failures
  - » Fault management systems require these inputs
- Driver already detects failure but does not report them
- Carburizer analysis performs two functions
  - » Detect when there is a device failure
  - » Report unless the driver is already reporting the failure

Detecting driver detected device failures

- Detect code that depends on tainted variables
  - Perform unreported loop timeouts
  - Returns negative error constants
  - Jumps to common cleanup code

```
while (ioread16 (regA) == 0x0f)  {
  if (timeout++ == 200)  {
    sys_report("Device timed out %s.\n", mod_name);
    return (-1);
  }
}
```

Reporting code added by Carburizer

Detecting existing reporting code

Carburizer detects function calls with string arguments

```
static u16 gm_phy_read(...)
{
  ...
 if (__gm_phy_read(...))
   printk(KERN_WARNING "%s: ...\n”, ...);
```

Carburizer detects existing reporting code

SysKonnect network driver(skge.c)

Evaluation

- Manual analysis of drivers of different classes

Driver	Class	Driver detected device failures	Carburizer reported failures
bnx2	network	24	17
mptbase	scsi	28	17
ens1371	sound	10	9

- No false positives

Carburizer *automatically* improves the fault diagnosis capabilities of the system

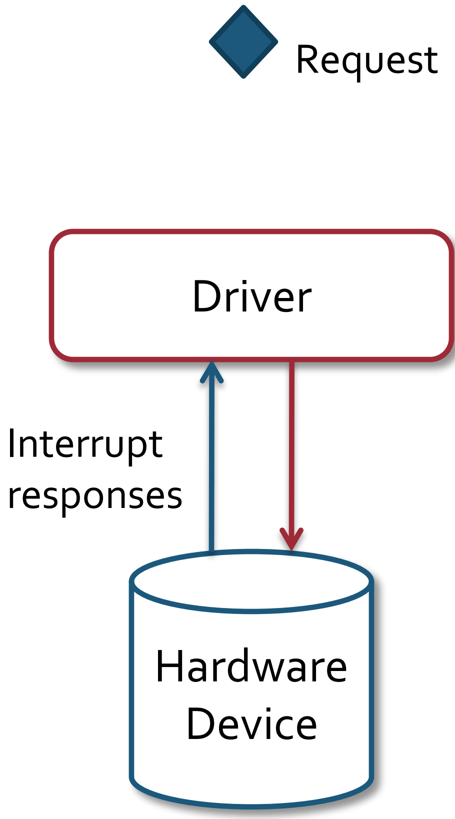
Outline

- Background
- Hardening drivers
- Reporting errors
- Runtime fault tolerance
- Cost of carburizing
- Conclusion

Runtime failure detection

- Static analysis cannot detect all device failures
  - » Missing interrupts: expected but never arrives
  - » Stuck interrupts (interrupts storm): interrupt cleared by driver but continues to be asserted

Tolerating missing interrupts



- Detect when to expect interrupts
  - » Detect driver activity via referenced bits
  - » Invoke ISR when bits referenced but no interrupt activity
- Detect how often to poll
  - » Dynamic polling based on previous invocation result

Tolerating stuck interrupts

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

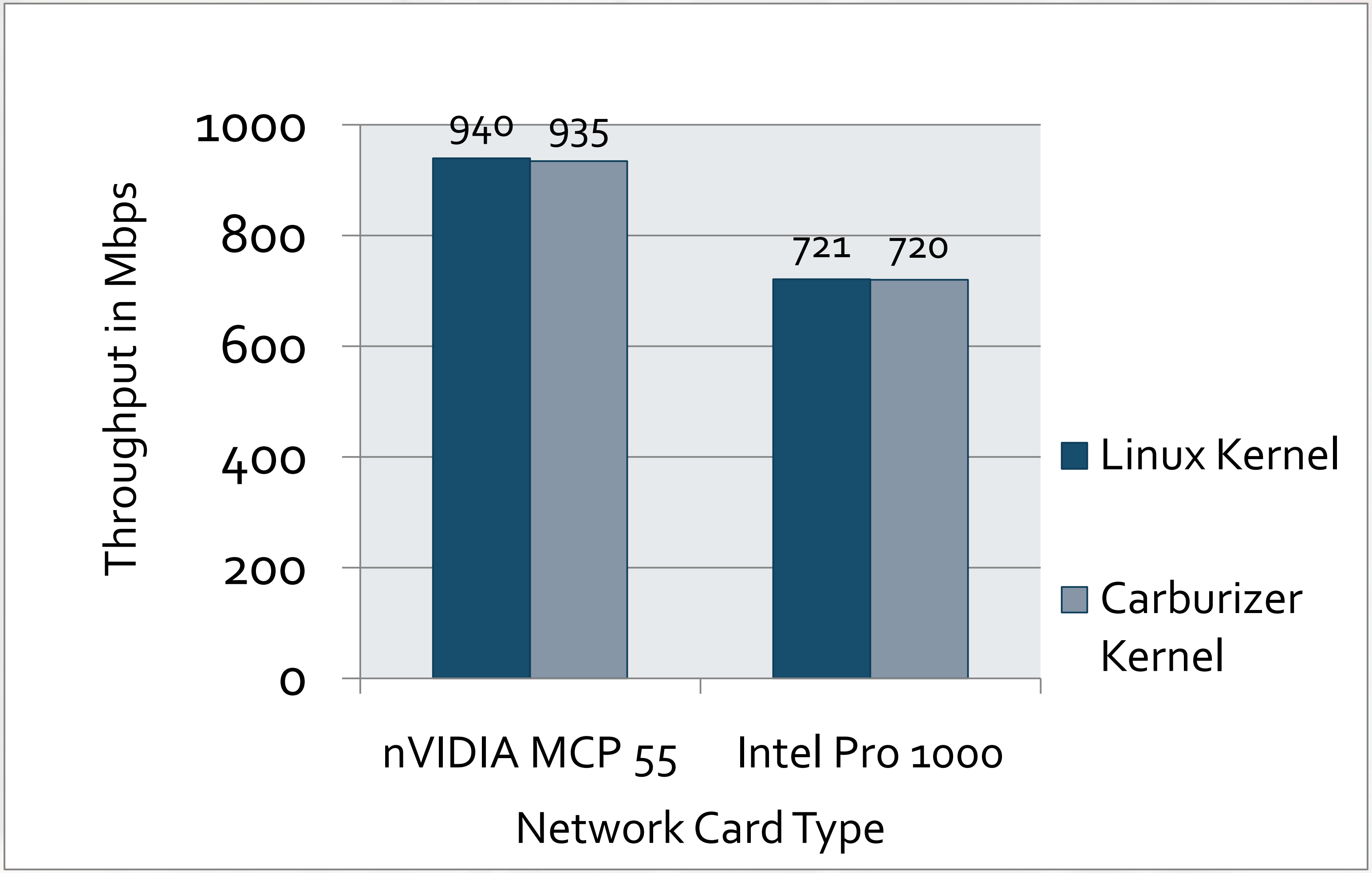
Driver Type	Driver Name	Throughput reduction due to polling
Disk	ide-core,ide-disk, ide-generic	Reduced by 50%
Network	e1000	Reduced from 750 Mb/s to 130 Mb/s
Sound	ens1371	Sounds plays with distortion

Carburizer ensures system and device make forward progress

Outline

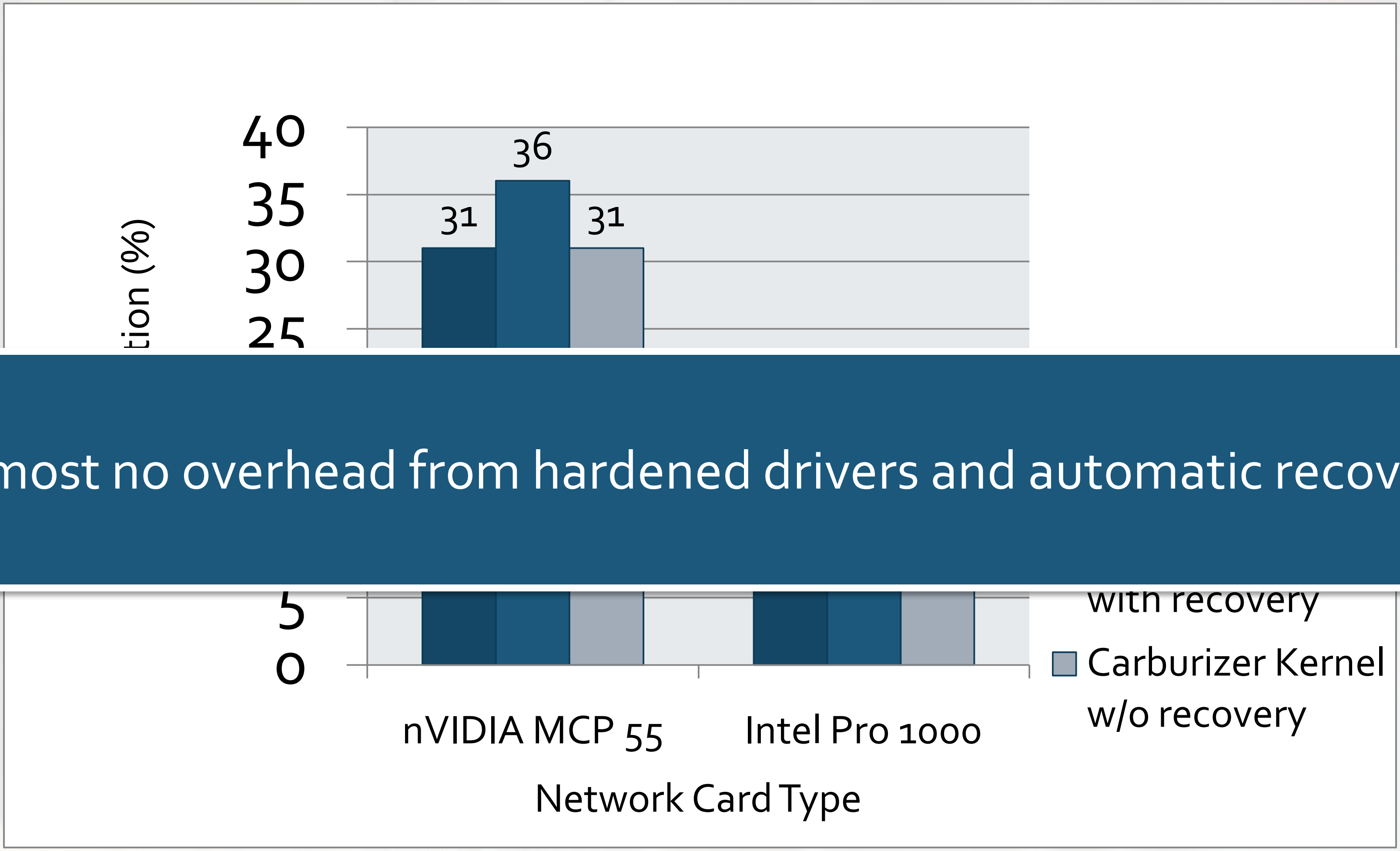
- Background
- Hardening drivers
- Reporting errors
- Runtime fault tolerance
- Cost of carburizing
- Conclusion

Throughput overhead



netperf on 2.2 GHz AMD machines

CPU overhead



Almost no overhead from hardened drivers and automatic recovery

netperf on 2.2 GHz AMD machines

Conclusion

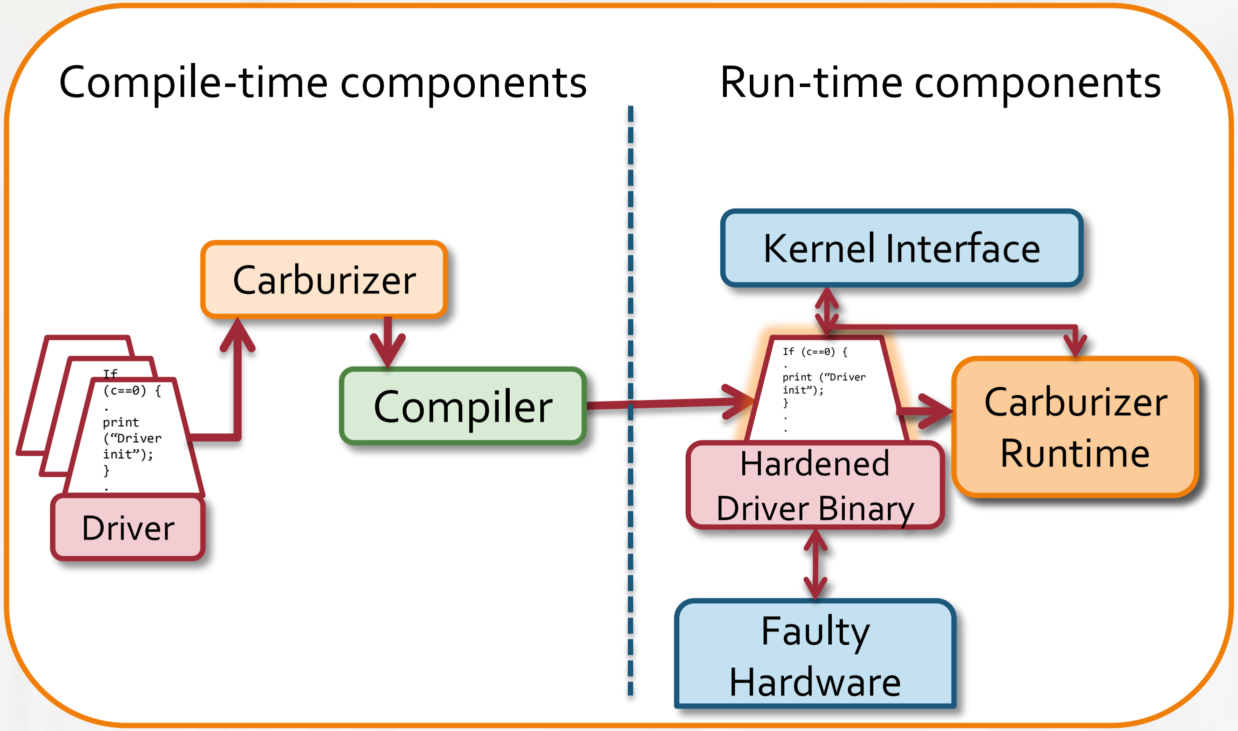
Recommendation	Summary	Recommended by			
		Intel	Sun	MS	Linux
Validation	Input validation	●	●	●	
	Read once& CRC data	●	●		●
	DMA protection	●	●		
Timing	Infinite polling	●	●	●	
	Stuck interrupt		●		
	Lost request			●	
	Avoid excess delay in OS			●	
	Unexpected events	●		●	
Reporting	Report all failures	●	●	●	
Recovery	Handle all failures		●	●	
	Cleanup correctly	●	●		
	Do not crash on failure	●		●	●
	Wrap I/O memory access	●	●	●	●

Conclusion

Recommendation	Summary	Recommended by				Carburizer Ensures
		Intel	Sun	MS	Linux	
Validation	Input validation	●	●	●		●
	Read once& CRC data	●	●		●	
	DMA protection	●	●			
Timing	Infinite polling	●	●	●		●
Reporting	Report all failures	●	●	●		●
Recovery	Handle all failures		●	●		●
	Cleanup correctly	●	●			●
	Do not crash on failure	●		●	●	●
	Wrap I/O memory access	●	●	●	●	

Carburizer improves system reliability by *automatically* ensuring that hardware failures are tolerated in software

Thank You



- Contact
  - » kadav@cs.wisc.edu
- Visit our website for research on drivers
  - » http://cs.wisc.edu/~swift/drivers

Backup slides

Improving analysis accuracy

- Detect existing driver validation code
  - » Track variable taint history
  - » Detect existing timeout code
  - » Detect existing sanity checks

```
while ((inb(nic_base + EN0_ISR) & ENISR_RDC) == 0)
      if (jiffies - dma_start> 2)   {
            ...
            break;
      }
```

```
ne2000 network driver (ne2k-pci.c)
```

Trend of hardware dependence bugs

- Many drivers either had one or two hardware bugs
  - » Developers were mostly careful but forgot in a few places
- Small number of drivers were badly written
  - » Developers did not account H/W dependence; many bugs

Implementation efforts

- Carburizer static analysis tool
  - » 3230 LOC in OCaml
- Carburizer runtime (Interrupt Monitoring)
  - » 1030 lines in C
- Carburizer runtime (Shadow drivers)
  - »19000 LOC in C
  - »~70% wrappers – can be automatically generated by scripts