# Tolerating Hardware Device Failures in Software

Asim Kadav, Matthew J. Renzelmann, and Michael M. Swift

## Introduction

What happens when the device fails while this device driver code executes (drivers/net/3c59x.c)?

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

Unreliable devices are a major source of system downtime. One study indicates that drivers designed with fault tolerance reduced reboot rates due to faulty hardware from 8% to 3%. In addition, transient device failures are common [1, 3, 6] and stem from the sources shown at right. OS driver design guidelines, shown at far right, outline how to harden drivers manually.

**Device Failures → Crashes**

| | |
|---|---|
| Bit flip faults | Wear out |
| Stuck-at faults | Insufficient burn in |
| Bridging faults | Firmware bugs |
| Electromagnetic Interference | |

## Hardware Device Failures

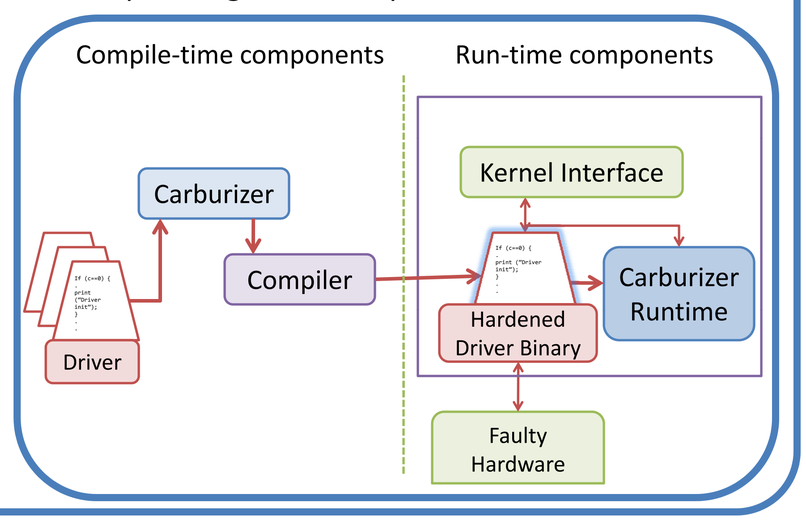
**Driver Design Guidelines [4, 7, 5]**

| | |
|---|---|
| **Validation** | **Reporting** |
| • Input validation | • Report hardware failures |
| • Unrepeatable reads | **Recovery** |
| • DMA protection | • Handle all failures |
| • Data corruption | • Cleanup properly |
| **Timing** | • Conceal failure |
| • Infinite polling | • Do not crash |
| • Stuck interrupts | • Test drivers |
| • Lost request | • Wrap I/O memory access |
| • Excessive delay | |
| • Unexpected events | |

Recommendations addressed by Carburizer are in black

## Carburizer

Carburizer *automatically* implements many driver design guidelines and provides runtime support to recover failed drivers. It relies on static code analysis and runtime support to harden drivers against device failure, and to report bugs to developers.




## Hardening Drivers

Carburizer corrects or reports four categories of bugs in device drivers: infinite polling, invalid array accesses, invalid pointer use, and calls to the kernel's `panic()` routine.

| Bug | Bug description | Carburizer's Resolution |
|---|---|---|
| Infinite Polling | Driver waits indefinitely for hardware device register to change | Insert a timeout to ensure all hardware-dependent loops terminate |
| Invalid array access | Driver uses a potentially-bad value read from the device as an array index | Insert a bounds check if possible and reports the error |
| Invalid pointer use | Driver dereferences a potentially-invalid pointer read from the device | Insert a not-null check and reports the error to developer |
| Calls panic() | Driver crashes the system | Replace calls to `panic()` with a generic driver recovery function |

```
reg_val = readl(mmio + PHY_ACCESS);
while (reg_val & PHY_CMD_ACTIVE)
  reg_val = readl(mmio + PHY_ACCESS);
```

Driver loops forever if the device malfunctions or is disconnected

Carburizer inserts a timeout, and calls the recovery mechanism if the device malfunctions

```
unsigned long long delta = (cpu/khz/HZ)*2;
unsigned long long _start = 0, _cur = 0;
…
timeout = rdstcll(_start) + delta;
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 ();
}
```

```
if ((pas_model = pas_read(0xFF88))) {
  char temp[100];
  …
  sprintf(temp, "%s rev %d",
    pas_model_names[(int) pas_model],
    pas_read(0x2789));  …
```

Driver accesses invalid memory if the device malfunctions

Carburizer calculates the array's size, inserts a bounds check, and calls recovery on failure

```
if ((pas_model = pas_read(0xFF88))) {
  char temp[100];
  …
  if (pas_model < 0 || pas_model >= 5)
    recover_driver();
  sprintf(temp, "%s rev %d",
    pas_model_names[(int) pas_model],
    pas_read(0x2789)); …
```

## Reporting Hardware Failures

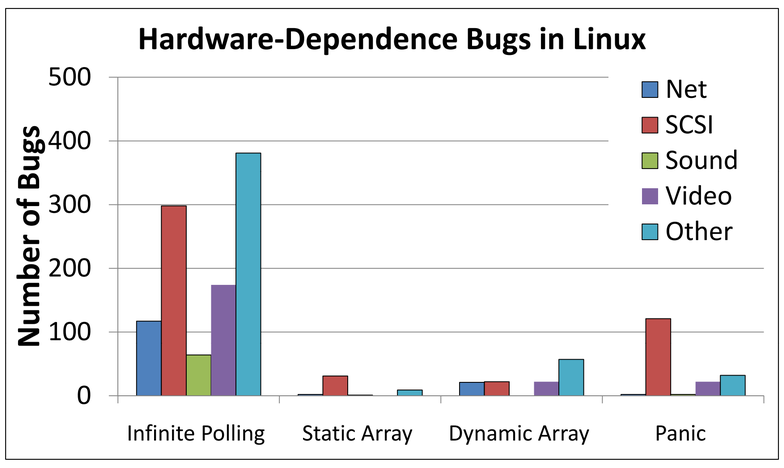
Transient hardware failures often precede permanent failures [6]. Reporting recoverable errors can allow proactive replacement of failure-prone devices. Carburizer locates driver code that detects hardware failures, and ensures the error is reported. If not, Carburizer inserts a reporting routine at the location of each detected hardware malfunction, including device timeouts and conditionally returning negative constants based on values read from the hardware.

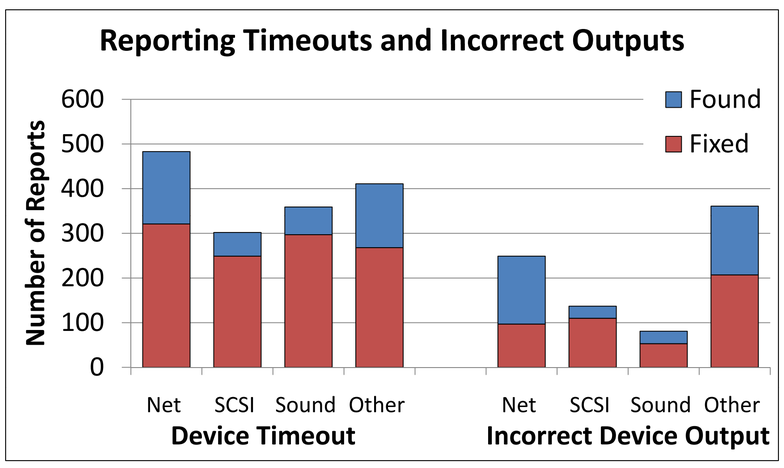
```
while (miicontrol & BMCR_RESET) {
  msleep(10);
  miicontrol = mii_rw(...);
  if (tries++ > 100)
    return -1;
}
```

Driver already detects the timeout, but does not report it

Carburizer automatically reports the timeout to the centralized fault management system

```
while (miicontrol & BMCR_RESET) {
  msleep(10);
  miicontrol = mii_rw(...);
  if (tries++ > 100)  {
    sys_report("...", module_name,
      device_id);
    return -1;
  }
}
```




## Runtime Fault Tolerance

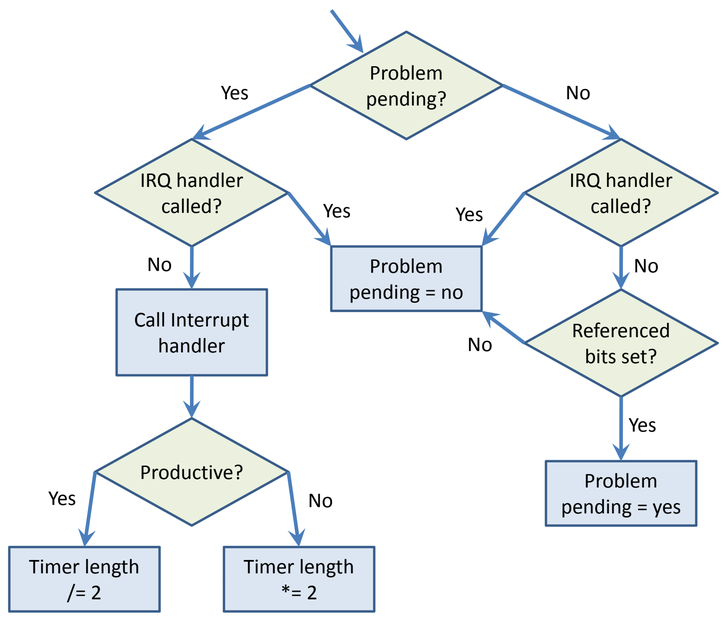
The Carburizer runtime recovery system is based on Shadow Drivers [8]. The runtime also recovers from stuck and missing interrupts.

### Dynamic Polling

Normally, devices generate interrupts in response to requests made of the driver. If interrupt activity stops, but requests continue, the runtime polls the interrupt handler. The runtime uses its return value to determine whether the call was productive. The polling rate doubles if the call was productive; otherwise, it halves. The runtime corrects interrupt storms by disabling the IRQ line and using the same polling mechanism.

### Driver Activity via Referenced Bits

The runtime uses the referenced bits on the pages containing the driver's code to determine when driver requests are made. If referenced bits are set, the driver has been invoked, so an interrupt is expected. If the interrupt handler is not called, it may indicate a missing interrupt. The following flow chart outlines the polling algorithm.





### References

[1] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. Failstutter fault tolerance. In *Proc. of the Eighth IEEE HOTOS,* May 2001.
[2] S. Arthur. Fault resilient drivers for Longhorn server, May 2004. Microsoft Corporation, WinHec 2004 Presentation DW04012.
[3] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An Analysis of Latent Sector Errors in Disk Drives. In *Proc. of the 7th SIGMETRICS, June 2007.*
[4] S. Graham. Writing drivers for reliability, robustness and fault tolerant systems. http://www.microsoft.com/whdc/archive/FTdrv.mspx, Apr. 2004.
[5] Intel Corporation and IBM Corporation. Device driver hardening design specification draft release 0.5h. http://hardeneddrivers.sourceforge.net/downloads/DDH-Spec-0.5h.pdf, Aug. 2002.
[6] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *Proc. of the 5th FAST, 2007.*
[7] Sun Microsystems. Solaris Express Software Developer Collection: Writing Device Drivers, chapter 13: Hardening Solaris Drivers. Sun Microsystems, 2007.
[8] M. Swift, M. Annamalau, B. N. Bershad, and H. M. Levy. Recovering device drivers. ACM Transactions on Computer Systems, 24(4), Nov. 2006.