Nooks

Robert Grimm
New York University

The Three Questions

- What is the problem?
- What is new or different?
- What are the contributions and limitations?

Design and Implementation

Nooks Overview

- An isolation and recovery service
  - Manages kernel-space extensions
  - Targeted at commodity kernels
    - Implemented in Linux, should be easily portable to other OS's
  - Detects, removes, and restarts misbehaving extensions
    - But not malicious ones
  - With shadow drivers, also hides recovery from applications

Why Safe Extensions for Today's Commodity Kernels?

- Cost of failures continues to rise
  - Downtime of mission-critical systems
  - Staffing for help-desk
- Extensions are common-place
  - 70% of Linux code
  - 35,000 different drivers with 120,000 versions for Windows XP
- Extensions are leading cause of failures
  - 85% of failures for Windows XP
  - 7 times more bugs in drivers than rest of kernel for Linux

Why Not Use *X*?

- Capabilities, segments
  - Need specialized hardware, no support for recovery
- Micro-, pico-, exo-kernels
  - No support for recovery, some performance concerns
- Transactions
  - Slooooooooooooooowwwwwwwwwww
- Type-safe languages and runtimes
  - Not backwards compatible
- Software fault isolation
  - No support for recovery

Why Not Use *X*? (cont.)

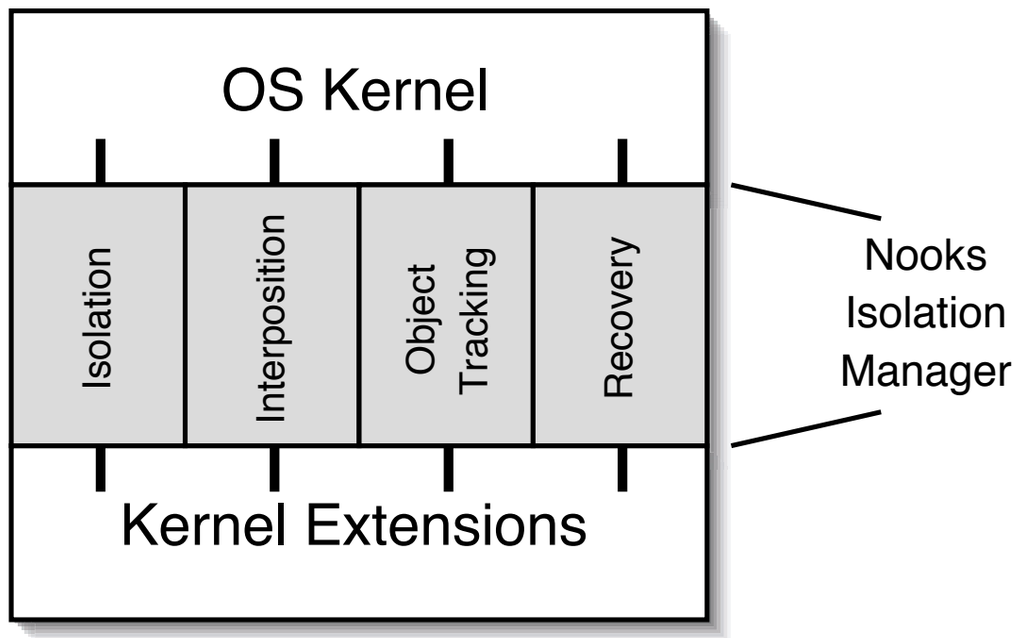
- Virtual machines
  - Still have drivers in VMM
- Insight: Virtualize interface between kernel and extensions but not harware
  - I.e., we don't need to be perfect, just good enough

Nooks Principles and Goals

- Two principles
  - Design for fault resistance, not fault tolerance
  - Design for mistakes, not abuse
- Three goals
  - Isolation
  - Recovery
  - Backwards compatibility

Nooks Functionality

- Isolation
  - Lightweight protection domains for extensions
  - Extension procedure call (XPC)
- Interposition
  - Wrappers for all kernel / extension crossings
    - Manage control and data flow
- Object-tracking
  - List of kernel data structures modified by extension
- Recovery
  - Removal and restarting of extensions



Nooks Implementation

- Additional layer for Linux 2.4.18
  - Same privilege for all code (ring 0)
  - Memory protection through page tables

Source Components	**# Lines**
Memory Management	1,882
Object Tracking	1,454
Extension Procedure Call	770
Wrappers	14,396
Recovery	1,136
Linux Kernel Changes	924
Miscellaneous	2,074
Total number of lines of code	22,266

Compared to
2.4 million lines
in the Linux kernel

Isolation

- Lightweight protection domains
  - Private memory structures for each extension
    - Heap, stacks, memory-mapped I/O regions, buffers
  - Different page tables for kernel and each extension (why?)
    - Kernel can read and write all memory
    - Each extension can only write its own memory
- XPC
  - Saves caller's context, finds stack, changes page tables
  - May be deferred
    - Amortize cost over several logical transfers

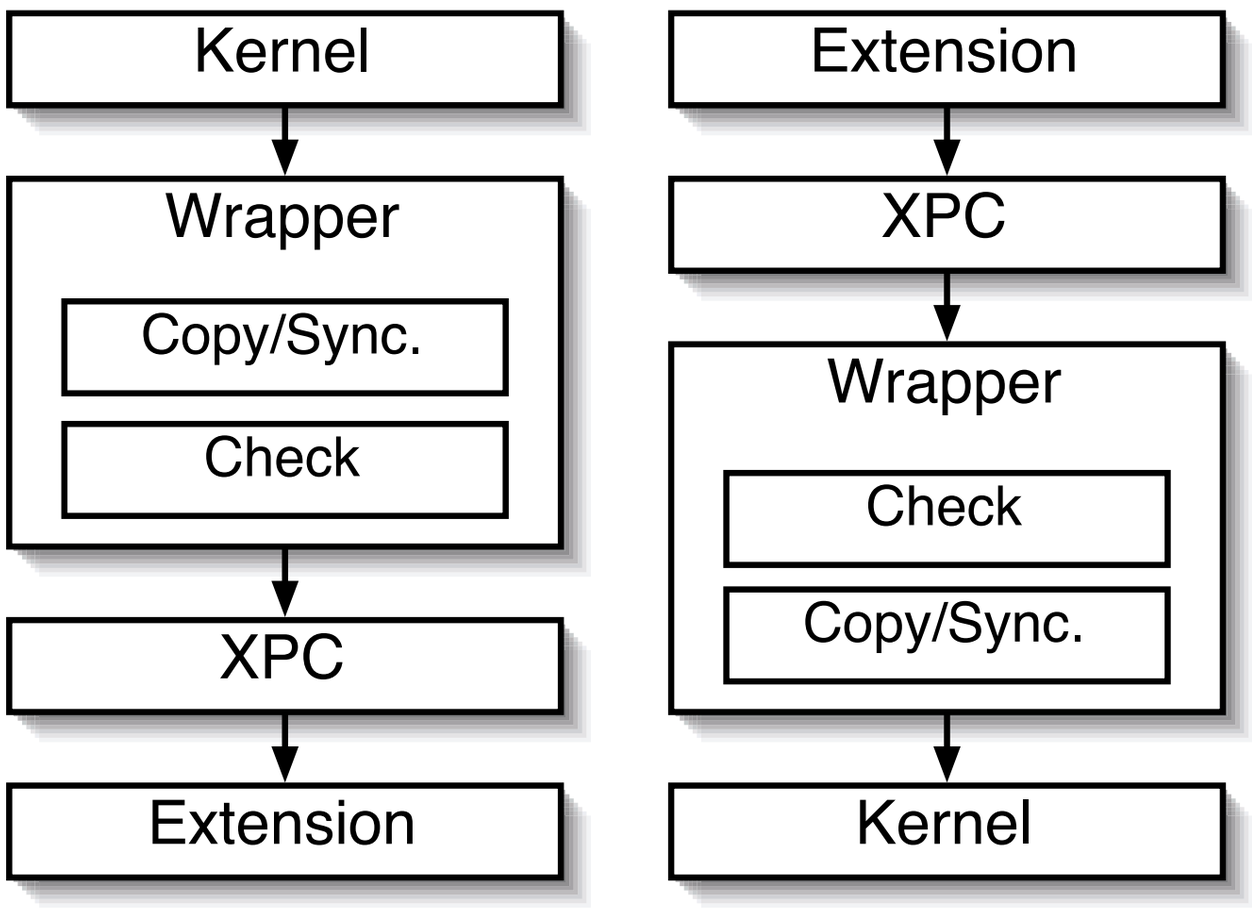
Interposition and Wrappers

- How to interpose?
  - Explicitly interpose on extension initialization call
  - Replace function pointers with wrapped versions
- What about kernel objects (i.e., data structures)?
  - Some are read only ➡ done
  - Some are written by extensions
    - Non-performance-critical updates through XPC
    - Performance-critical updates on shadow copy, synchronized through a deferred XPC on next regular XPC
      - Call-by-*what*?

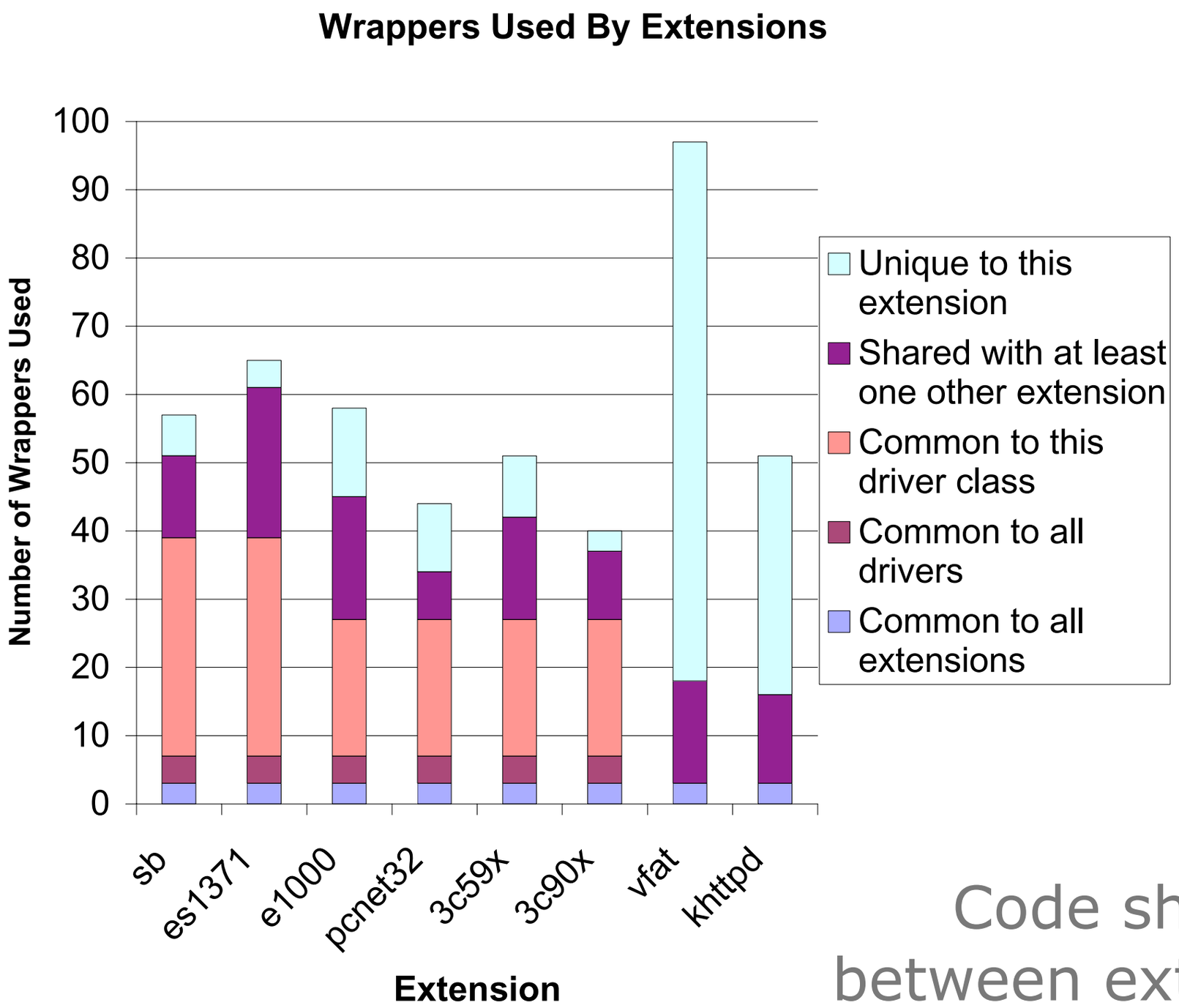
More on Wrappers

- Check parameters for validity
- Implement call-by-value-result for kernel objects
- Perform XPC

- Skeleton generated by tool
- Body written by hand



Even More on Wrappers



Code shared between extensions!

Object Tracker

- Currently supports 43 types
  - E.g., tasklets, PCI devices, inodes
- Records addresses of all objects
  - If object used for one XPC, table attached to task structure
  - If object used across several XPCs, hash table
- Keeps mapping between kernel and extension versions
- Tracks object lifetimes
  - Single XPC call
  - Explicit allocation and deallocation
  - Semantics of object (e.g., timer data structure)

Recovery

- Triggered by
  - Parameter validation, livelock detection, exceptions, signals
- Performed by
  - Recovery manager
    - Cleans up after extension
  - User-mode agent
    - Determines recovery policy
- Broken into several stages
  - Disable interrupts, unwind tasks, release resources, unload extension, reload and init extension, re-enable interrupts

Limitations

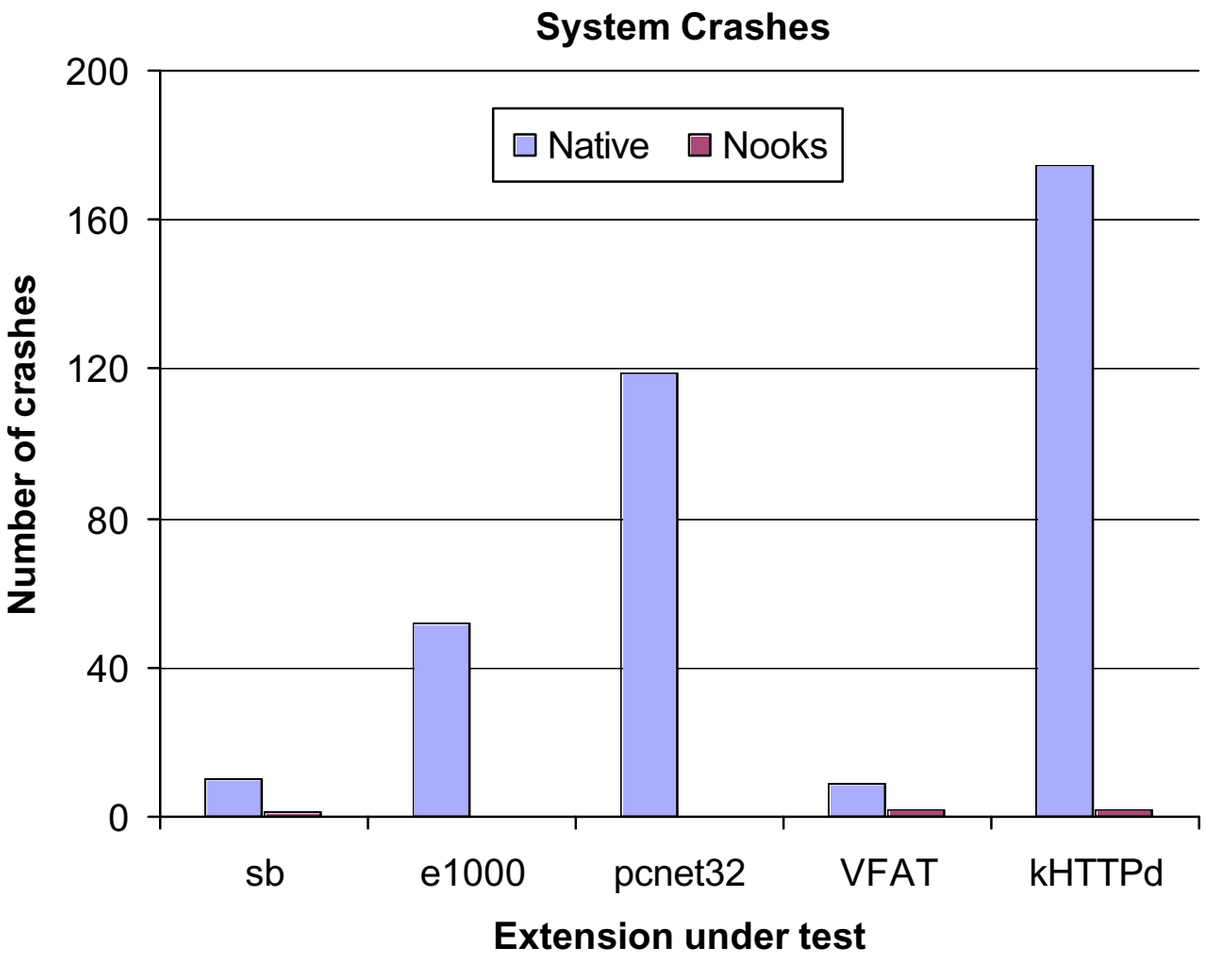
- Extensions run in kernel mode
  - May execute privileged instructions
  - May loop forever (but Nooks detects livelock)
- Parameter checking is incomplete
- Recovery safe only for dynamically loaded extensions

Evaluation

Evaluation Criteria

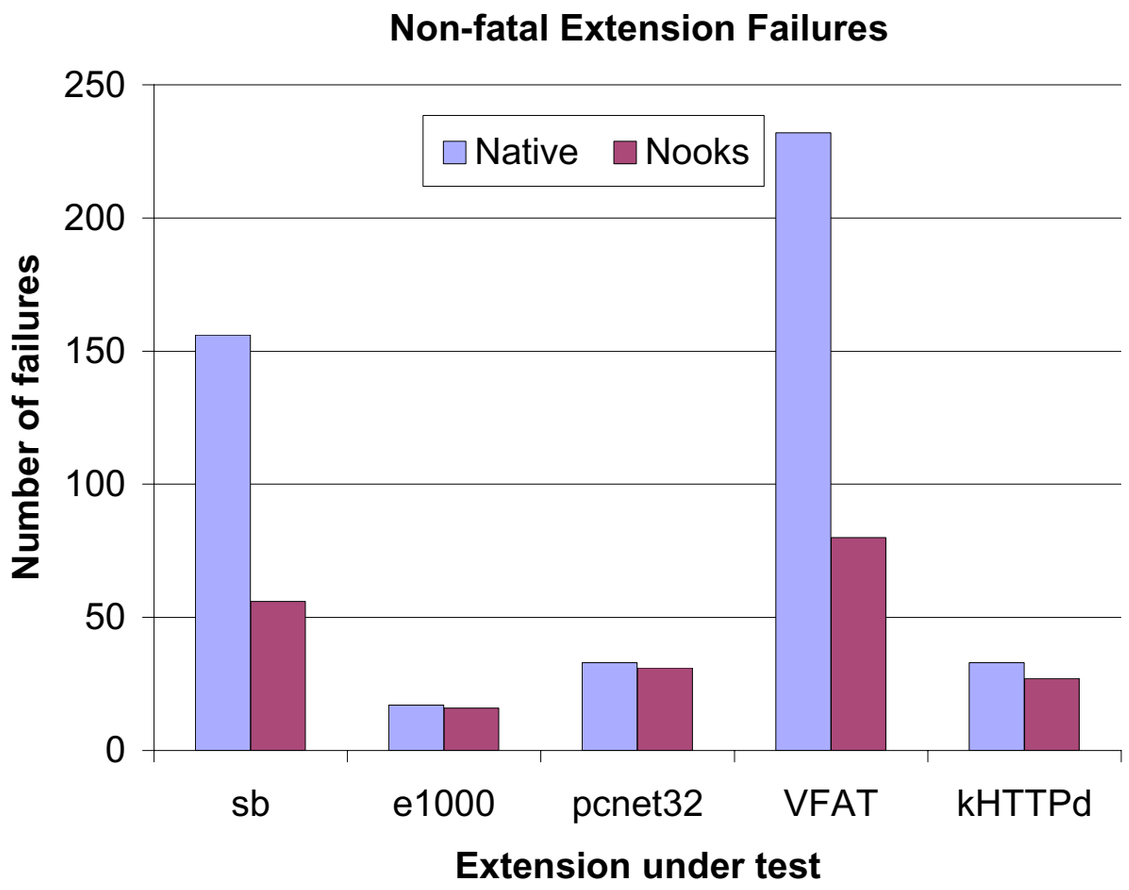
- The two eff's
  - Effectiveness (reliability)
    - Inject faults into extensions
      - By hand
      - Automatically based on common bugs
  - Efficiency (performance)
    - Measure latency/throughput with and w/o Nooks but exactly same code (why?)

Effectiveness



Nooks recovers form 99% of system crashes

But catches only a fraction of non-fatal failures

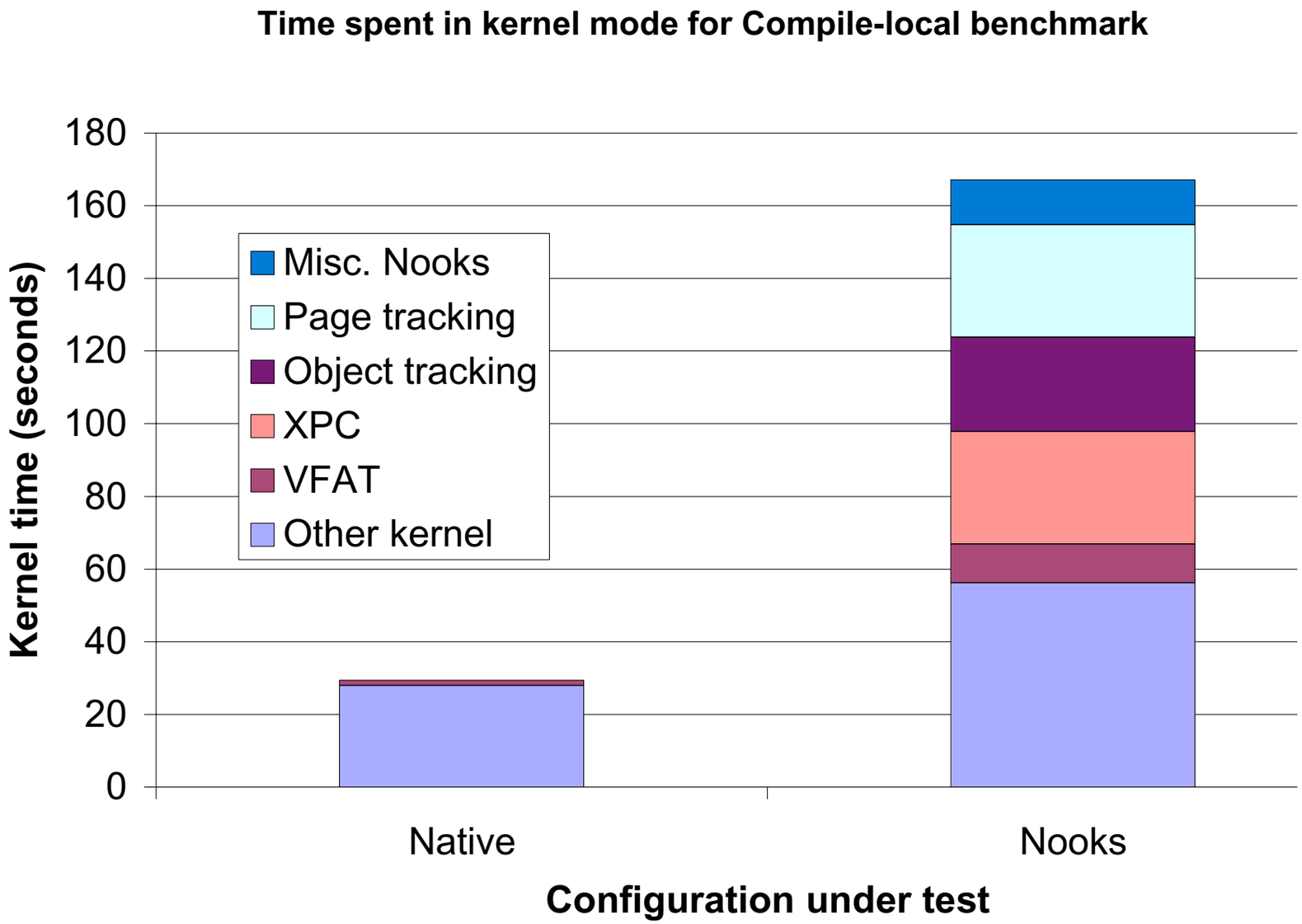


Efficiency

Benchmark	**Extension**	*XPC Rate (per sec)*	*Nooks Relative Performance*	*Native CPU Util. (%)*	*Nooks CPU Util. (%)*
Play-mp3	sb	150	1	4.8	4.6
Receive-stream	e1000 (receiver)	8,923	0.92	15.2	15.5
Send-stream	e1000 (sender)	60,352	0.91	21.4	39.3
Compile-local	VFAT	22,653	0.78	97.5	96.8
Serve-simple-web-page	kHTTPd (server)	61,183	0.44	96.6	96.8
Serve-complex-web-page	e1000 (server)	1,960	0.97	90.5	92.6

- XPC rate serves as performance indicator
  - Three broad categories

Efficiency (cont.)



- There's more code to run
- The code runs more slowly

All Is Good, Is It?

Problem and Requirements

- Kernel recovers driver (most of the time)
- But applications using failed driver still crash
  - Need to conceal crash and recovery from applications
  - Need to centralize *generic* recovery logic
  - Need to ensure low overhead

Drivers in Action

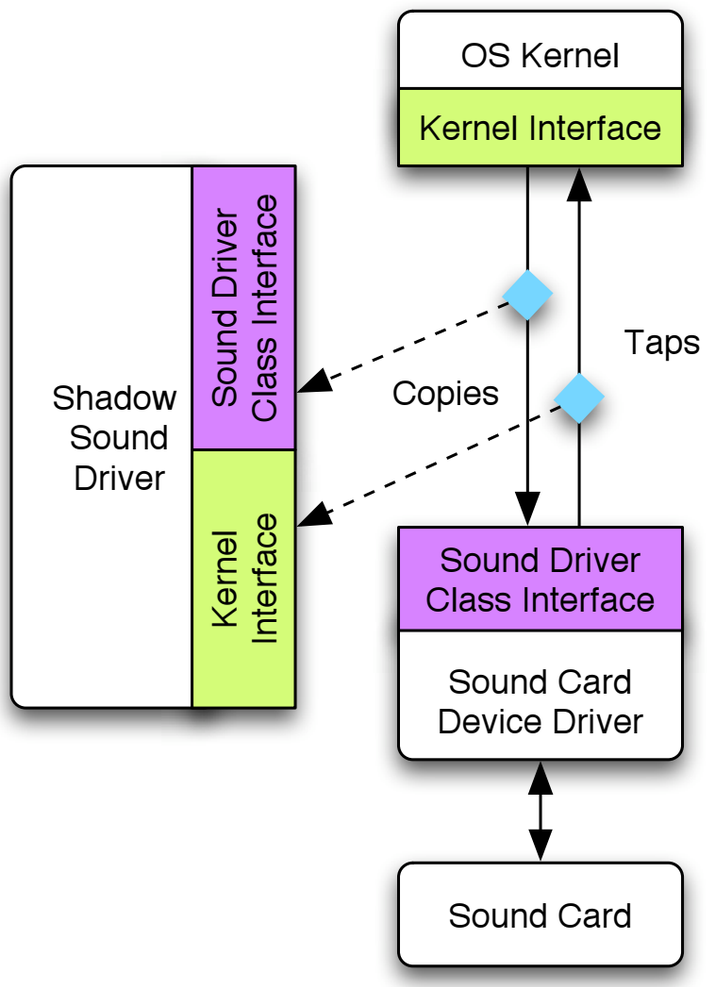
- Organized into classes
- Handle requests
  - To access hardware
  - To change configuration
- May crash
  - Due to request stream, hardware interaction, kernel environment
  - Either deterministically or transiently
  - Either right away (fail-stop) or some time later

Enter Shadow Drivers

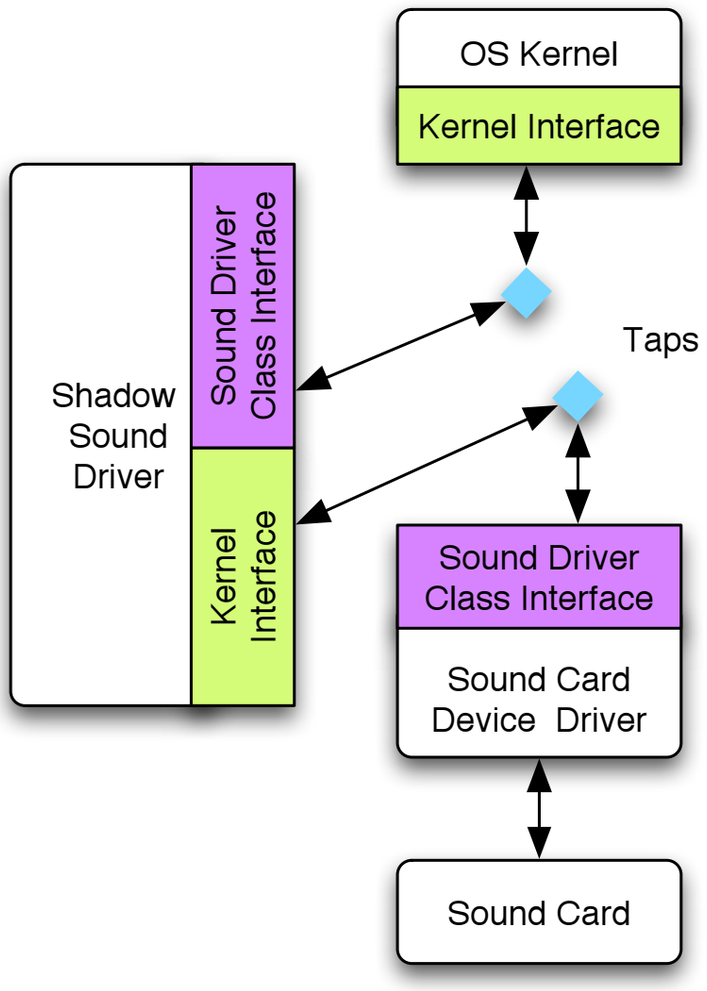
- Conceal transient *and* fail-stop failures for *entire* class of drivers
- Monitor driver in passive mode
  - Replicate procedure calls
  - Log requests and responses
- Impersonate driver/kernel in active mode
  - Respond to kernel requests
  - Respond to driver requests during re-initialization
  - Restore state in driver

Enabling Mechanism

- Taps: T-junction between kernel and drivers
  - Requires ability to interpose on all communications



Passive mode



Active mode

Implementation

- General infrastructure
  - Isolation service to prevent kernel corruption
  - Redirection service to implement taps
  - Object tracking service to facilitate recovery
- Luckily, we got Nooks
  - Lightweight protection domains, wrappers, object tracking

Passive Monitoring

- Track all requests
  - Either: state of each active connection
  - Or: log of pending commands
- Record configuration parameters
  - Log `ioctl` calls
- Track all kernel objects (used by driver)

- In reality: many calls require *no* work

Recovery: Stop Failed Driver

- Disable execution of driver (e.g., any tasks in driver)
- Disable hardware device (e.g., interrupts)
  - Also remove I/O mappings (for memory mapped I/O)
- Garbage collect resources held by driver
  - However, not those used by kernel to request driver services

Recovery: Re-initialization

- Goal: reboot driver from clean slate
  - Need to keep copy of device driver's clean data section
    - No need to access disk, whose driver may have crashed
    - What about driver's code?
- Re-initialization in action
  - Initialize driver's internal state
  - Repeat kernel's initialization sequence
  - Reattach driver to pre-failure kernel resources

Recovery: Transfer State

- Goal: Restore driver to just before time of failure
- Restore configuration state
  - Connections and `ioctl`'s, depending on class
- Handle outstanding requests as well as new arrivals
  - Strategy depends on driver class
    - Disk and network devices, drivers, and kernel stack tolerate duplicate requests → reissue requests
    - Printers do not like duplicates → drop requests

Recovery: Proxy Requests

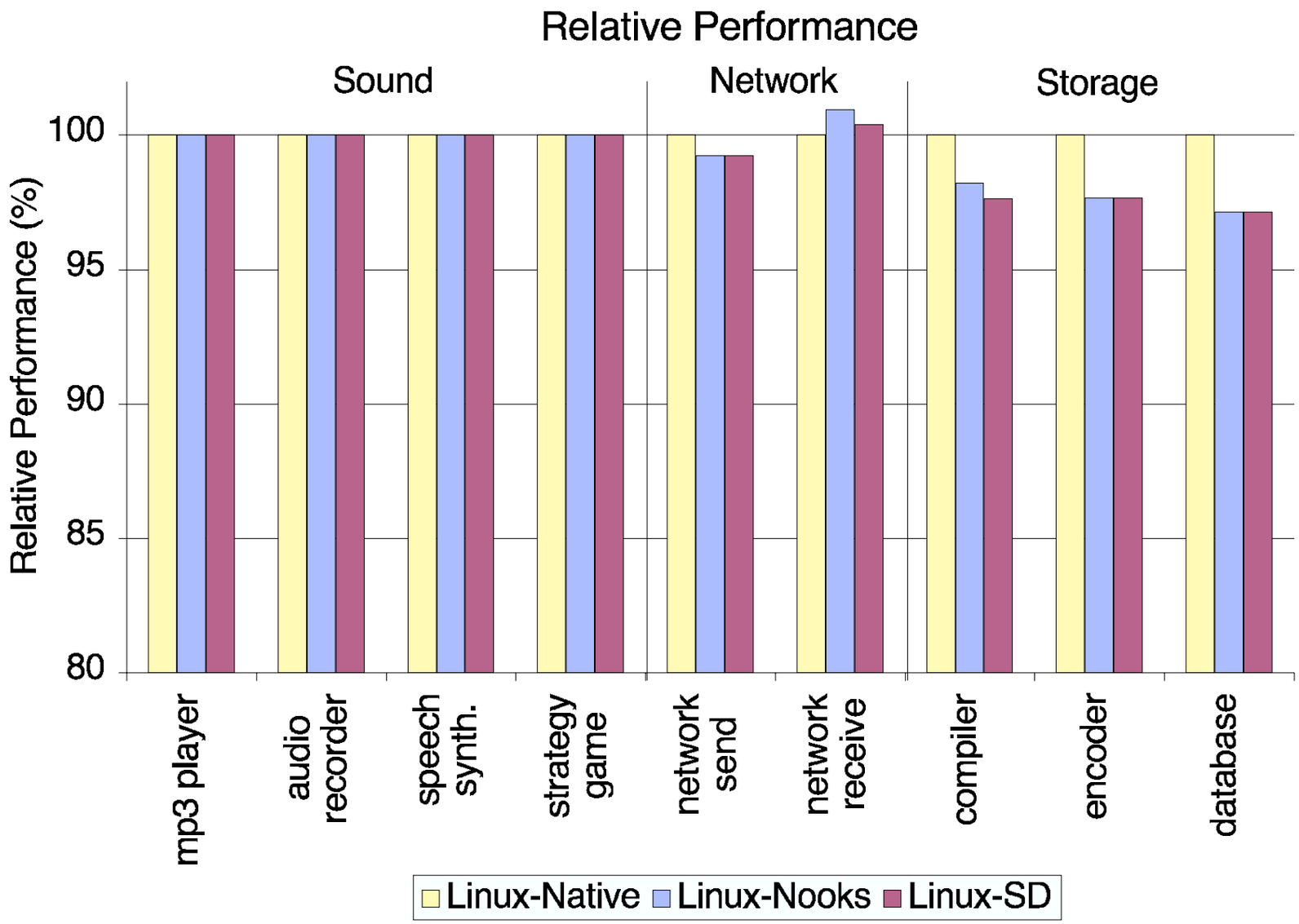
- Responses depend on driver/request semantics:
  - Respond with recorded information
  - Silently drop request
  - Queue request for later processing
  - Block request until recovery is complete
  - Report driver as busy
    - "Please call again during regular business hours"
- Implementation depends on interface spec but not drivers themselves

Another Paper, Another Evaluation

Criteria

- Performance: what is the overhead of shadow drivers?
- Fault tolerance: can applications continue to run?
- Limitations: how realistic is fail-stop assumption?
- Code size: how hard is it to implement?
  - Not hard: 700 lines for sound, 200 for network, 300 for IDE

Performance



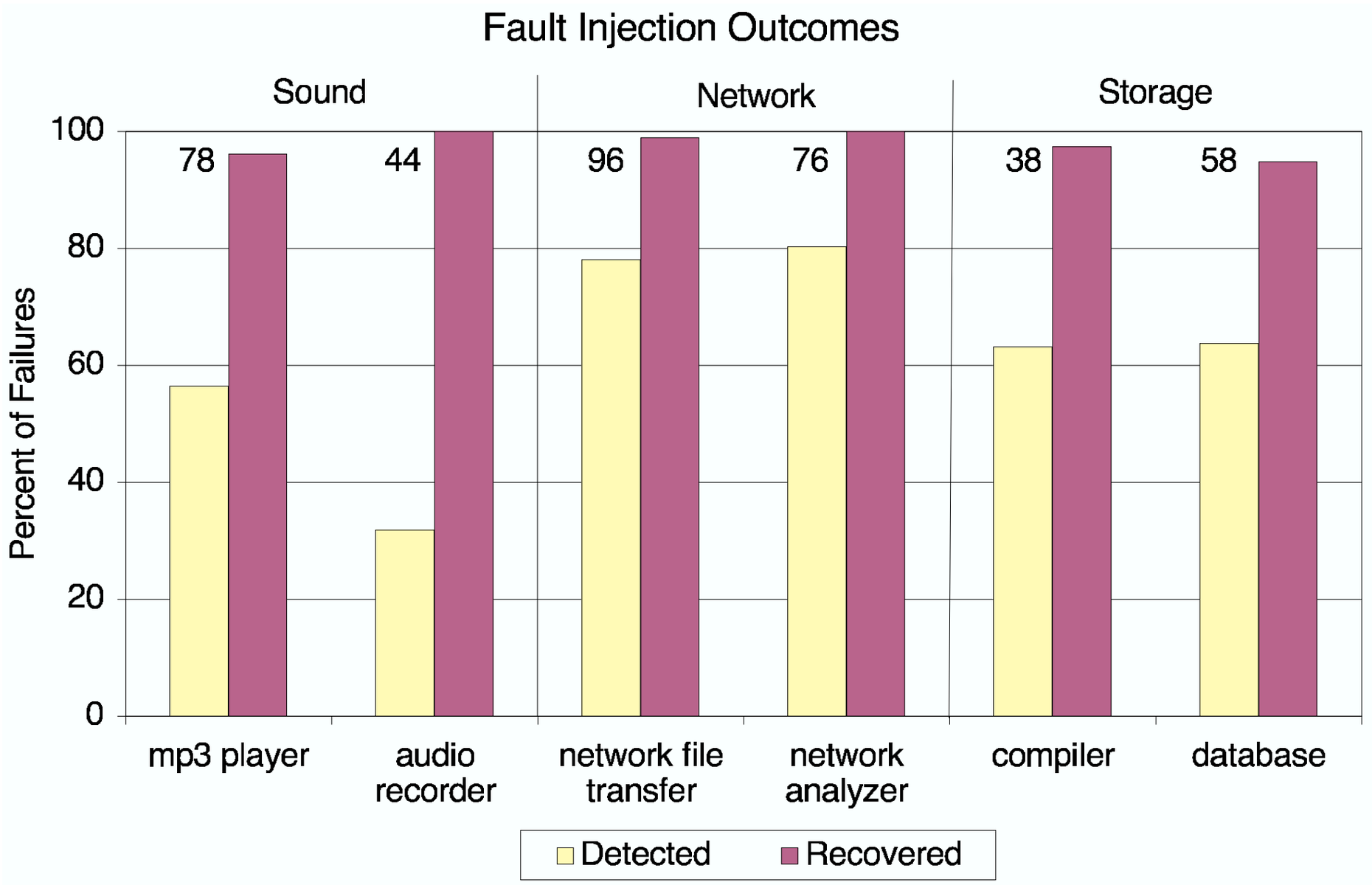
* My slightly pessimistic take: Nooks already is so slow that shadowing doesn't matter

Fault-Tolerance

Device Driver	Application Activity	Application Behavior		
		Linux-Native	Linux-Nooks	Linux-SD
Sound (audigy driver)	mp3 player	CRASH	MALFUNCTION	√
	audio recorder	CRASH	MALFUNCTION	√
	speech synthesizer	CRASH	√	√
	strategy game	CRASH	MALFUNCTION	√
Network (e1000 driver)	network file transfer	CRASH	√	√
	remote window manager	CRASH	√	√
	network analyzer	CRASH	MALFUNCTION	√
IDE (ide-disk driver)	compiler	CRASH	CRASH	√
	encoder	CRASH	CRASH	√
	database	CRASH	CRASH	√

- Why does Nooks crash for IDE driver crashes?
- Why do speech synthesizer and some network tools continue to function with Nooks?

Limits to Recovery



* Automatic detection is incomplete, but recovery really good

Pulling Back

Some Issues

- If only we had a software/tagged TLB...
- What about end-to-end benchmarking?
  - All/most drivers managed by Nooks
  - Typical application mix
- How many wrappers is enough?
- How general is Nooks?
  - Supports only one communication pattern
    - Kernel / extension, but not between extensions
- What about deterministic faults?

What Do You Think?