Rx: Treating Bugs As Allergies – A Safe Method to Survive Software Failures

by Feng Qin, Joseph Tucek, Jagadeesan Sundaresan and Yuanyuan Zhou

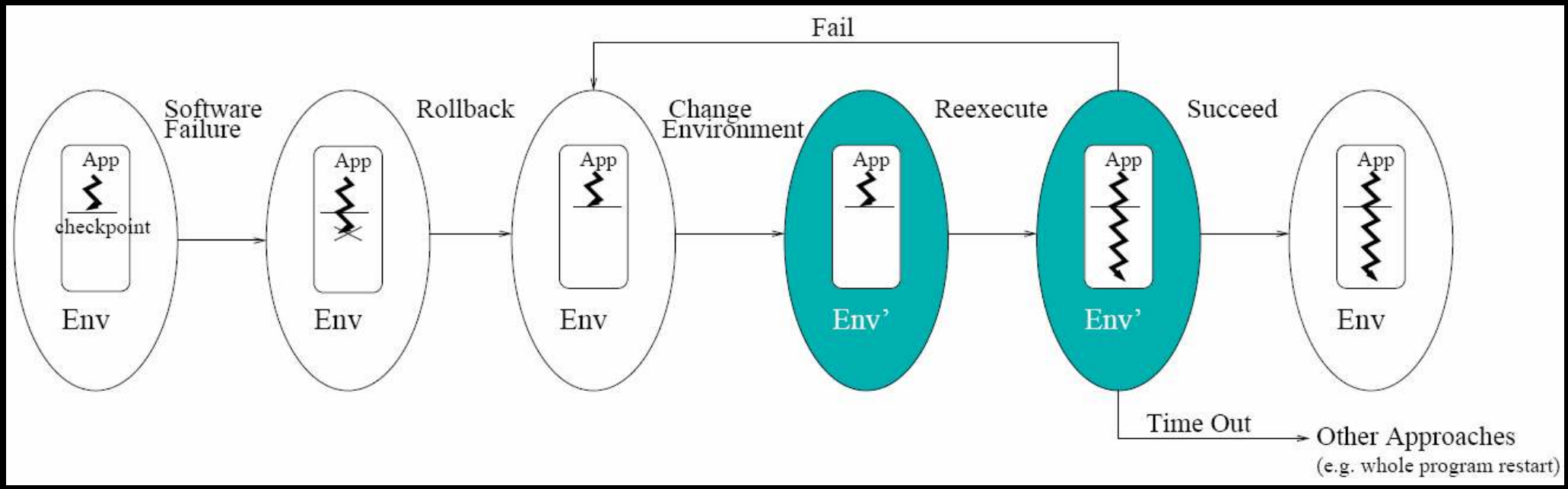
ACM Symposium on Operating Systems Principles (SOSP) 2005

presented by Johannes Pletzer

Software System Seminar 2007, Department of Computer Sciences, University of Salzburg

Main Idea

- Rx recovers software from bugs at runtime
- Upon failure: Rollback and reexecute in a modified environment
- Similarity to allergies in real life



Execution Environment

- Three levels:
  - Hardware
  - OS Kernel
  - Libraries
- Broad definition
- Studies show a lot of bugs depend on execution environment

Environmental Changes

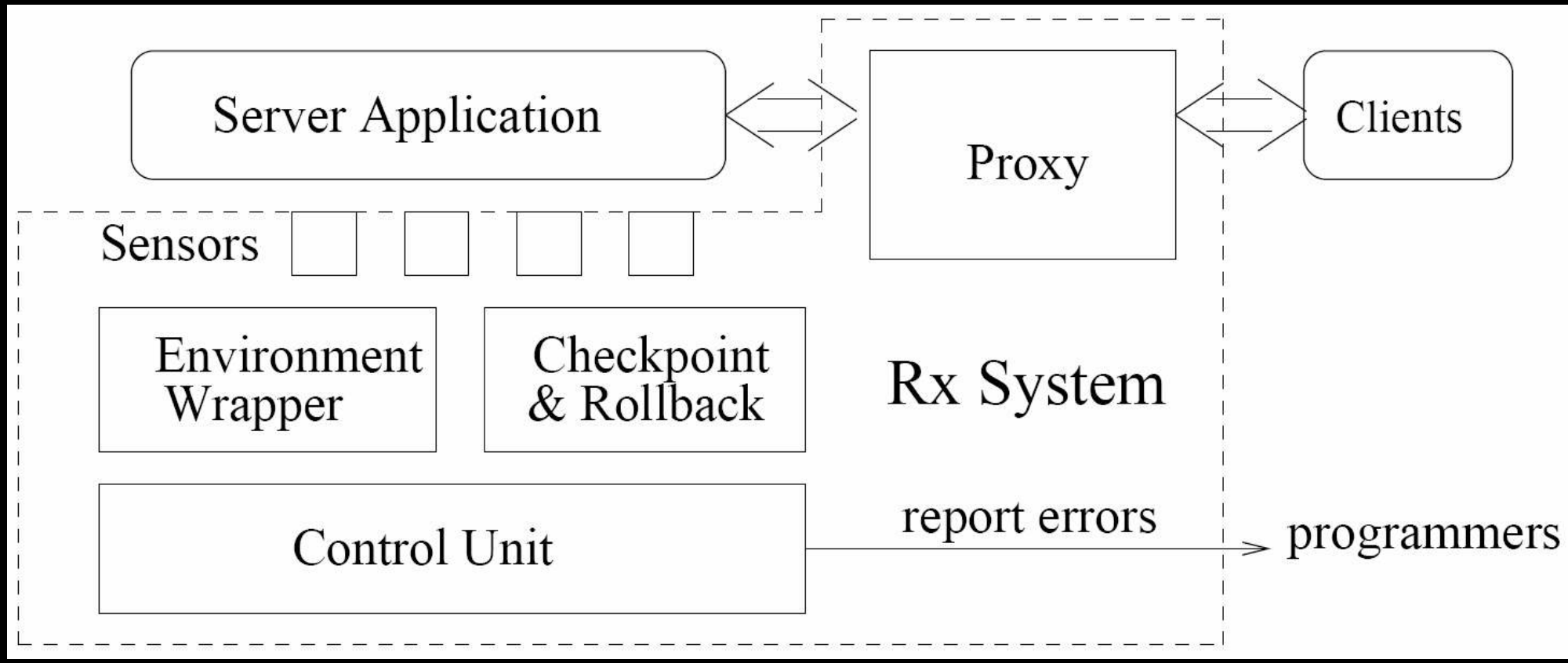
Category	Environmental Changes	Potentially-Avoided Bugs	Deterministic?
Memory Management	delayed recycling of freed buffer	double free, dangling pointer	YES
	padding allocated memory blocks	dynamic buffer overflow	YES
	allocating memory in an alternate location	memory corruption	YES
	zero-filling newly allocated memory buffers	uninitialized read	YES
Asynchronous	scheduling	data race	NO
	signal delivery	data race	NO
	message reordering	data race	NO
User-Related	dropping user requests	bugs related to the dropped request	Depends

- Safe to apply, e.g. memory interface spec.

Rx Advantages

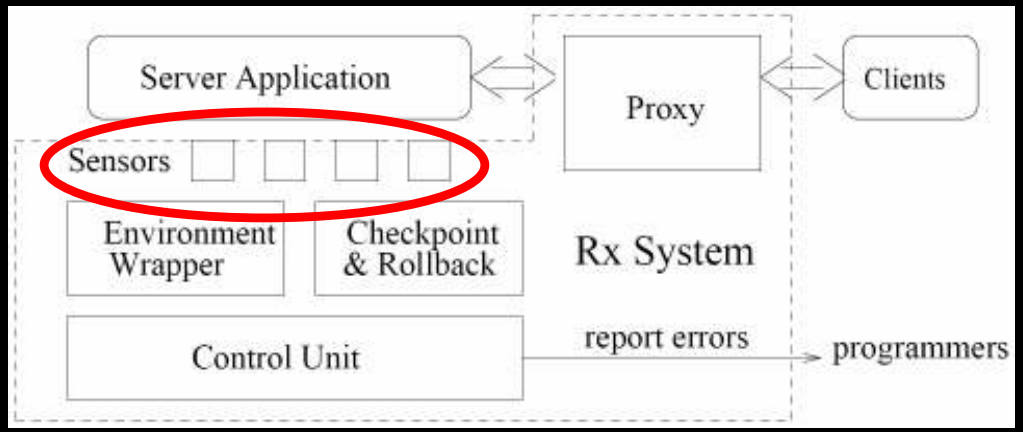
- Comprehensive
  survives a lot of (non-) deterministic bugs
- Safe
  no speculative fixes
- Noninvasive
  little to no modifications in applications required
- Efficient
  program restart approaches are much slower
- Informative
  additional information for programmers

Rx Design


Server Application
Proxy
Clients
Sensors
Environment Wrapper
Checkpoint & Rollback
Rx System
Control Unit
report errors
programmers

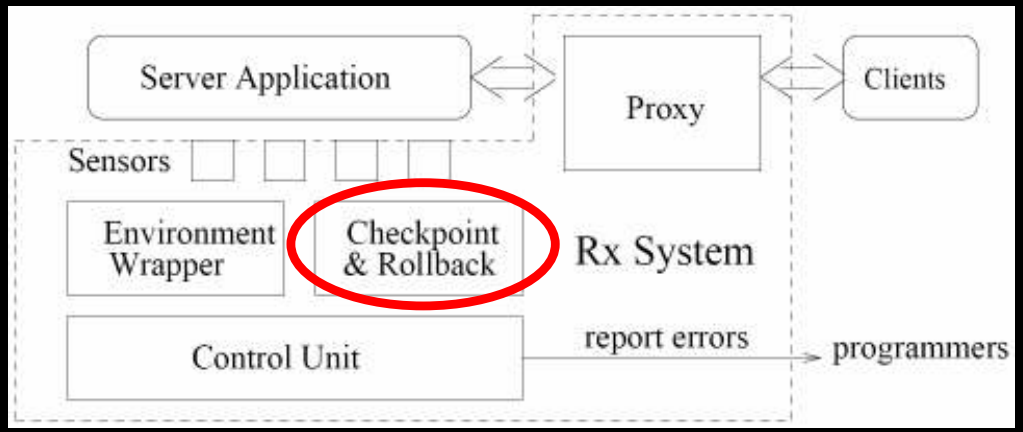
Sensors

- Detect software failures
- Type 1: Software errors reported by OS-raised exceptions
- Type 2: Detect bugs before program crashes



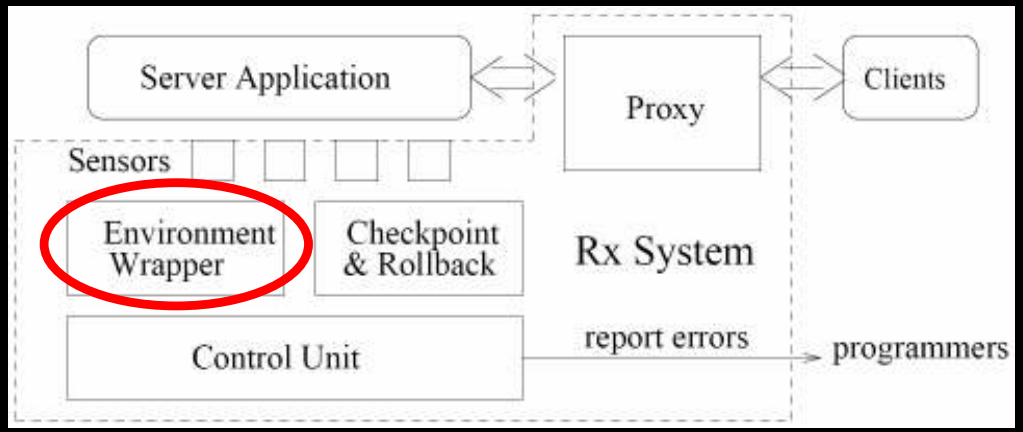
Checkpoint and Rollback

- Takes checkpoints of applications and rolls back
- Minimize amount of checkpoints to keep them in memory
- Checkpoint interval ~200ms in experiments



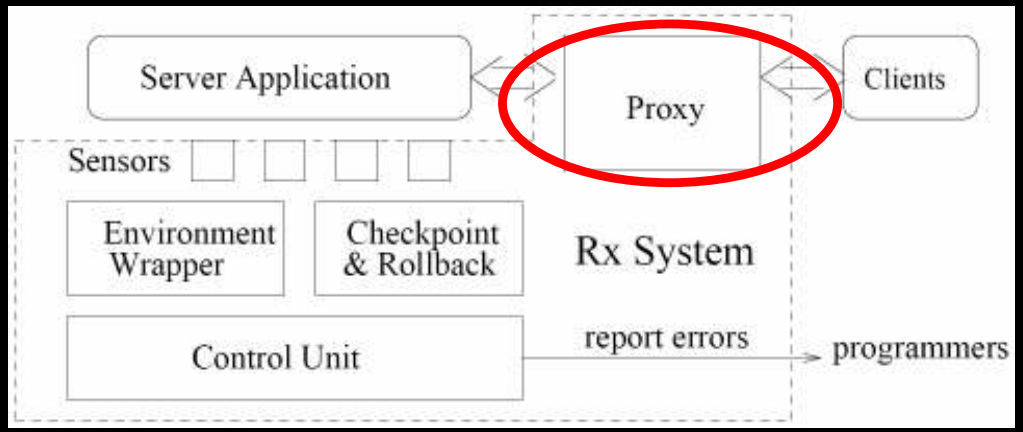
Environment Wrappers

- Memory Wrapper, Message Wrapper, Process Scheduling, Signal Delivery, Dropping User Requests
- Implemented on different levels



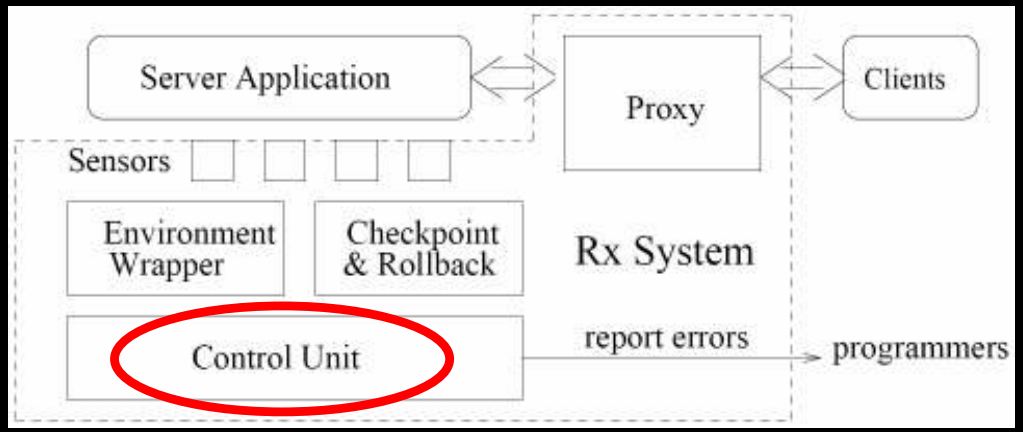
Proxy

- Buffers requests, replays them on rollback
- Granularity: User requests & responses
- Avoids partial and inconsistent responses



Control Unit

- Controls checkpoint interval & rollback
- Failure diagnosis based on symptoms and experiences
- Selection of environmental change
- Information for post-mortem bug analysis



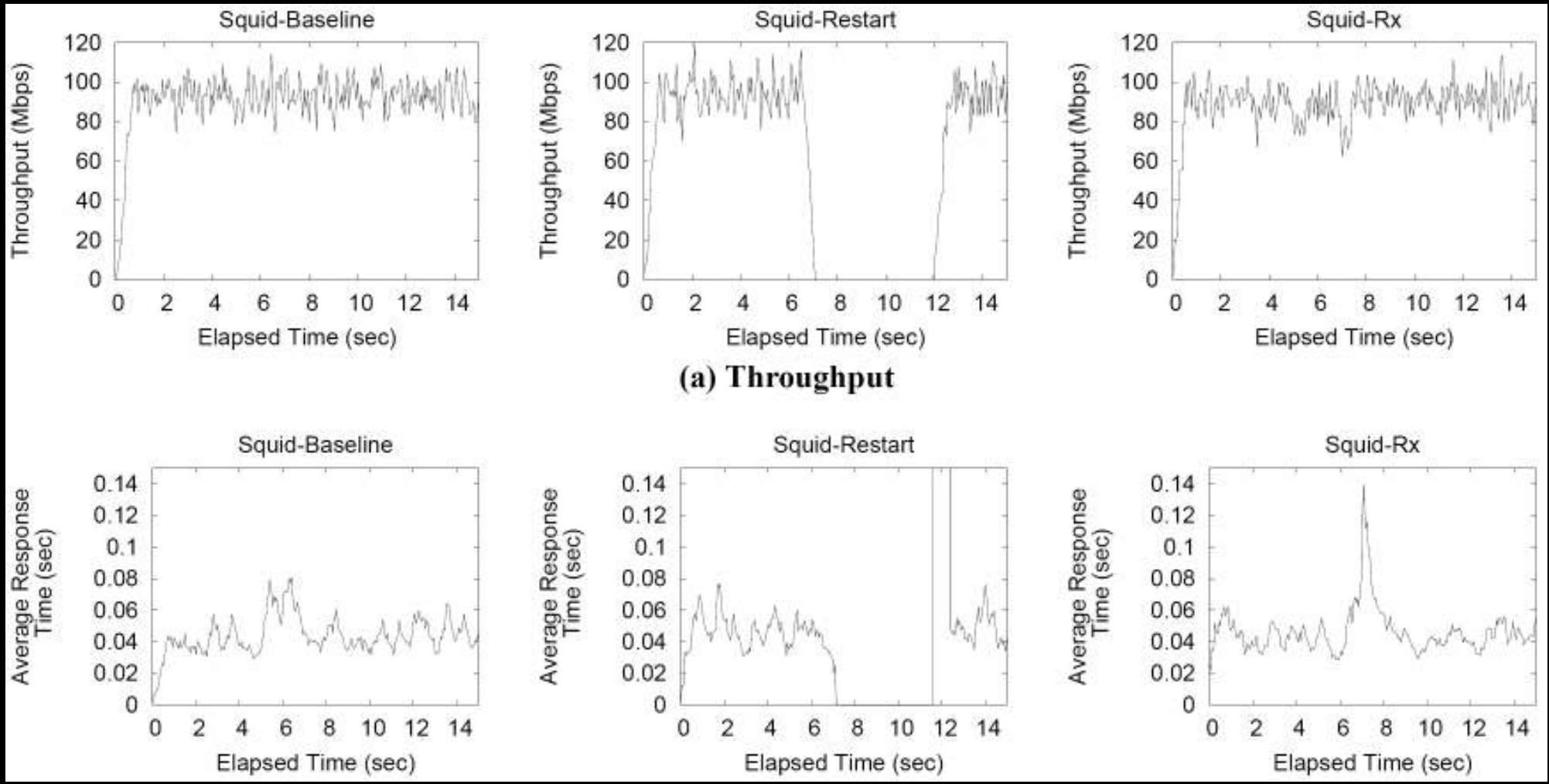
Evaluation

App	Ver	Bug	#LOC	App Description
MySQL	4.1.1.a	data race	588K	a database server
Squid	2.3.s5	buffer overflow	93K	a Web proxy cache server
Squid-ui	2.3.s5	uninitialized read		
Squid-dp	2.3.s5	dangling pointer		
Apache	2.0.47	stack overflow	283K	a Web server
CVS	1.11.4	double free	114K	a version control server

Apps	Bugs	Failure Symptoms	Environmental Changes	Clients Experience Failure?		Recoverable?		Average Recovery Time (s)	
				Alternatives	Rx	Alternatives	Rx	Restart	Rx
Squid	Buffer Overflow	SEGV	Padding	Yes	No	No	Yes	5.113	0.095
MySQL	Data Race	SEGV	Schedule Change	Yes	No	40% probablity	Yes*	3.500	0.161
Apache	Stack Overflow	Assert	Drop User Request	Yes	No	No	Yes	1.115	0.026
CVS	Double Free	SEGV	Delay Free	Yes	No	No	Yes	0.010	0.017
Squid-ui	Uninit Read	SEGV	Zero All	Yes	No	No	Yes	5.000	0.126
Squid-dp	Dangling Pointer	SEGV	Delay Free	Yes	No	No	Yes	5.006	0.113

- Alternatives: Simple rollback, restart
- CVS is invoked at every request

Rx Performance

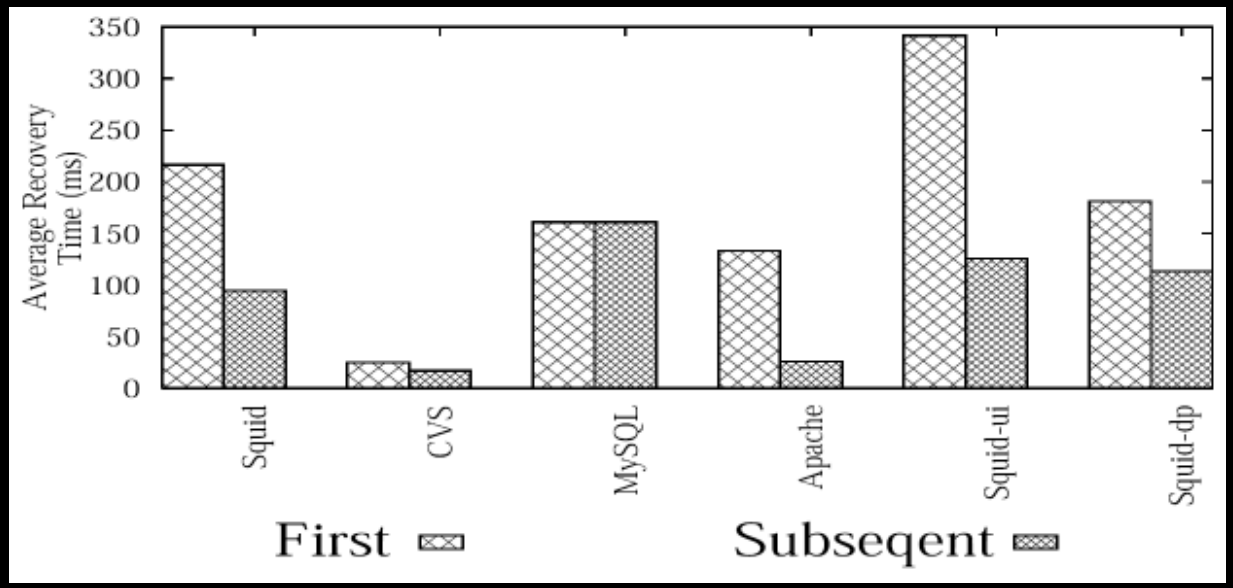


Rx Performance

- Space Overhead
  typically 5-20 checkpoints

Apps	Rx Space Overhead (kB/checkpoint)		
	kernel	proxy	total
Squid	405.35	3.70	409.05
Mysql	300.00	0.16	300.16
Apache	460.00	3.60	463.60
CVS	42.22	2.89	45.11

- Failure Table
  Rx stores what prevented a bug



Conclusion

- Critics
  - Is it really possible to survive a lot of bugs with Rx?
- Questions