zIO: Accelerating IO-Intensive Applications with Transparent Zero-Copy IO

Tim Stamler[1], Deukyeon Hwang[1], Amanda Raybuck[1], Wei Zhang[2], Simon Peter[3]

The University of Texas at Austin[1] *Microsoft*[2] *University of Washington*[3]

IO Copies are Common

Robust data exchange mechanism among application subsystems

IO copy call sites:

Applications (& libraries)

Eg: gRPC, Protobuf

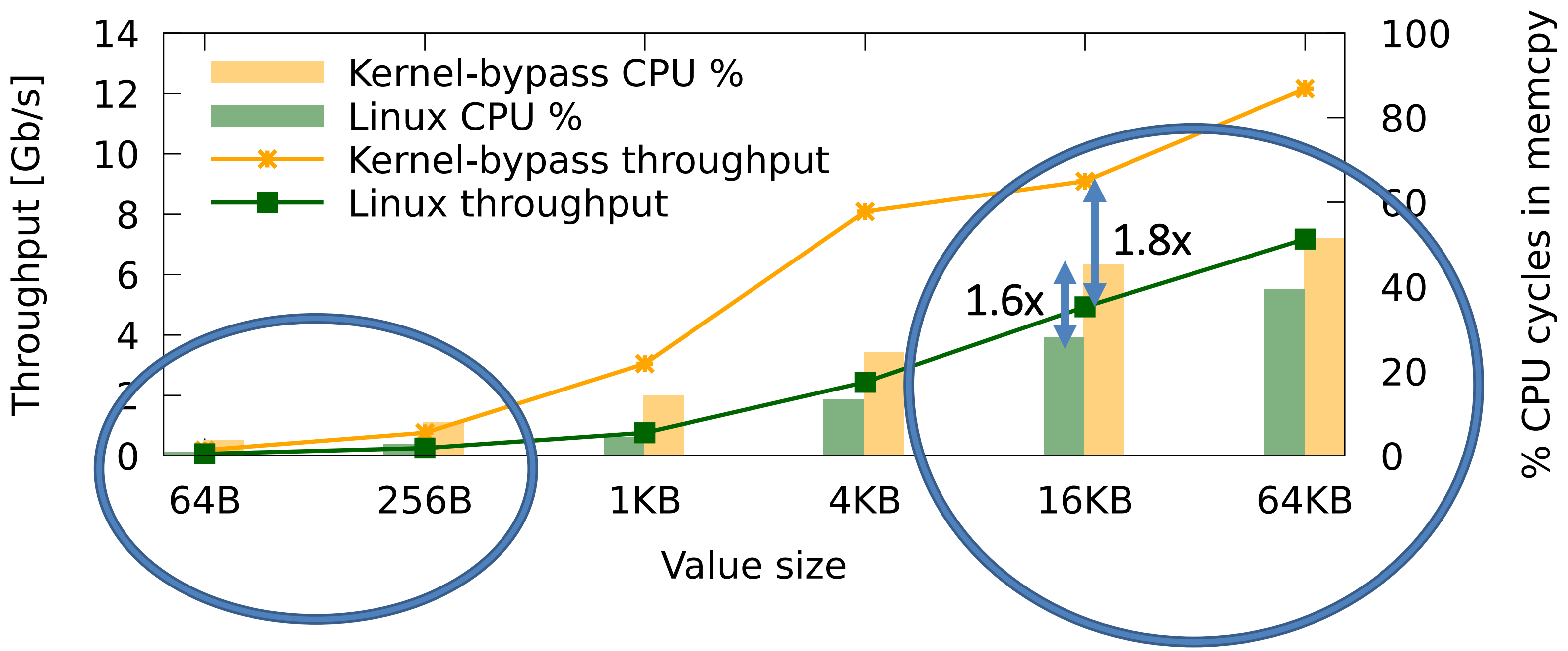
I/O stack APIs

Eg: POSIX API (`recv/send`)

		IO Copy call site	
Application	**Operation**	**App**	**IO Stack**
Redis	SET	4	2
	GET	2	1
Icecast	Cast to N clients	0	1 + N
Ceph	Write	1	2
	Read	0	2
Anna	PUT	5	3
	GET	4	3
MongoDB	Insert	3	2
	Disk sync	1	1
	Read	2	2
Tensorflow-serving	Inference	2	1
Nebula Graph	Insert vertex	5	2
	Store a vertex	4	3
		33	**24**

IO-Intensive Apps are Increasingly Copy-Limited

		Copy call site	
Application	**Operation**	**App**	**IO Stack**
Redis	SET	4	2



More IO => more copies

High CPU overhead from copies at high throughput

Kernel-bypass IO stacks intensify the overhead

Other overheads reduced

Zero-copy IO?

Lots of work on single-stack zero-copy IO APIs:

- Network: Solaris [ATC '96], FreeBSD [IEEE '01], RDMA, netmap [ATC '12]
- Storage: Memory-mapped files

Cross-Stack APIs minimize copies across different IO stacks:

- Demikernel [SOSP '21], PASTE [NSDI '18], Linux sendfile

Success has been limited:

- Many require application modification or have non-transparent requirements
- None seek to eliminate copies within the application (even if more prevalent)

zIO: Transparent Zero-Copy IO

An open-source, transparent IO copy elimination library

Transparently interposes on IO buffer copies

Eliminates application *and* IO stack API copies

Compatible with applications using POSIX IO and libc memcpy/memmove

zIO eliminates IO copies without application modification

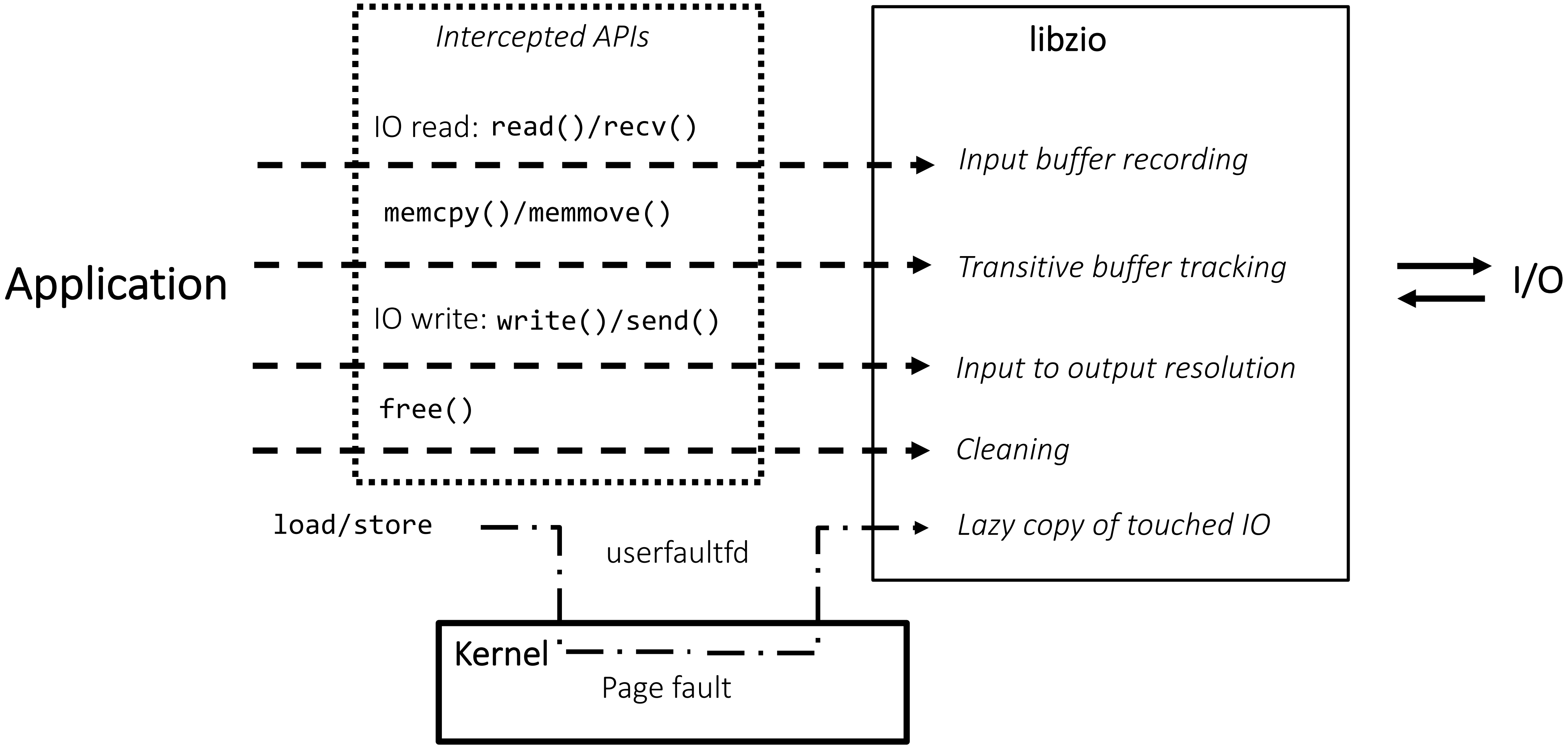
Key Insights

Assumption: much IO data remains untouched by applications
- In this case, the copy doesn't need to happen

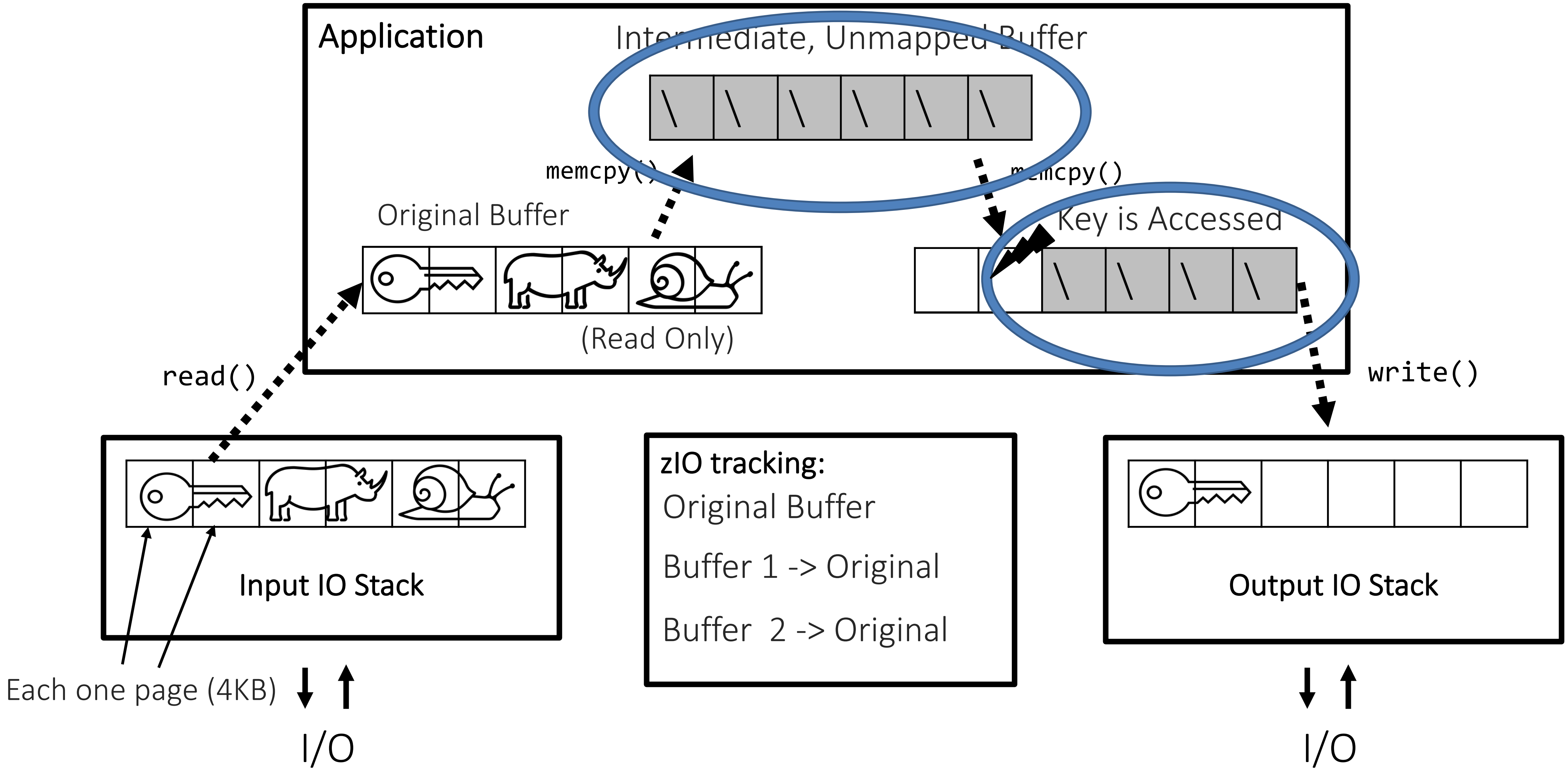
zIO speculatively elides and tracks IO buffer copies
- Record original input buffer location when read from IO stack
- Track and elide subsequent copies of this buffer
- When writing to an output stack, present the *original input buffer*

Upon mis-speculation (IO buffer touched), lazily execute copy

zIO Transparent IO Copy Elision



Example: Application IO Copy Elision



IO Stack API Copy Elision

To elide IO stack API copies, zIO needs to track across the API boundary

- Difficult with kernel stacks; their APIs involve system calls
- Discussed in paper

With kernel-bypass IO:

Kernel-bypass IO stacks hold IO in private buffers in user space

- IO stack API simply copies between app-provided and private buffers
- zIO tracks IO from private buffers as the original and elides the copy

Evaluation

Evaluation Questions

Does zIO improve IO throughput by eliminating copies?

Does zIO improve the performance of real world applications?

Does zIO affect scalability?

How does zIO compare to zero-copy IO APIs?

Experimental Setup

Intel Xeon Gold 6252 CPU 24 cores @ 2.10GHz

196GB RAM

Mellanox ConnectX-5 100Gb/s Ethernet

Benchmarks:

- Network echo server
- Key-value store (Redis)
- HTTP streaming & serving (Icecast)

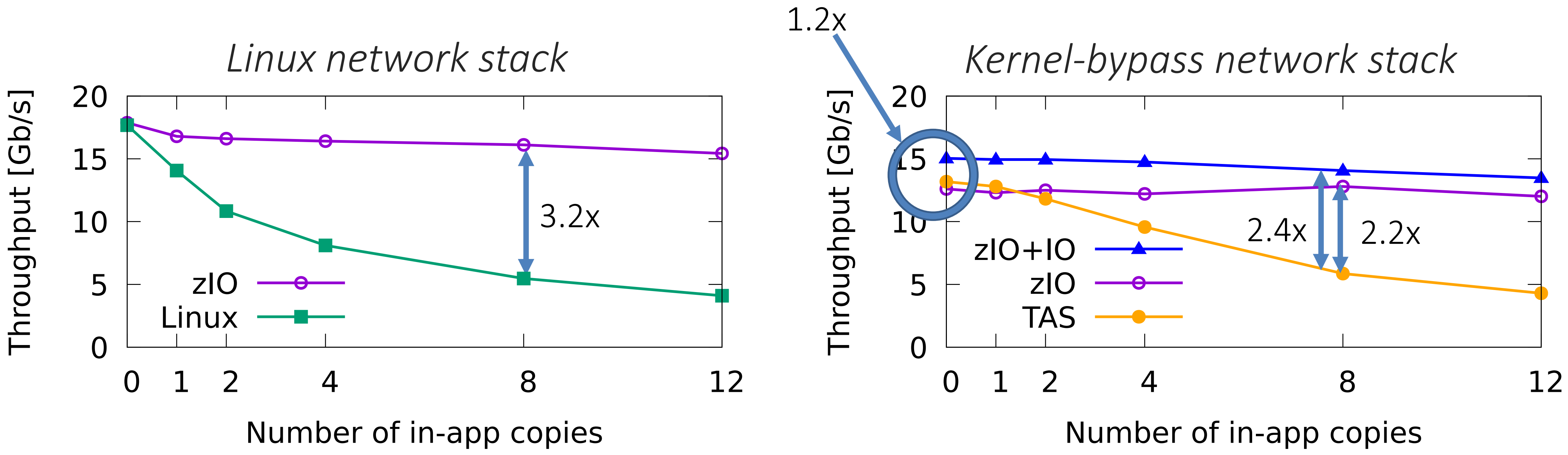
Four configurations:

- Linux
- Elided in-app copies (zIO)
- Kernel-bypass IO (TAS [EuroSys'19], Strata [SOSP'17])
- Elided in-app + IO stack API copies (zIO+IO)

Does zIO Improve IO Throughput?

Network echo server with varying intermediate copies and 512KB messages

Receive data (`recv`), configurable number of app copies, send data (`send`)

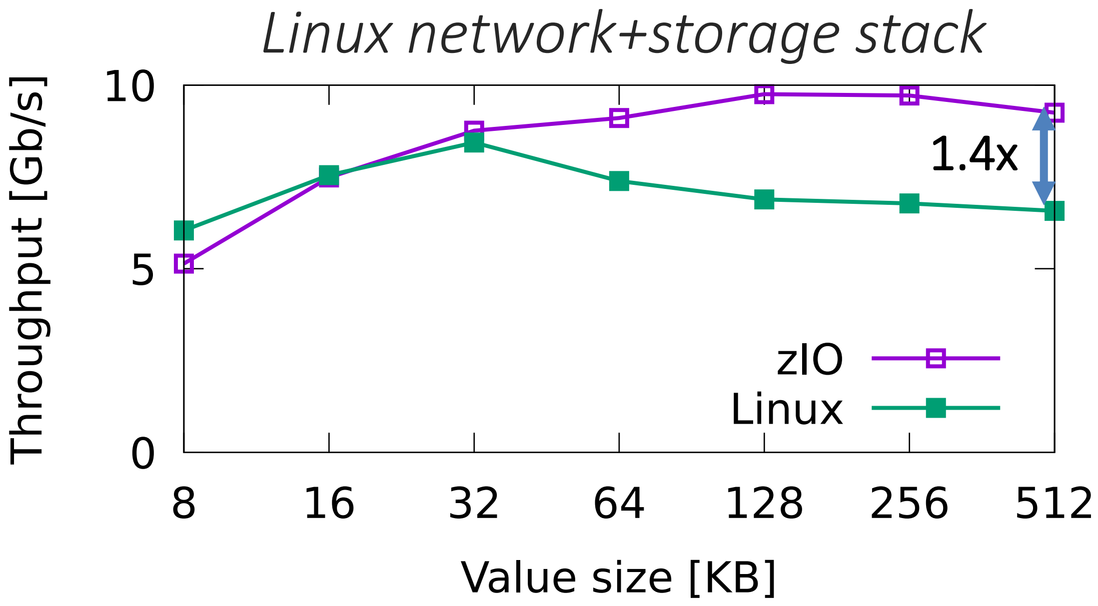


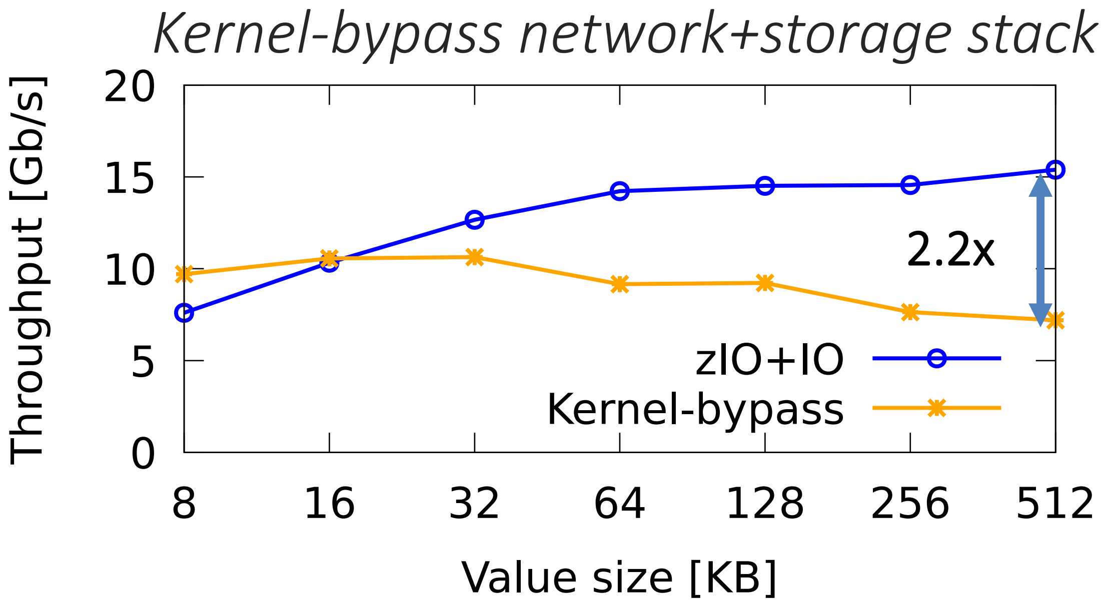
Key-Value Store

		Copy call site	
Application	**Operation**	**App**	**IO Stack**
Redis	SET	4	2
	GET	2	1

YCSB Workload A (50% GET, 50% SET)

Redis with append-only file, persisting every request





HTTP Streaming

Application	Operation	Copy call site	
		App	IO Stack
Icecast	Cast to N clients	0	1 + N

Icecast streaming 1MB audio files in 64KB IO buffer chunks

- Enough listener clients to saturate Icecast server
- Using kernel-bypass IO

Network to network (1.16x higher throughput)

- Single casting client connected to Icecast

Storage to network (1.27x higher throughput)

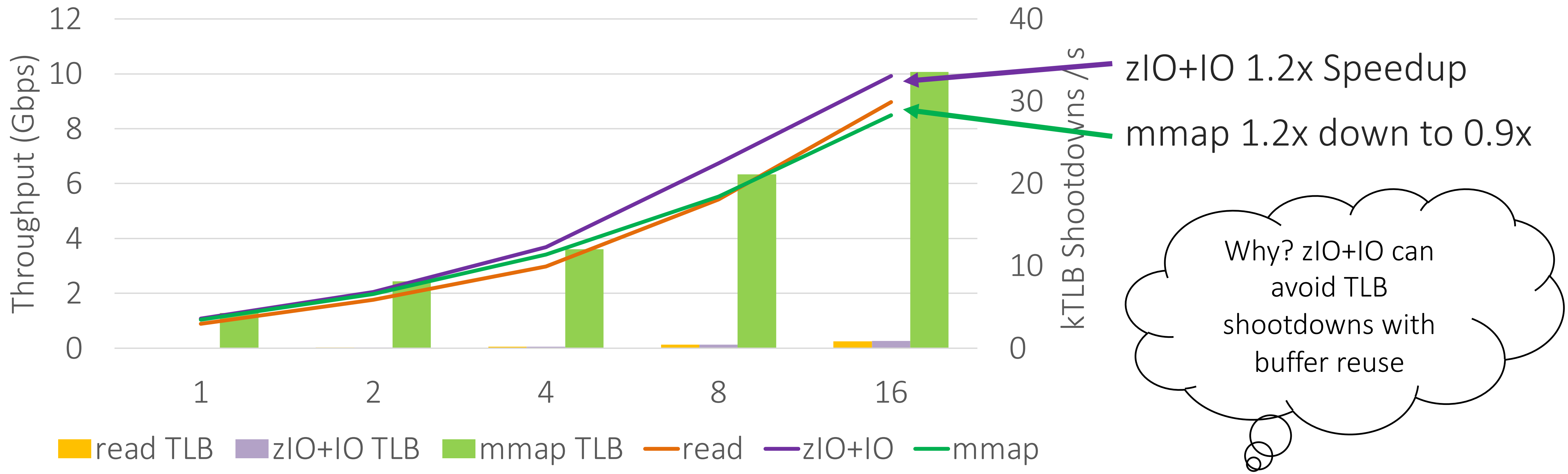
- Icecast streams from local disk

HTTP Serving

Application	Operation	Copy call site: App	Copy call site: IO Stack
Icecast	Serve to N clients	0	1 + N

512KB file in 64KB IO chunks, enough clients to saturate server, kernel-bypass IO

Two versions: 1. **`read`** from file, 2. **`mmap`** file (zero-copy API); both **`send`** on network



Summary

zIO transparently accelerates IO intensive applications

Achieved by

1. Interposing on and eliding IO buffer copies
2. Tracking copied IO buffers, presenting the original on IO output
3. Lazily copying touched IO

1.8x speedup with Linux IO and 2.5x speedup with kernel bypass with Redis

Try it out here!
https://github.com/tstamler/zIO