TCP≈RDMA: CPU-efficient Remote Storage Access with i10

Jaehyun Hwang  Qizhe Cai  Ao Tang  Rachit Agarwal
i10 Motivation: Two trends in Remote I/O

- Move from SCSI to NVMe
- Emergence of NVMe SSDs:
  - >1M IOPS (read), >400K IOPS (write)

1. High-throughput NVMe SSD, high-bandwidth access links
2. Apps use disaggregated storage via Remote I/O
What do these trends mean for Remote I/O?

- **Software stack (iSCSI) performance**

  - 70K IOPS
  - 1M IOPS
  - 2.8M IOPS

- **Storage + network overlap**

  - Bottlenecks at the boundary of storage and network stacks!

**Bottlenecks pushed back to OS, software stack!**

Per-core Throughput (kIOPS)

- Long-lived
- Local I/O
- Remote I/O

- ~30Gbps
### Previous Approaches

#### User-space storage and network stacks
- Storage + Remote I/O (user) + DPDK
- **Performance:** Good!

#### In-Kernel (NVMe-over-TCP)
- Storage + Remote I/O + TCP (all in the kernel)
- **Performance:** Not-so-good!

#### NVMe-over-RDMA
- Storage + Remote I/O (kernel) + RDMA
- **Performance:** Good!

---

**Fundamental?**
Remote Storage Access Overheads: TCP vs. RDMA

CPU Usage (%)

- NVMe-over-TCP
- NVMe-over-RDMA

Network processing overhead!

Context switching overhead!
i10 Summary

• A new remote I/O stack implemented entirely in the kernel
  – No changes in apps, no changes in TCP/IP stack, no changes in hardware

• Throughput-per-core similar to NVMe-over-RDMA
  – Latency within 1.7x of RDMA (for SSD accesses)

• Three simple ideas
  – i10-lane: dedicated resources
  – i10-caravans: request and data batching
  – Delayed doorbells: Interrupt coalescing

Minimize network processing

Minimize context switching
**i10-lane**: Dedicated per-(core, target) pair “pipe”

- **Block**
- **i10**
- **TCP/IP**

**Diagram**:
- **Block i10**: Dedicated per-(core, target) pair “pipe”
- **NVMe SSD**
- **NIC**
- **Target**
- **Host**
- **I/O syscalls**
- **Per-core block queues**
- **Dedicated i10 queues**
- **Dedicated TCP buffers**
**i10-lane**: Dedicated per-(core, target) pair “pipe”

- **Block**
- **i10**
- **TCP/IP**

[Diagram showing data flow from Block to i10 to TCP/IP to NIC, with NVMe SSD and NIC connections.]
**i10-lane**: Why per-(core, target) pair lanes?

- **Per-target**: Too much contention
- **Per-core**: Fewer batching opportunities

All requests in each i10 queue are destined to the same target over the same TCP connection.
**i10 Caravans:** i10-lanes enable efficient batching

Significantly reduce per-byte network processing overhead!

- Allow larger payloads up to 64KB using TSO
- No CPU cycles for packet segmentation

One socket call per caravan (~64KB)
Context switching in Remote I/O (without i10)

High thread switching overhead! (1-3us per request)
**Delayed doorbells: Minimizing context switching**

- **Block**
- **i10**
- **TCP/IP**
- **NVMe SSD**
- **NIC**
- **Target**
- **Host**

- Ring doorbell only after “caravan size” worth of requests
- Under low loads, use a timer (e.g., 50us)
i10 Evaluation Setup

• Two 24-core servers connected directly
  • 100Gbps Mellanox CX-5
  • No switches in middle — ensure bottlenecks in the kernel

• NVMe-device at both servers
  • ~700k IOPS (read), ~400k IOPS (write)
  • ~100us read latency

• No specialized hardware functionalities used in i10 evaluation
  • For hardware and software configuration, see the paper.
i10 Evaluation: how does i10 performance ...

• ... compare to NVMe-over-RDMA?
  – Metrics of interest: throughput per core, average latency, tail latency

• ... compare to user-space stacks?

• ... vary with different workloads, hardware and applications?
  – read/write ratios
  – Delayed doorbell timer
  – Aggregation size
  – I/O Request sizes
  – Storage device access latency
  – Real applications
  – Number of target devices

• ... scale with number of cores?

• ... depend on various design aspects (lanes, caravans, delayed doorbell)?

Please see our paper!

Answer:

* Throughput: Comparable (or better)
* Latency: Not terrible (<1.7X)

Teacher: why do you have the same answers?

Student: because we have the same questions

Teacher:
TCP \approx RDMA:
- Throughput: **Comparable**
- Tail latency: \(<1.7X\)

High load latency: TCP \approx RDMA

---

TCP \approx RDMA:
- Throughput: **Comparable (or better)**
- Tail latency: +97us
Scalability with number of cores

TCP ≈ RDMA:

- Throughput: Scales similar (~14 cores) or better
- Seems related to hardware scalability
Benefits from individual design components

Each of the design component contributes to i10 performance
i10 improves over NVMe-over-TCP by using

Fewer cycles for network processing (Net Tx/Rx) and scheduling (Others)

More cycles for Applications, and block layer operations (Blk Tx/Rx)
Kernel implementation
Further evaluation
Test scripts ...
All are available at:
https://github.com/i10-kernel/

TCP ≈ RDMA: CPU-efficient Remote Storage Access with i10

i10 is a new in-kernel remote storage I/O stack for high-performance network and storage hardware. Our i10 design offers a number of benefits: