SDC

STORAGE DEVELOPER CONFERENCE

SNIA SANTA CLARA, 2016

Performance Implications Libiscsi RDMA support

Roy Shterman

Software Engineer, Mellanox

Sagi Grimberg

Principal architect, Lightbits labs

Shlomo Greenberg

Phd. Electricity and computer department

Ben-Gurion University, Israel

Agenda

- Introduction to Libiscsi
- Introduction to iSER
- Libiscsi/iSER implementation
- The memory Challenge in user-space RDMA
- Performance results
- Future work

What is Libiscsi?

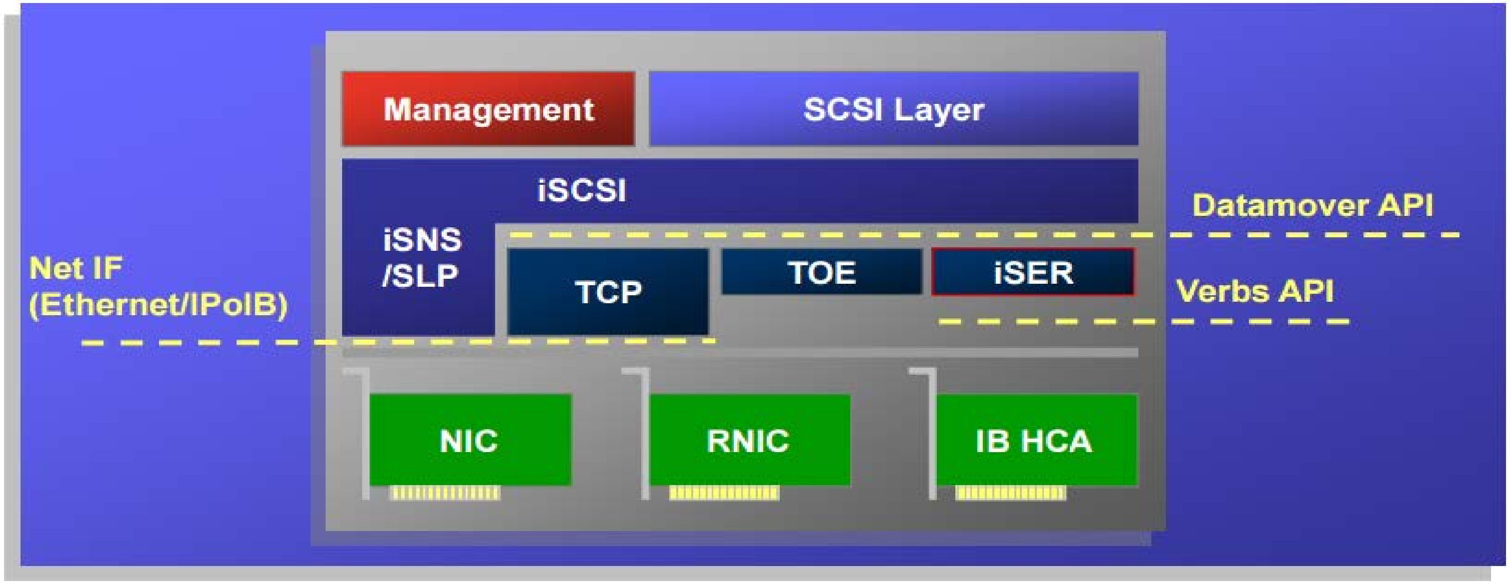
- iSCSI initiator user-space implementation.
- High performance non-blocking async API.
- Mature.
- Permissive license (GPL).
- Portable, OS independent.
- Fully integrated in Qemu.
- Written and maintained by Ronnie Sahlberg [https://github.com/sahlberg/Libiscsi]

Why Libiscsi?

- Originally developed to provide built-in iSCSI client side support for KVM/QEMU.
- Process private Logical Units (LUNs) without the need to have root permissions.
- Since, grew iSCSI/SCSI compliance test-suits.

iSCSI Extensions for RDMA (iSER)

- Part of IETF RFC-7147

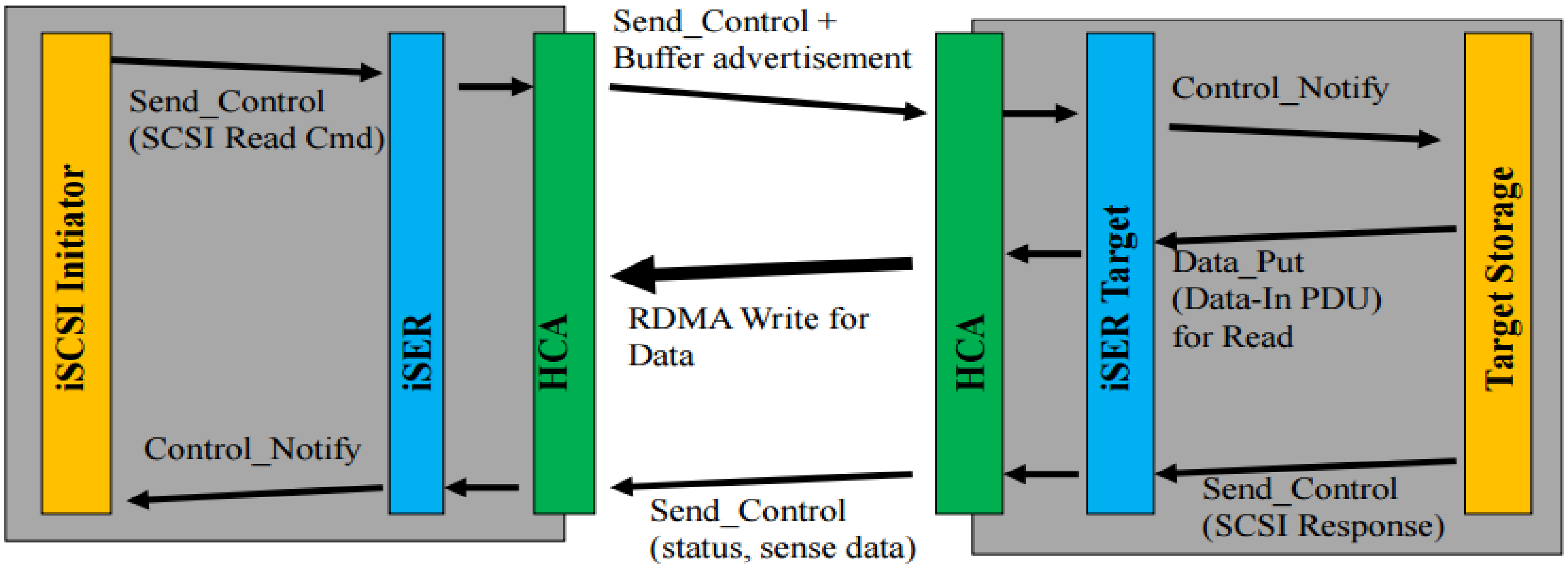


- Transport layer iSER or iSCSI/TCP are transparent to the user.

iSER benefits

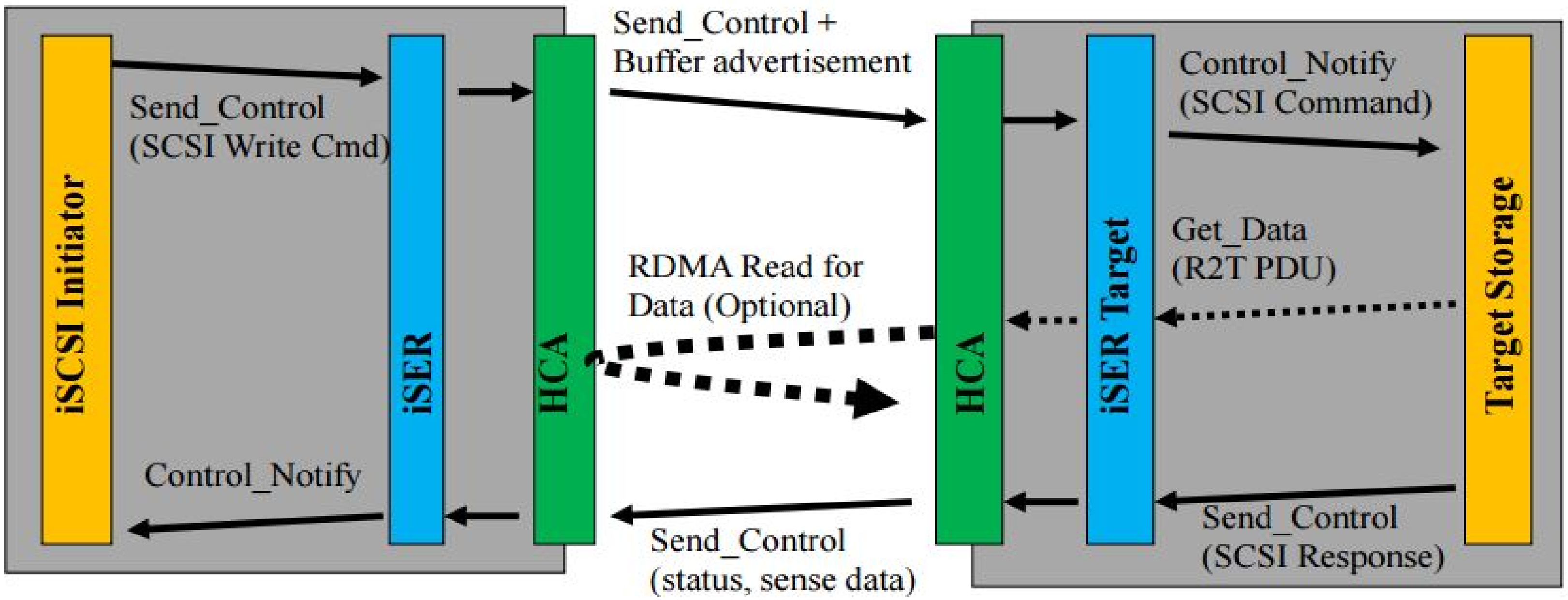
- Zero-Copy
- CPU offload
- Fabric reliability
- High IOPs, Low latency
- Inherits iSCSI management
- Fabric/Hardware consolidation
- InfiniBand and/or Ethernet (RoCE/iWARP)

iSER Read command flow



- SCSI Reads
  - Initiator send Protocol Data Unit with encapsulated SCSI read to target.
  - Target writes the data into Initiator buffers with RDMA_WRITE command.
  - Target initiate Response to the Initiator that will complete the SCSI command.

iSER Write command flow



- SCSI Writes
  - Initiator send Protocol Data Unit with encapsulated SCSI write to target (can contain also inline data to improve latency).
  - Target reads the data from initiator buffers with RDMA_READ commands.
  - Target initiate Response to the Initiator that will complete the SCSI command.

Libiscsi iSER implementation

- Transparent integration.
- User-space networking (kernel bypass).
- High performance.
- Separation of data and control plane.
- Reduce latency by using non-blocking fd polling.

Libiscsi stack modification

- Layered the stack
  - Centralized transport specific code
  - Added a nice transport abstraction API
  - Plugged in iSER

```
typedef struct iscsi_transport {
		int (*connect)(struct iscsi_context *iscsi, union socket_address *sa, int ai_family);
		int (*queue_pdu)(struct iscsi_context *iscsi, struct iscsi_pdu *pdu);
		struct iscsi_pdu* (*new_pdu)(struct iscsi_context *iscsi, size_t size);
		int (*disconnect)(struct iscsi_context *iscsi);
		void (*free_pdu)(struct iscsi_context *iscsi, struct iscsi_pdu *pdu);
		int (*service)(struct iscsi_context *iscsi, int revents);
		int (*get_fd)(struct iscsi_context *iscsi);
		int (*which_events)(struct iscsi_context *iscsi);
} iscsi_transport;
```

QEMU iSER support

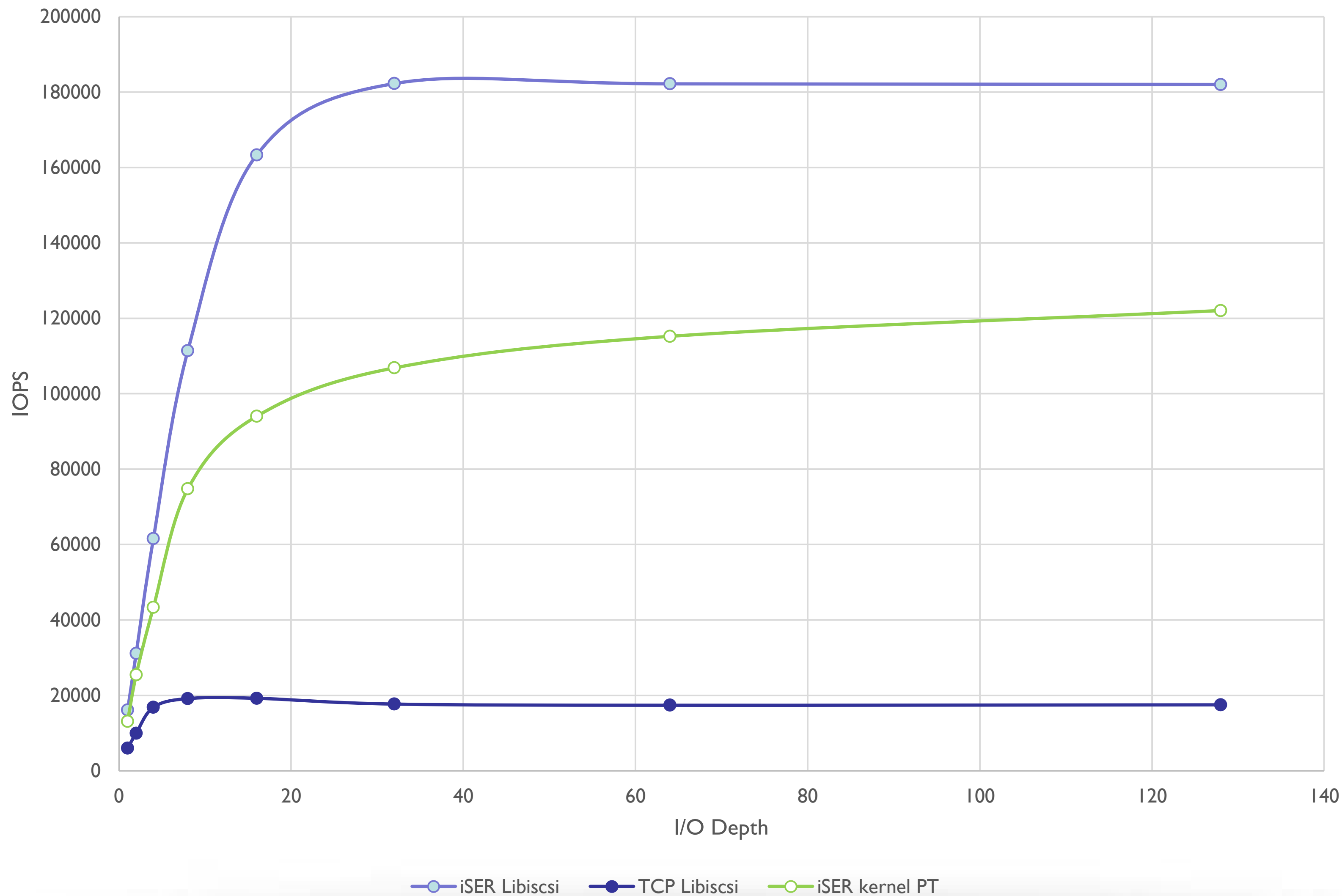
- Qemu iSCSI block driver needed some modifications to support iSER.
  - Move polling logic to the transport layer.
  - Pass IO vectors to the transport stack.
- Work in progress
  - should be available in the next few weeks.
- Also through libvirt!

SDC 16

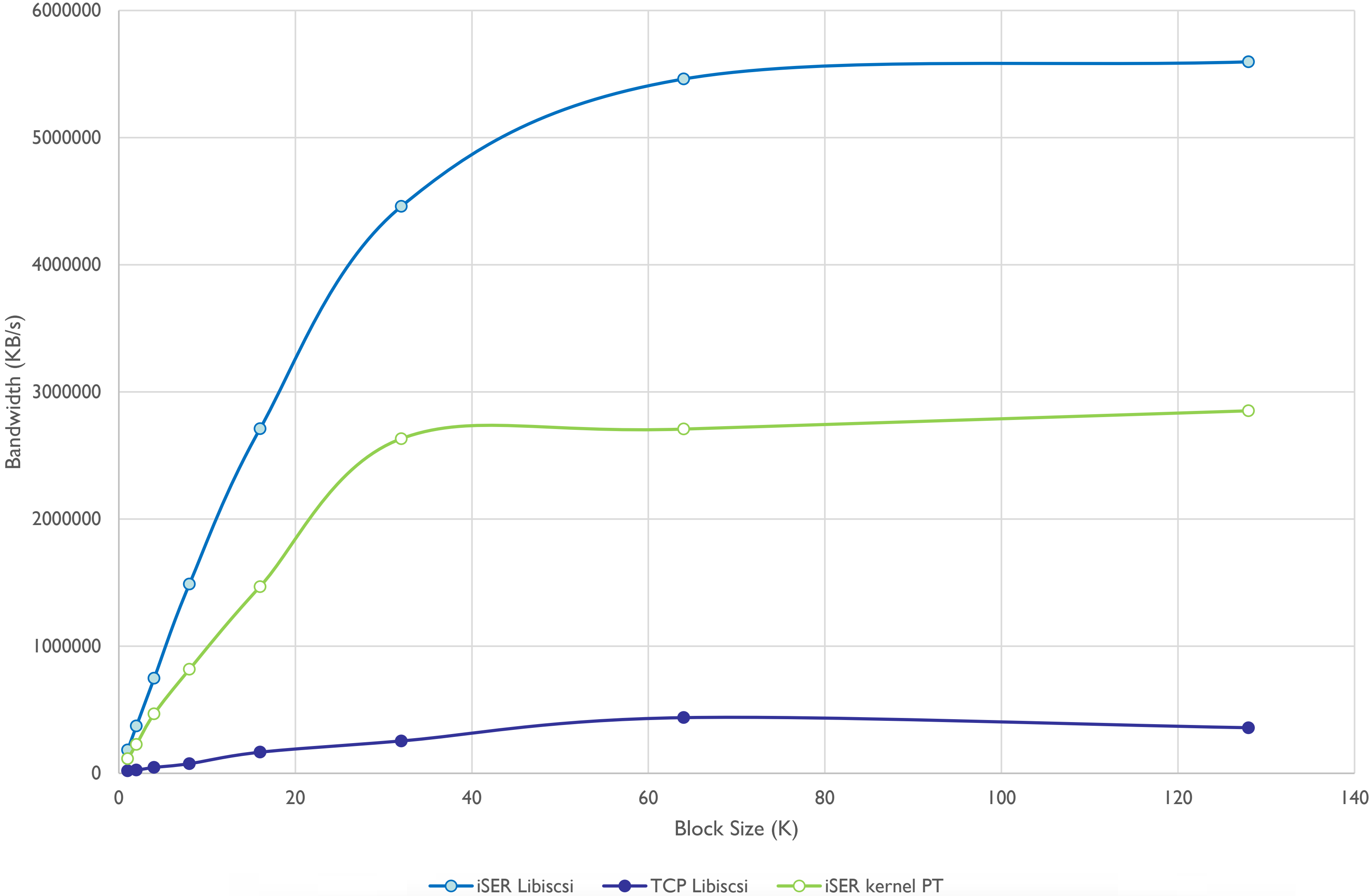
Experiments and results

- Performance measured with Mellanox ConnectX4 on both initiator and target.
- Target side was TGT user-space iSCSI target with RAM storage devices.
- IO generator was FIO (Flexible I/O tester).
- Each guest with single CPU core and single FIO process.
- Comparison against iSCSI/TCP and block device pass-through of iSER devices.

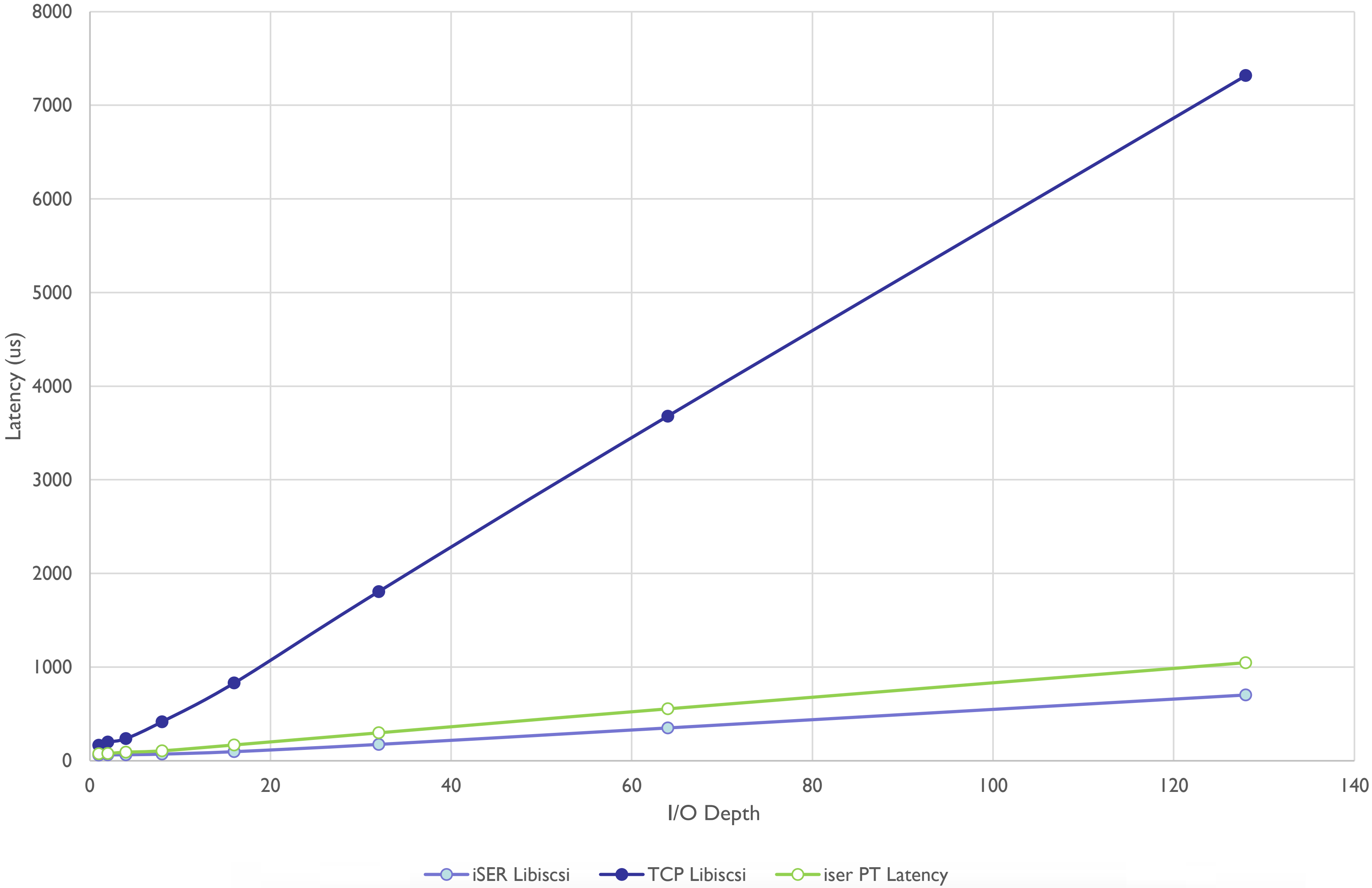
IOPS vs I/O depth

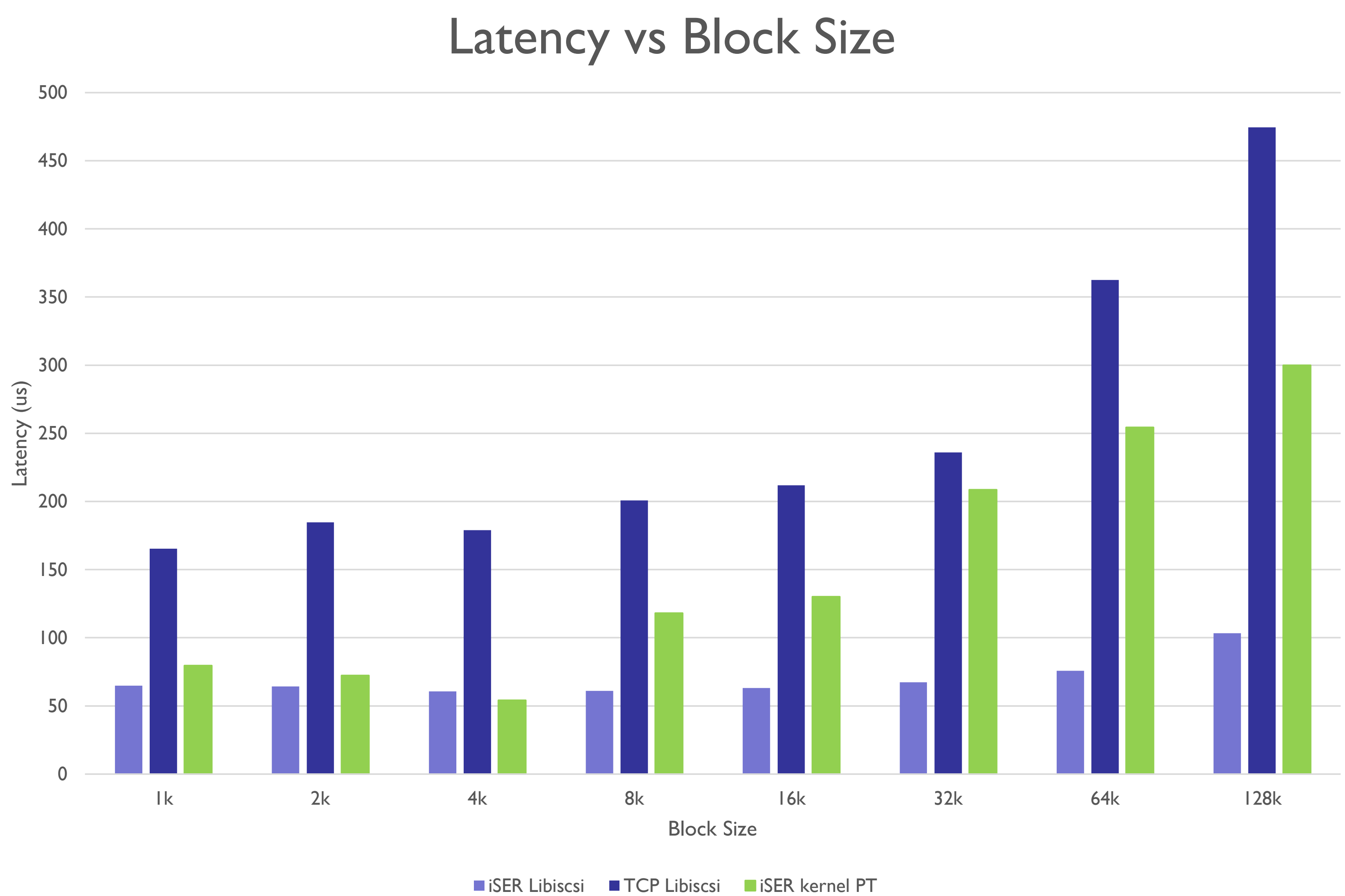


Bandwidth vs Block size

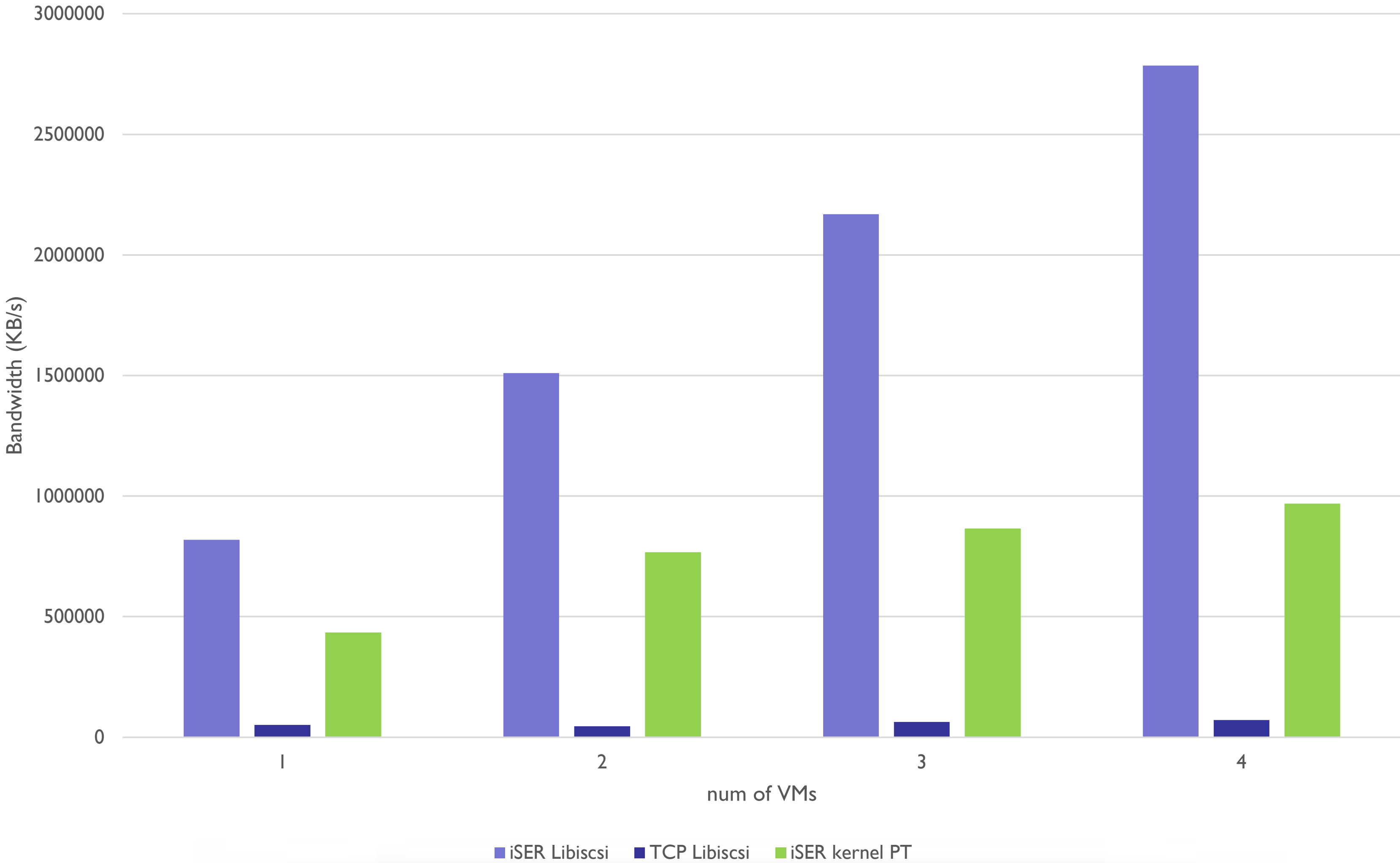


Latency vs I/O depth

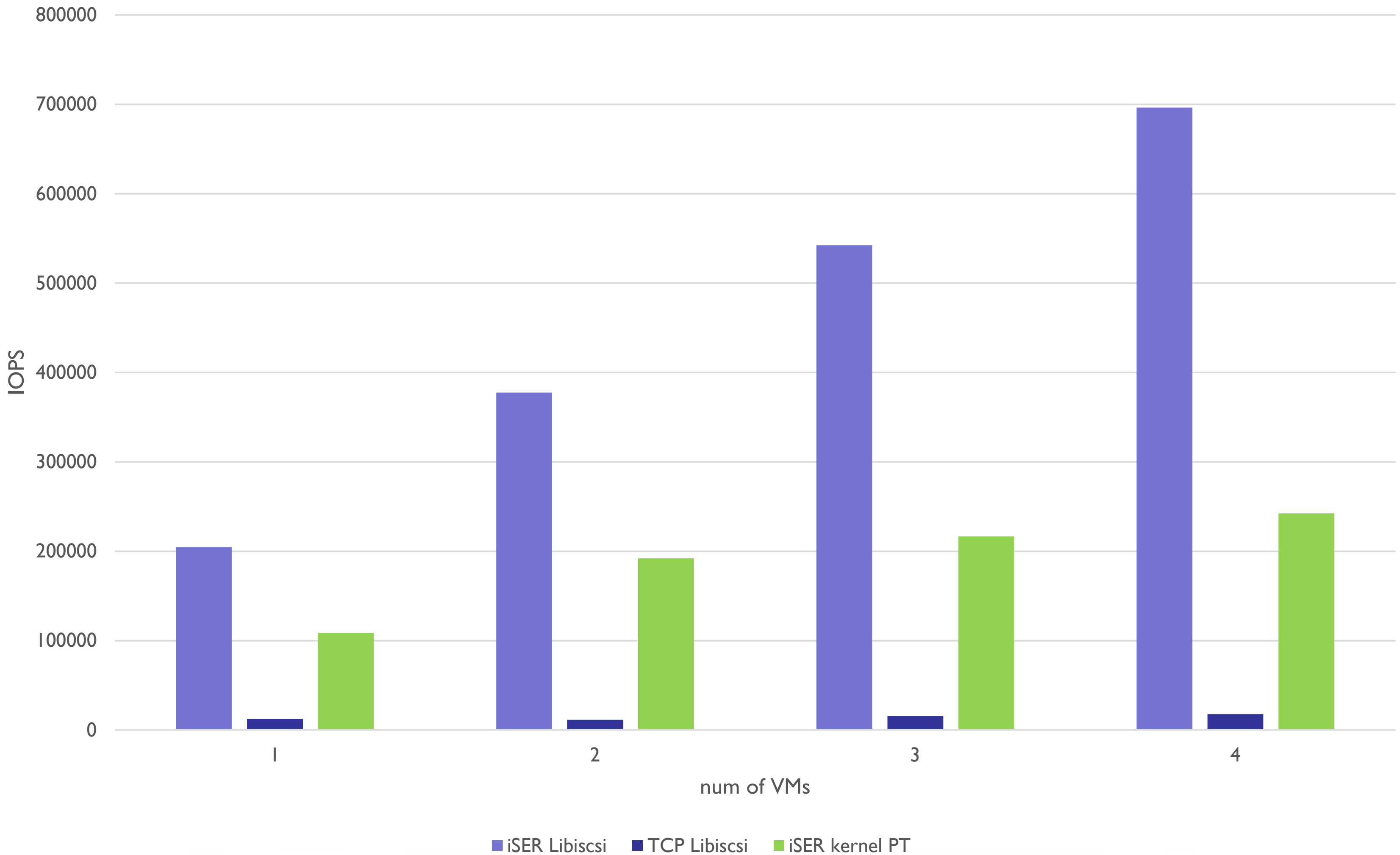



Latency vs Block Size
500
450
400
350
300
250
200
150
100
50
0
Latency (us)
1k
2k
4k
8k
16k
32k
64k
128k
Block Size
iSER Libiscsi
TCP Libiscsi
iSER kernel PT

Bandwidth across multiple VMs



IOPS across multiple VMs



RDMA Memory registration

- In order to allow remote access the application needs to map the buffer with remote access permissions.
- Mapping operation is slow and not suite-able for the data-plane.
- Applications usually preregister all the buffers intended for networking and RDMA

Memory registration in Mid-layers

- Mid-layers often don't own the buffers but rather receives them from the application.
  - Examples: OpenMPI, SHMEM and Libiscsi/iSER
- Memory registration for each data-transfer is not acceptable.

Possible solutions

1) Pre-register the entire application space.
2) Modify applications to use Mid-layer buffers.
3) “Pin-down” Cache: Register and cache mappings on the fly.
4) Page-able RDMA (ODP): Let the device and the kernel handle IO page-faults

RDMA paging - ODP

- RDMA devices can supports IO page-faults
- App can register "huge" virtual memory region (even entire memory space).
- HW and kernel handle page-faults and page invalidations
- If locality is good enough, performance penalty is amortized.
- Not bounded to physical memory.

iSER with ODP and memory windows

- iSER can leverage ODP for a more efficient data-path
- But, cannot map non-IO related memory for remote access.
  - Solution: Open a memory window on a page-able memory region (fast operation – can be used in the data-path).
  - ODP support for memory windows is on the works.
- Initial experiments with ODP look promising.

Future Work

- Leveraging RDMA paging support to reduce the memory foot-print.
- Plenty of room for performance optimizations.
- Stability improvements.
- Libiscsi iSER unit tests.

Acknowledgments

- This project was conducted under the supervision and guidance of Dr. Shlomo Greenberg, Ben-Gurion University.
- Special thanks to Ronnie Sahlberg, creator and maintainer of the Libiscsi library for his support.