VSOCK

From Convenience to Performant VirtIO Communication

Amery Hung, Linux Kernel Engineer
Bobby Eshleman, Linux Kernel Engineer
Systems Technology and Engineering, ByteDance

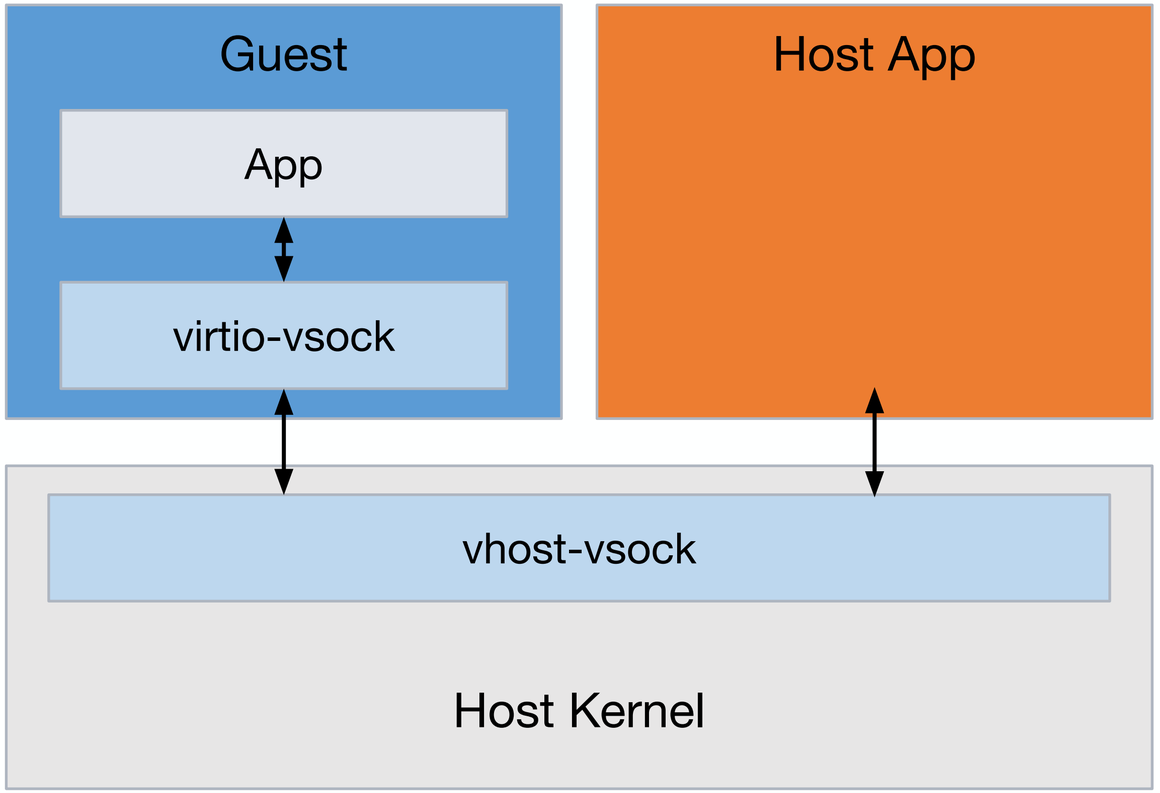
ByteDance 字节跳动

Agenda

- Background
  - VSOCK usage / features
  - Protocol Overview
- Performance
- Recent work
  - Datagram Support
  - Socket Map
- Future work

Background

- VSOCK
  - socket family (AF_VSOCK)
  - zero-configuration communication between virtual machines and the host
  - context ID (CID) identifies host or VM
  - ports used as usual
  - specified by virtio (hyper v and vmware also have implementations)



Background

- Common use case is guest agents (*qemu-guest-agent* and *kata-agent*)
  - qemu-guest-agent
    - suspend, backup, etcc
  - kata-agent
    - talk with kata-runtime, manage/supervisor containers in guest
  - Preferred over TCP/IP because
    - Adding a virtual NIC is overly intrusive
    - Requires changes (and maintenance) to both host and guest network configuration
  - Preferred over serial because
    - Serial doesn't support socket API
      - Multiple senders/receivers often require additional proxy service for multiplexing
    - The serial link limits read/write access to one process at a time
    - Custom protocol must be handwritten for some features like message boundaries

The Protocol

- Connection Establishment:
  - Initiated with a two-way handshake
    - Client sends a REQUEST packet.
    - Server responds with a RESPONSE packet to establish the connection.
- Data Transmission:
  - Application data is sent in RW packets.
  - Received data is forwarded to the application by the destination socket.
  - Destination socket sends control flow information (credit update) to the source.
- Connection Termination:
  - Terminated with a two-way tear-down process.
  - Disconnecting side sends a SHUTDOWN packet.
  - This SHUTDOWN packet is acknowledged with an RST packet, terminating the connection.

The Protocol

- Control flow via credit allocation
- All packet headers include *fwd_cnt and buf_alloc* so that endpoints may inform each other of
  - How much data the has already been forwarded to the application
  - How much total receive buffer is currently allocated
- Sources may calculate:
  - free_buffer = total_buffer - (already_sent - already_forwarded)
- That is, senders always know how much more data a receiver can handle
- Implicit updates when sending RW payloads
- Updates may also happen explicitly via credit request and credit update messages
- For Linux virtio-vsock, credit updates are volunteered after forwarding messages to user space (e.g., recvmsg())

VSOCK Performance

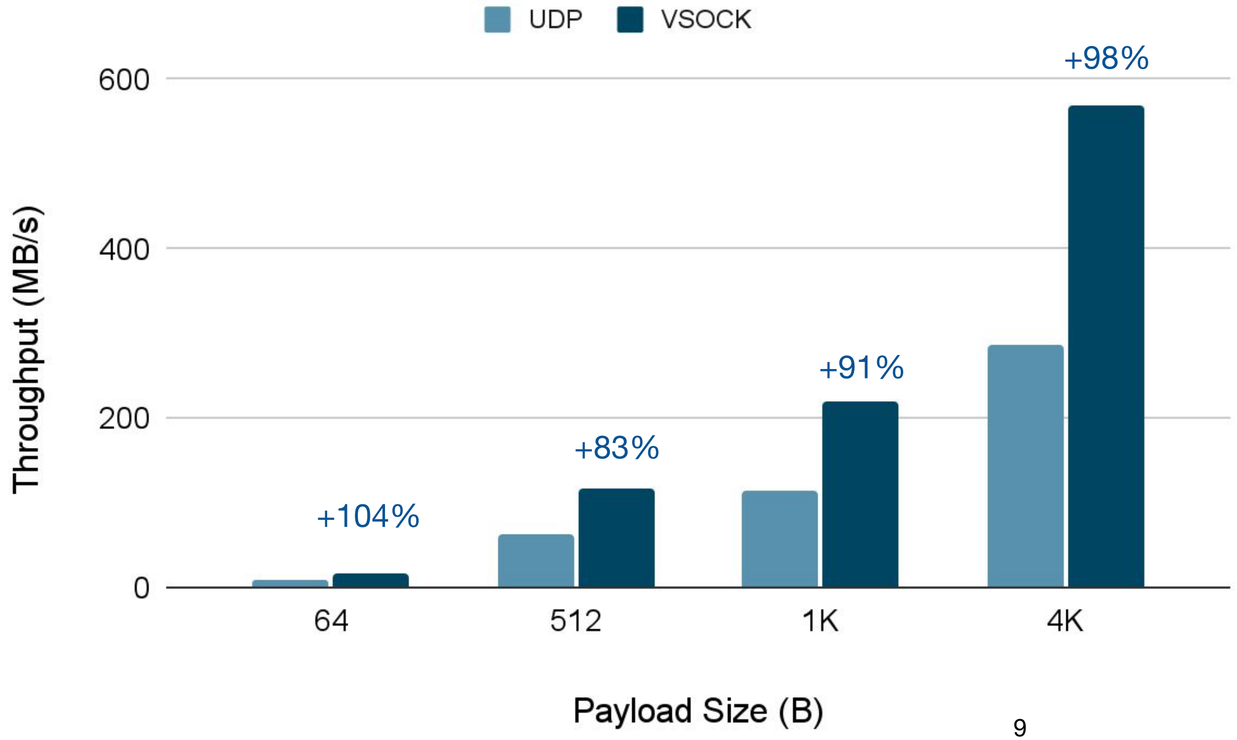
- VSOCK is optimized for convenient usage by applications and design simplicity
- Not originally designed with performance as the primary goal
  - Single queue virtqueue
  - workqueue sender only
    - workqueue wakeup latency always incurred (amortized by batching)
  - possible room for improvement in synchronization, cache, and memory efficiency
- Some implementation details are very helpful for performance
  - Batching
  - No need for much of networking stack (routing, filters, etc...)
    - Cycles saved because features not needed/supported
- Currently not often used for performance-sensitive workloads
  - We are seeing potential for this use case

Recent Work: Datagrams for virtio-vsock

- Status
  - virtio not upstream yet
  - Datagrams already supported on vmware, but not virtio
  - Datagram POC patches sent to mailing list
    - Linux:
      - https://lore.kernel.org/all/20230413-b4-vsock-dgram-v5-0-581bd37fdb26@bytedance.com/
    - virtio spec:
      - https://lore.kernel.org/all/20230829181549-mutt-send-email-mst@kernel.org/#r
- Features/notes
  - Connection-less, unreliable, may drop packets
  - Does not use credits
    - No destination socket necessarily exists to allocate credits
    - Destination and source socket lifetimes are out-of-sync, therefore forwarded and sent totals are also out-of-sync
    - Proposal in development uses start/stop messages to control flow

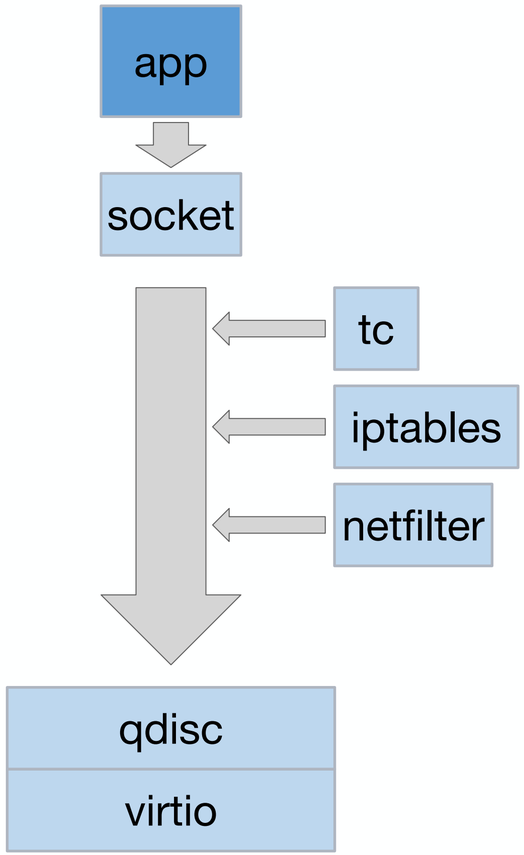
Recent Work: Datagrams for virtio-vsock

- Performance comparison with UDP
  - Configuration
    - virtio-net is vhost/virtio with mq (1 rx and 1 tx queue per CPU)
    - 8 vcpus
    - 1 sending thread per vcpu
    - Linux host w/ 1 guest
    - vsock and udp have equal SO_SNDBUF



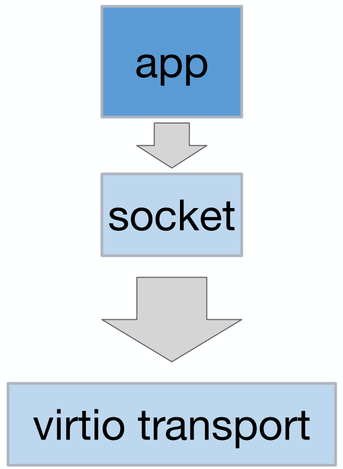
High-level Component View

UDP/IP/virtio



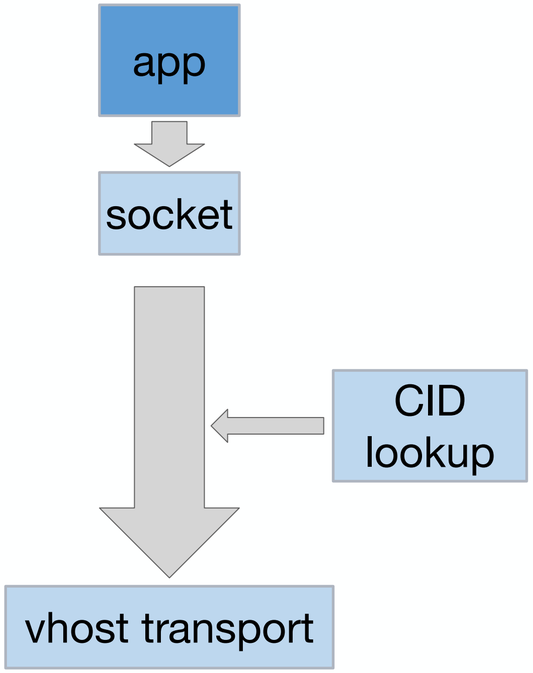
Becomes

VSOCK



Guest to Host

or



Host to Guest

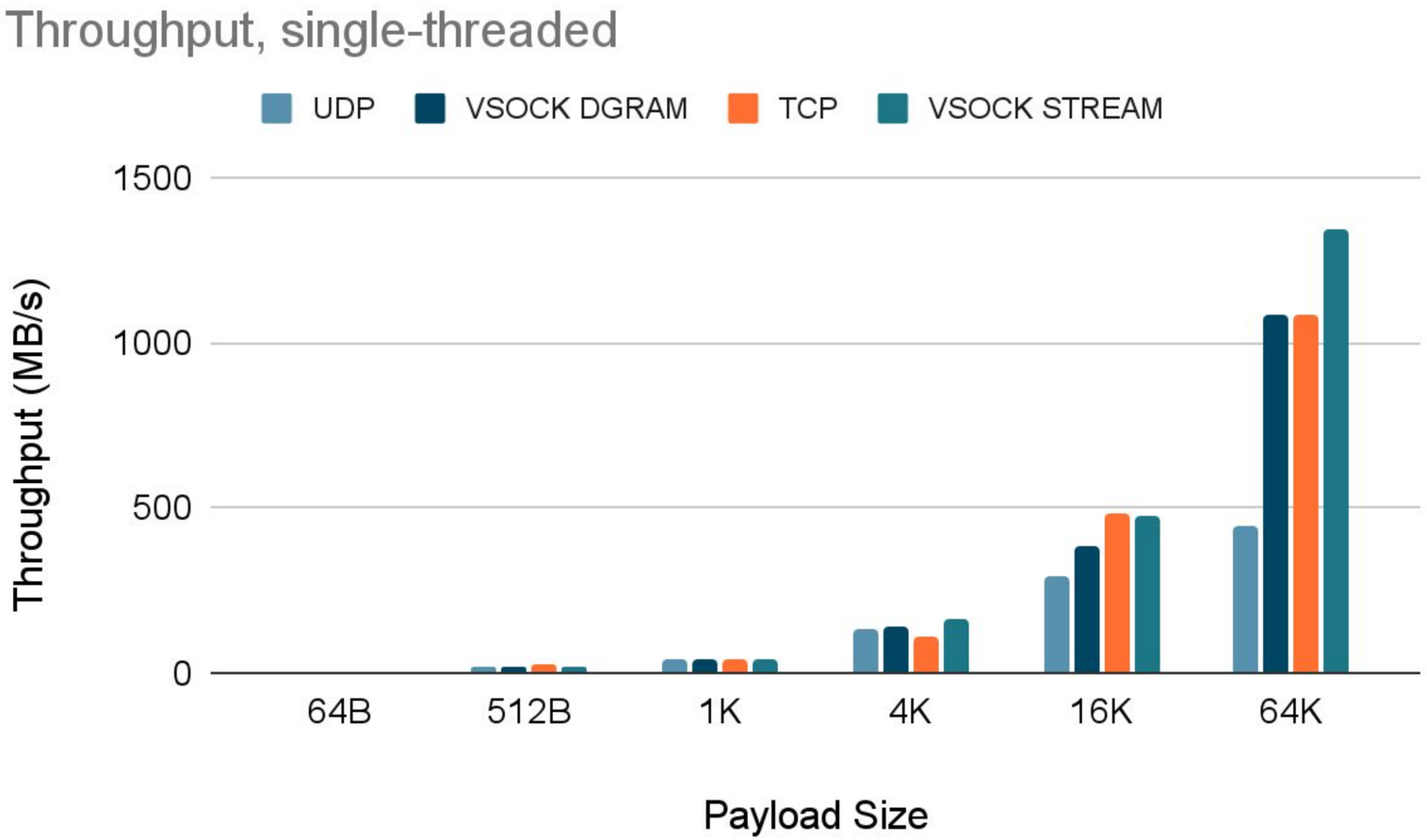
UDP vs VSOCK DGRAM function latencies

- Why does VSOCK datagram perform well?
  - One reason: VSOCK datagram enqueueing has comparatively low latency

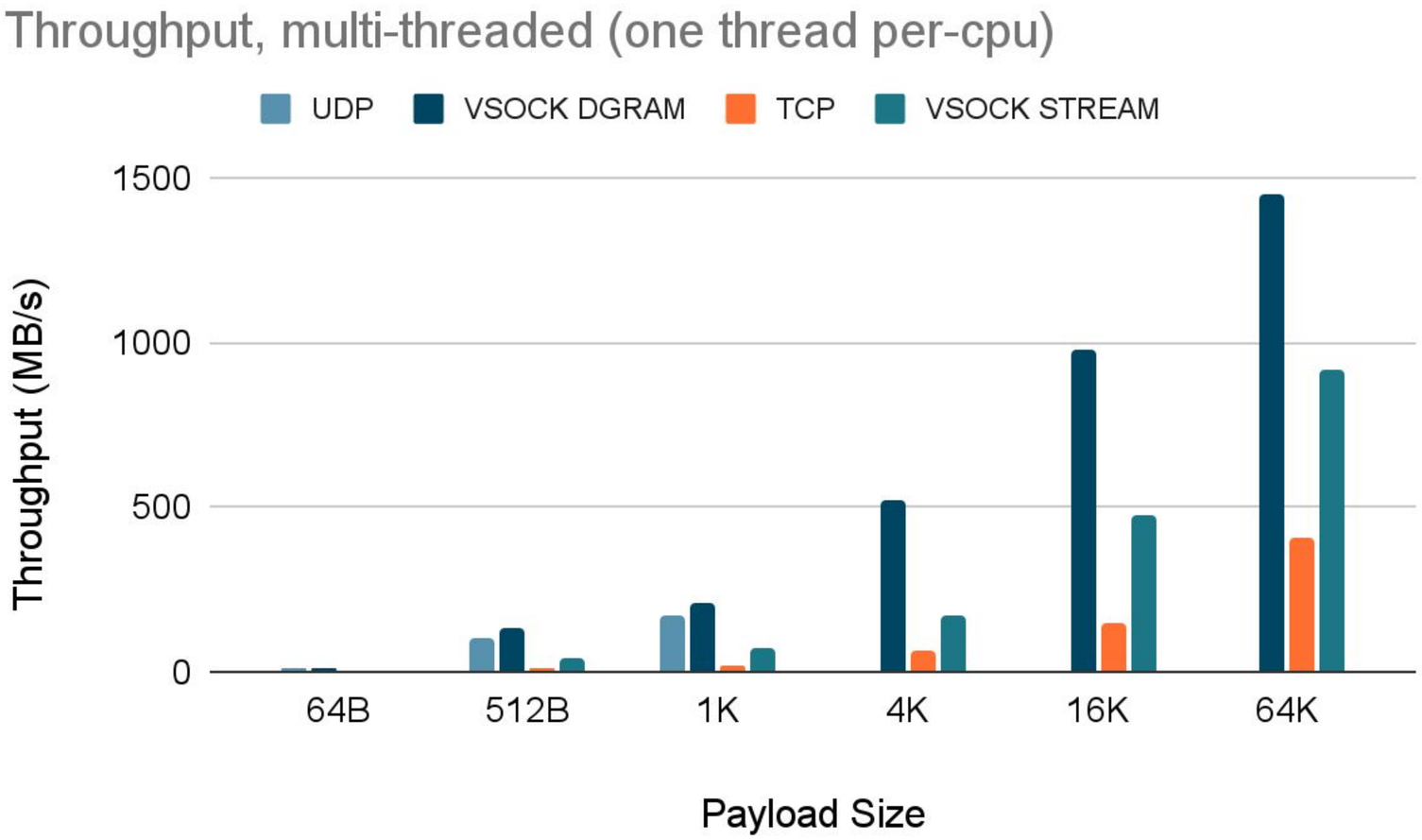
system	func name	avg latency (us)
vsock*	virtio_transport_send_pkt_work()	23
	vsock_dgram_sendmsg()	***8***
udp over virtio	start_xmit()	16
	udp_sendmsg() **	***72***

* a very important delay is missing from this chart: the workqueue delay (avg ~300 us on some test runs)
** udp_sendmsg() often calls start_xmit() directly, so the start_xmit() is sometimes included in udp_sendmsg()'s latency

Overall Comparison



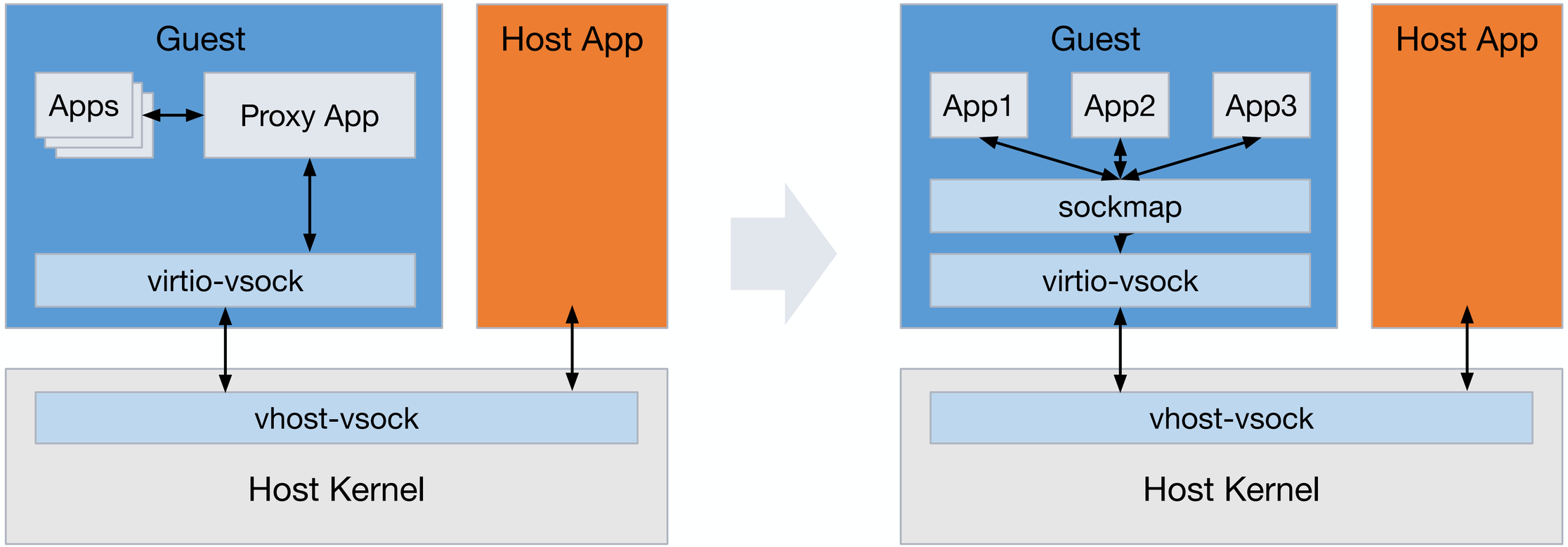
Overall Comparison



*udp only shown for <= 1K because excessive out of buffer errors for 1K > payloads

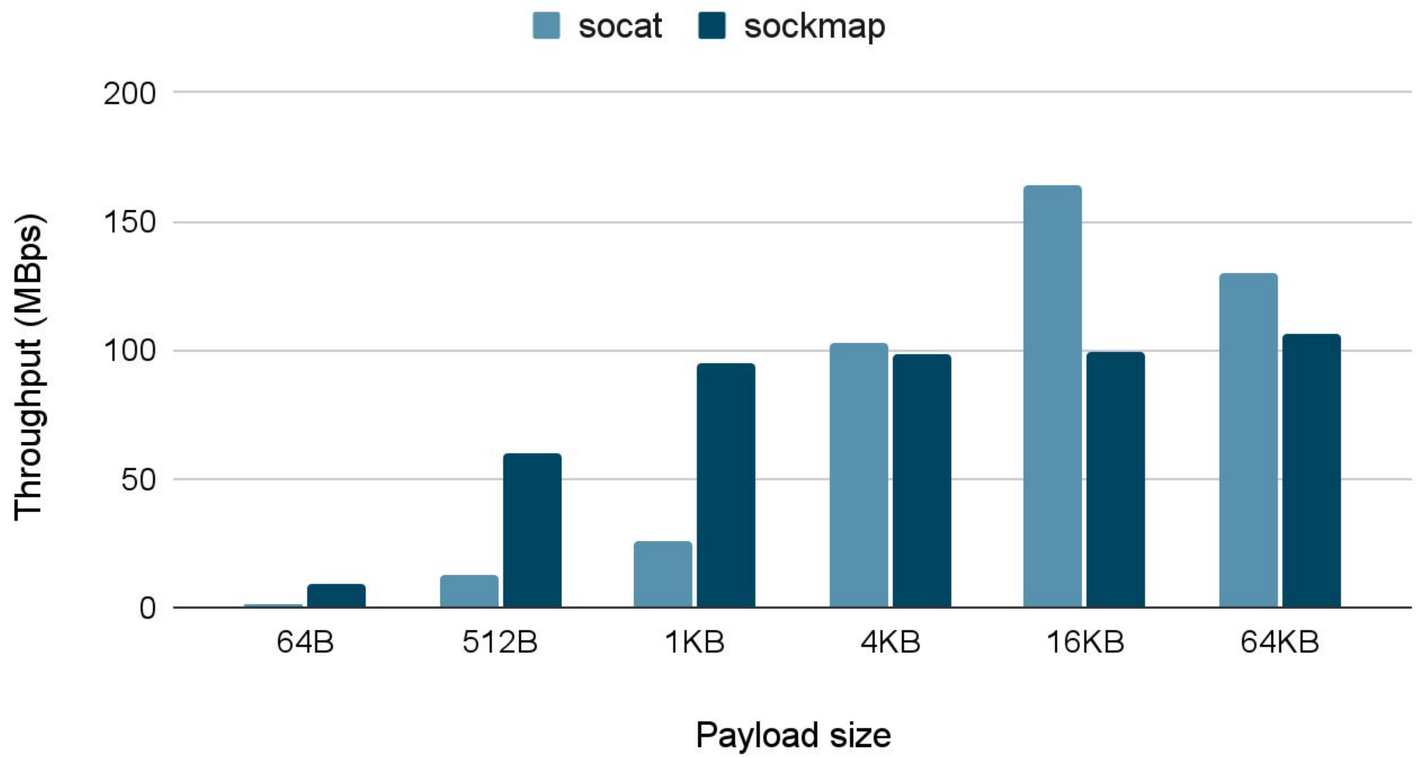
Recent Work: Sockmap for virtio-vsock

- Sockmap offers programmable skb redirection in the kernel space



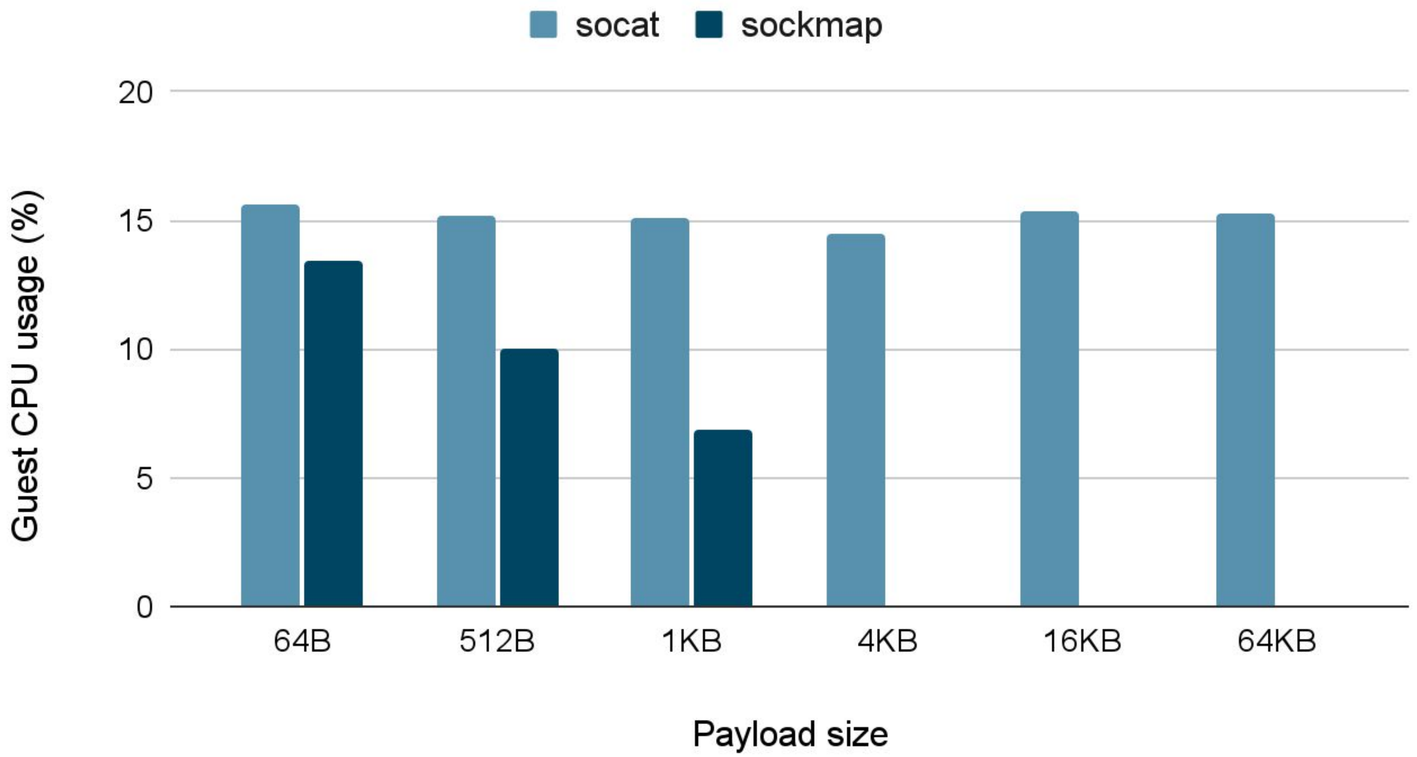
Sockmap removes overheads due to syscalls and data copying

- Comparison of different packet redirection: *socat* (8KB buffer) vs *sockmap*
- Guest: 8 vcpus, 1 single-thread client sending UDP traffic
- Host: 1 single-thread server receiving data from vsock



Sockmap removes overheads due to syscalls and data copying

- Comparison of different packet redirection: *socat* (8KB buffer) vs *sockmap*
- Guest: 8 vcpus, 1 single-thread client sending UDP traffic
- Host: 1 single-thread server receiving data from vsock



Future Work

- Finish upstreaming current patches
- Support multiple virtqueues?
  - Current protocol can't support this... introduce v2?
- General optimizations
  - Improve workqueue-induced delay
  - Improve lock duration and granularity

Summary

- VSOCK is a zero-configuration communication channel between host and guest
- Recent improvements in vsock make it more performant
- More optimizations to come

THANKS