Transparent Shared Memory Communications with eBPF

Cong Wang, A K M Fazla Mehrab
System Technologies & Engineering, ByteDance

ByteDance 字节跳动

Overview

- Problem: The overhead of TCP communication is high for co-locating applications
- Proposal:
  - Bypass TCP stack at socket layer
  - Use shared memory as the communication channel
  - Use eBPF to maintain application transparency
- Result:
  - We observed ~12% throughput improvement for container so far
  - There is still room for improvement

1. **Background**
2. Problem statement
3. State-of-the-art
4. Proposed solution
5. Summary

ByteDance字节跳动

Shared Memory

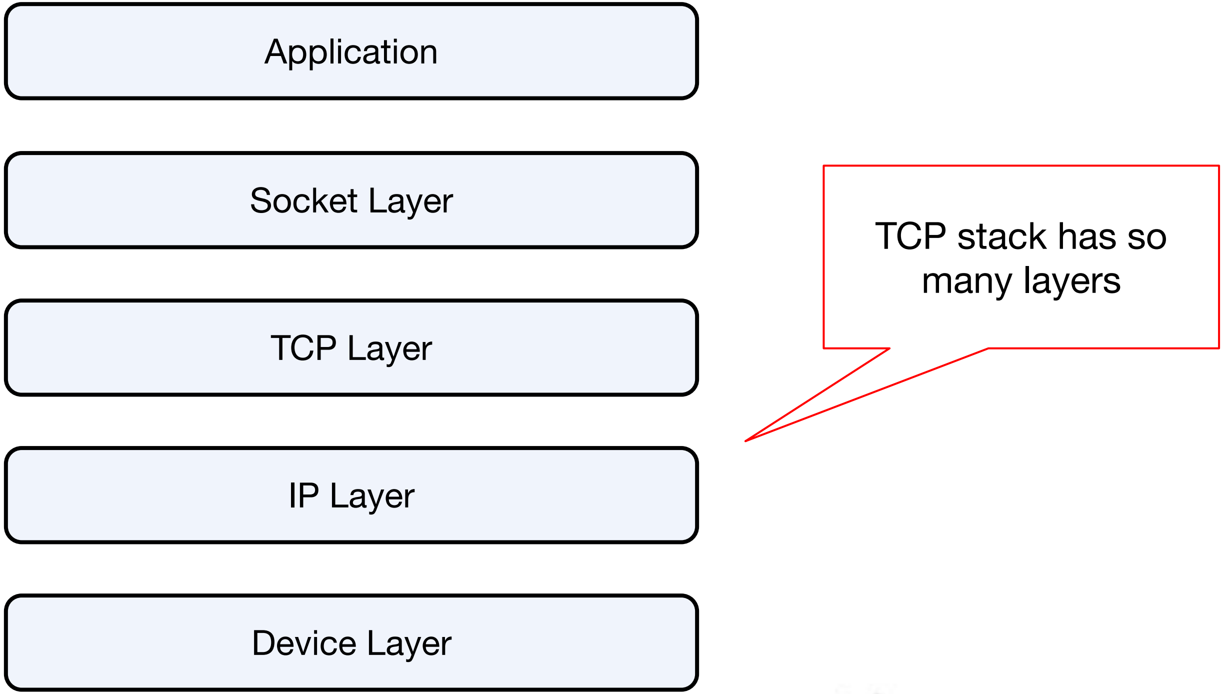
- Shared memory is the most effective communication we could achieve in the compute system
- Shared memory is common for IPC in OS
- For networking, RDMA technology implements this by allowing one system to share the memory of another system directly without involving the CPU
- Could we leverage shared memory for TCP in a single node?

Why still TCP?

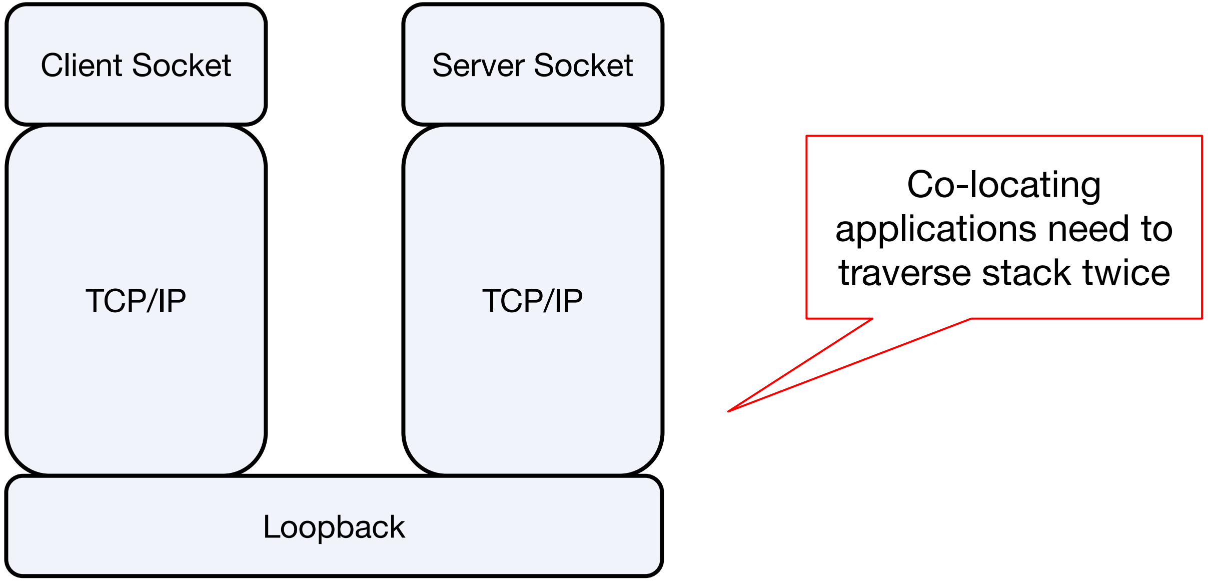
- TCP/IP is now the de facto standard for communication in data centers and across the internet as a whole
- It is the backbone of networking in large data centers due to its reliability, scalability, and compatibility with a wide range of hardware and software systems
- TCP is the protocol of choice for applications in data centers that require reliable, connection-oriented communication over IP networks
- Its widespread adoption and support across various programming languages and platforms make it well-suited for facilitating communication between applications
- TCP socket API is the de facto standard API for networking applications

1. Background
2. **Problem statement**
3. State-of-the-art
4. Proposed solution
5. Summary

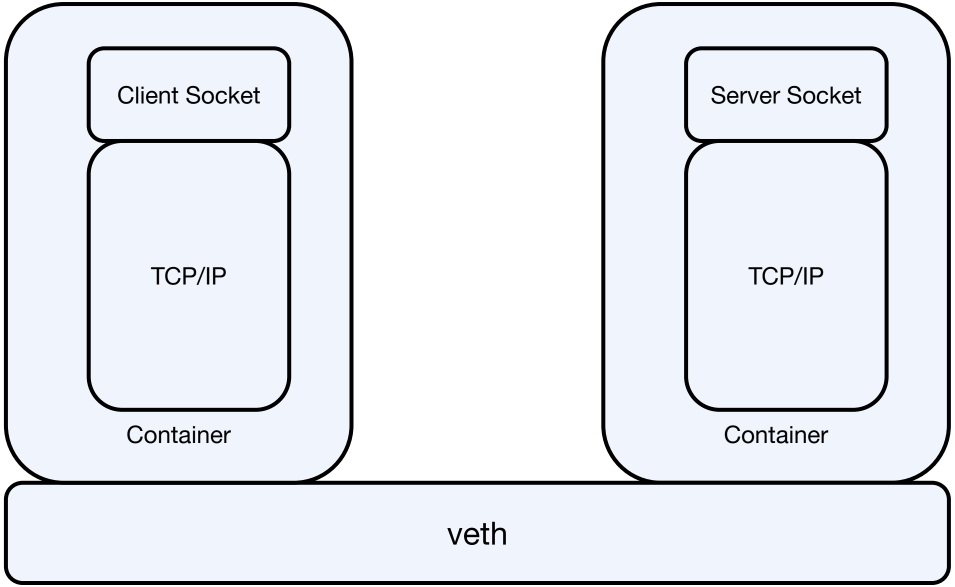
Problem Statement



Loopback

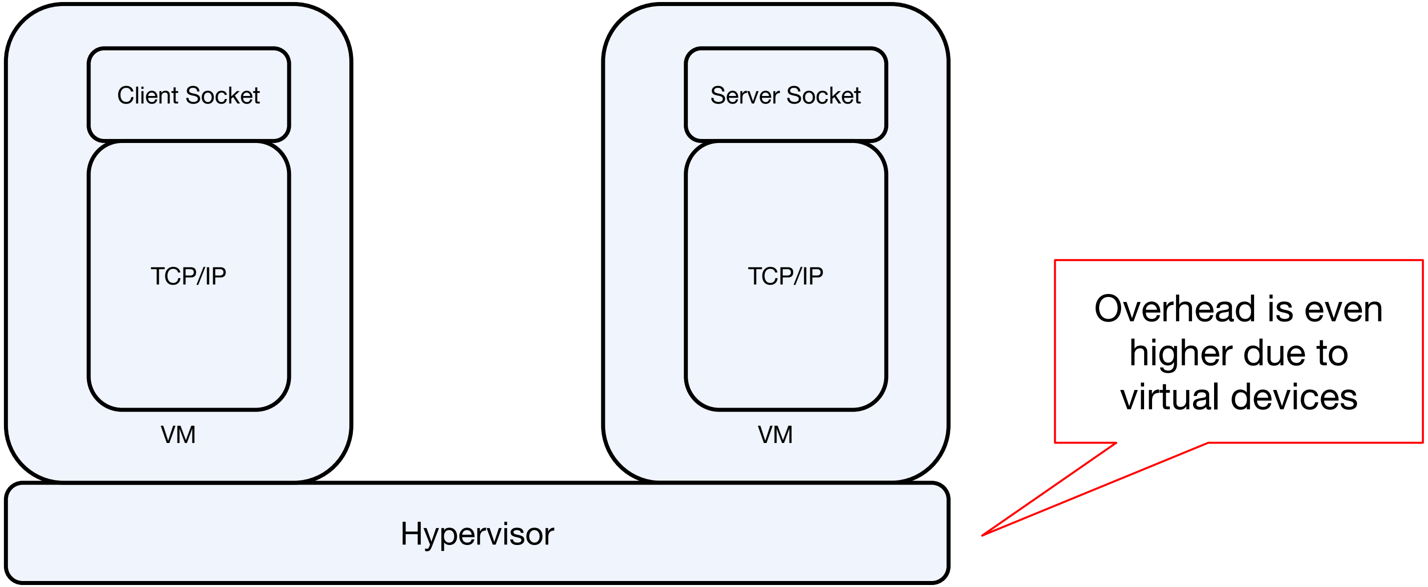

Client Socket
Server Socket
TCP/IP
TCP/IP
Loopback
Co-locating applications need to traverse stack twice

Co-locating Containers



Same scenario for containers

Co-resident VMs



ByteDance字节跳动

State-of-the-art

- Unix Domain Socket
- Virtual Socket
- RDMA
- SMC-R

Unix Domain Socket

- A method for inter-process communication (IPC) in Unix-like systems
- No stack, faster than TCP/IP for local communication
- Uses filesystem paths for socket addressing
- Supports bidirectional, streams or datagrams
- Utilized by system and user-level applications
- Operates in both connection-oriented and connectionless modes.
- Requires application modification

Virtual Socket

- Communication technology employed in virtualized and distributed environments
- Enables IPC between different VMs or containers
- Allows for reduced overhead and latency compared to traditional network-based communication
- Utilizes the host system's resources, bypassing the regular network stack
- Facilitates efficient, high-speed data transmission between distributed components
- Integral to microservices architecture and container orchestration platforms like Kubernetes
- Requires application modification

RDMA

- **Bypasses the OS**: RDMA allows data to be transferred directly between the RAM of different computers without CPU intervention, bypassing the OS and kernel entirely
- **Low Latency and High Throughput**: Direct data transfers significantly reduce latency and increase throughput, ideal for performance-critical applications
- **Zero-Copy Networking**: Enables zero-copy networking behavior, reducing the number of data copies between applications and the network stack
- **RDMA-Capable NICs**: Requires network interface cards that support RDMA, such as those implementing InfiniBand, RoCE (RDMA over Converged Ethernet), or iWARP (Internet Wide Area RDMA Protocol)
- **Verbs API**: RDMA offers a low-level "verbs" programming API that allows for fine-grained control over RDMA operations
- **Complexity**: RDMA programming and setup can be complex and may require a deep understanding of networking concepts

SMC-R

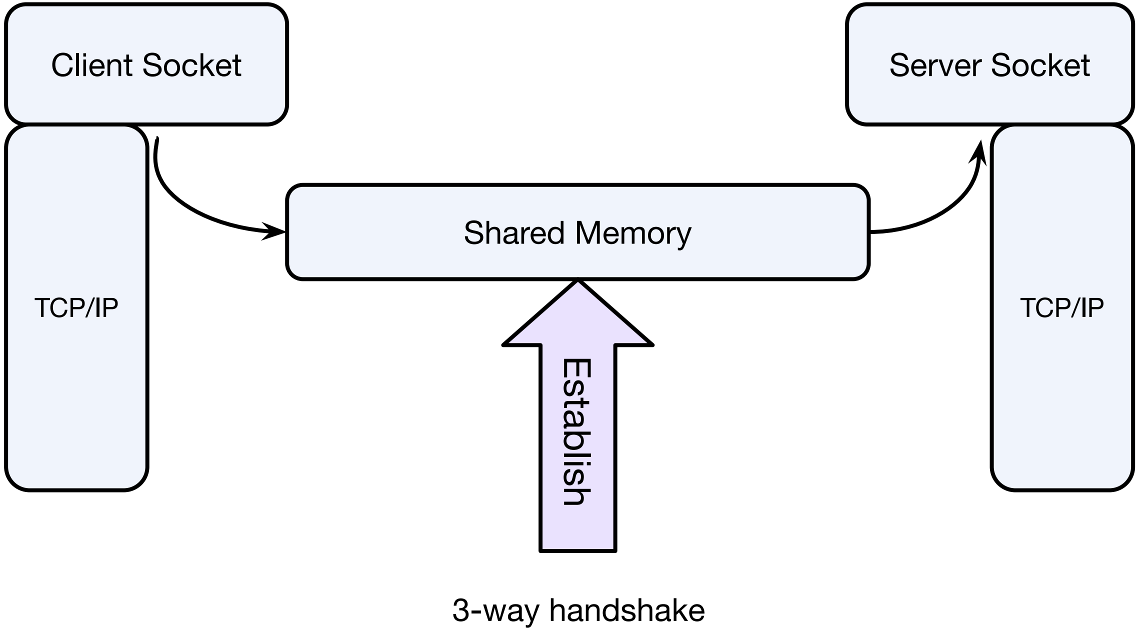
- **RDMA-based**: Utilizes RDMA for efficient data transfer, enabling high-speed communication between systems by bypassing the Linux kernel network stack
- **Low Latency**: Aims to reduce network latency compared to traditional TCP/IP, which is beneficial for latency-sensitive applications
- **TCP-compatibility**: Designed to be compatible with existing TCP/IP applications, allowing them to take advantage of RDMA-enabled hardware *via LD_PRELOAD*
- **Fallback to TCP**: If RDMA is not available or if setup negotiation fails, SMC-R automatically falls back to standard TCP/IP communication
- **Shared Memory**: Establishes a shared memory space between communication endpoints, allowing for efficient data exchange
- **RDMA-Capable NICs:** Requires network interfaces that support RDMA, such as those with InfiniBand or RoCE (RDMA over Converged Ethernet) capabilities

1. Background
2. Problem statement
3. State-of-the-art
4. **Proposed solution**
5. Summary

Our Idea

- Bypass TCP stack layers
  - To avoid overheads
  - With shared memory
  - Ideally zero-copy
- Maintain application transparency
  - To avoid specializations
  - To support all existing applications
- Inspirations from SMC
  - Exchange information during TCP 3-way handshake
- Inspirations from eBPF
  - Sockops
  - Sockmap

Our Idea


Client Socket
Server Socket
TCP/IP
Shared Memory
TCP/IP
Establish
3-way handshake

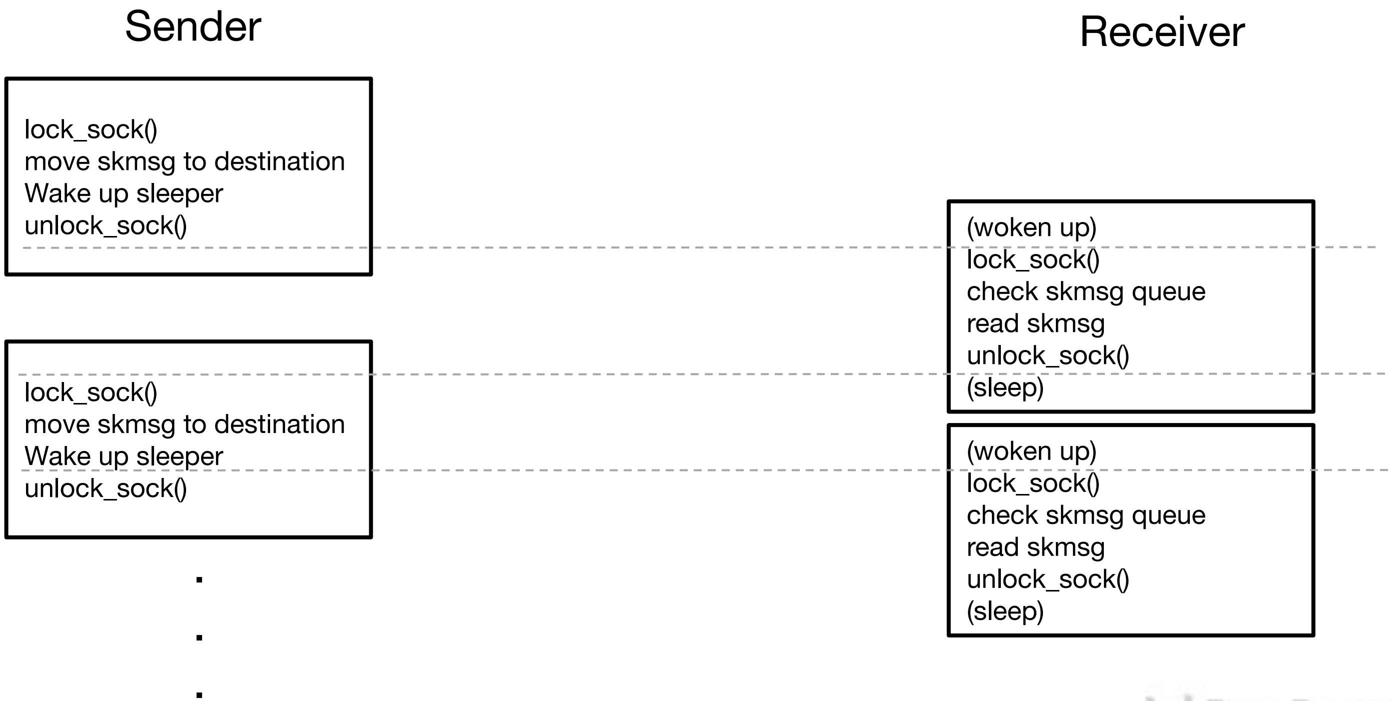
Non-VM Case

- Breaking it down into pieces:
    - Hijacking 3-way handshake: sockops
    - Communication channel: sockmap (sk_msg)
    - sendmsg() hook: BPF_SK_MSG_VERDICT for redirection
- It turns out Cilium already has a similar implementation: sockops-enable option
- Surprisingly, its performance is ***much worse*** than TCP!!

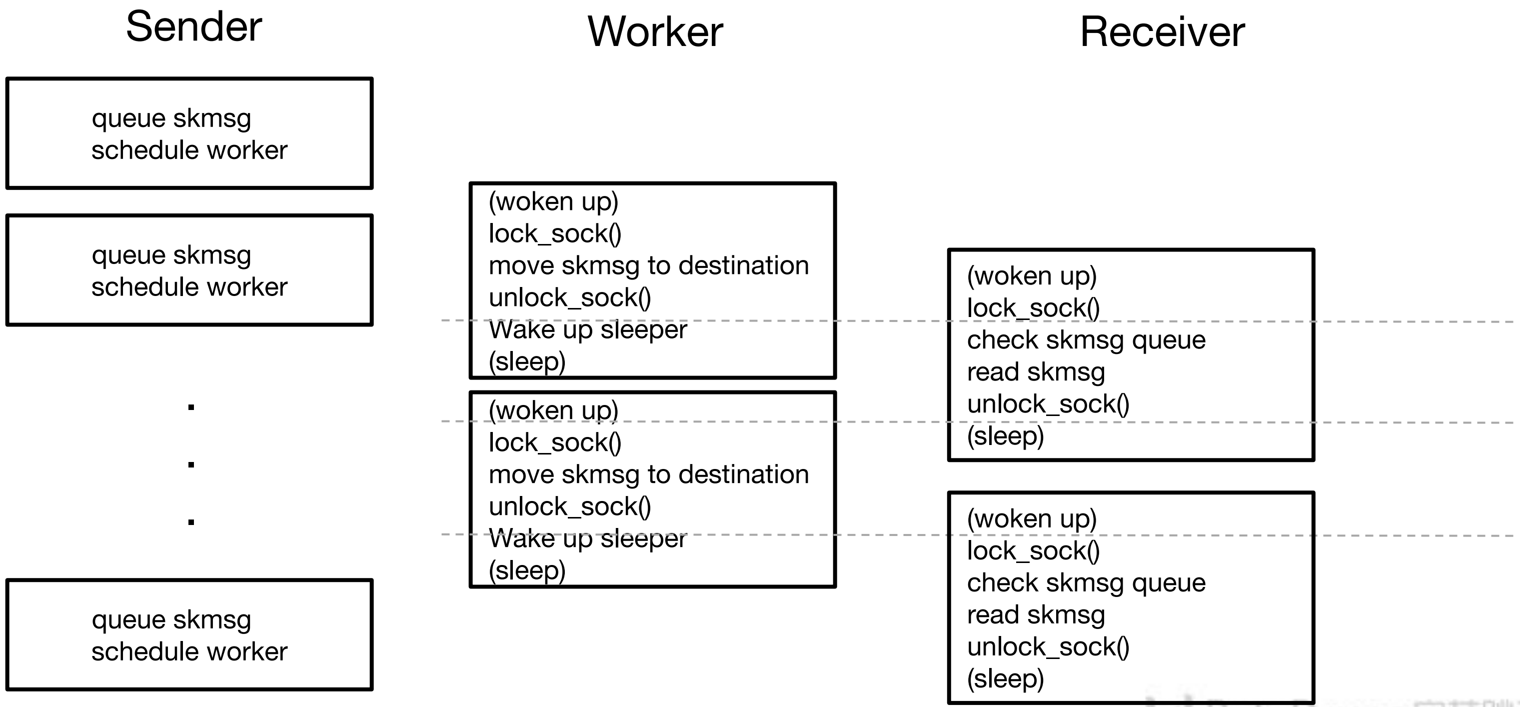
Cilium Socket Acceleration

- TCP is not as bad as it appears
- Linux TCP/IP stack has been optimized for decades, batching is excellent
- sk_msg is not optimized at all, not as sophisticated as skb at batching
    - For example, batching in release_sock()
- Sender needs to acquire receiver’s sock lock for accounting purpose
- Sock lock becomes the source of all evil
- We have an idea for optimization and Zijian Zhang already finished preliminary work

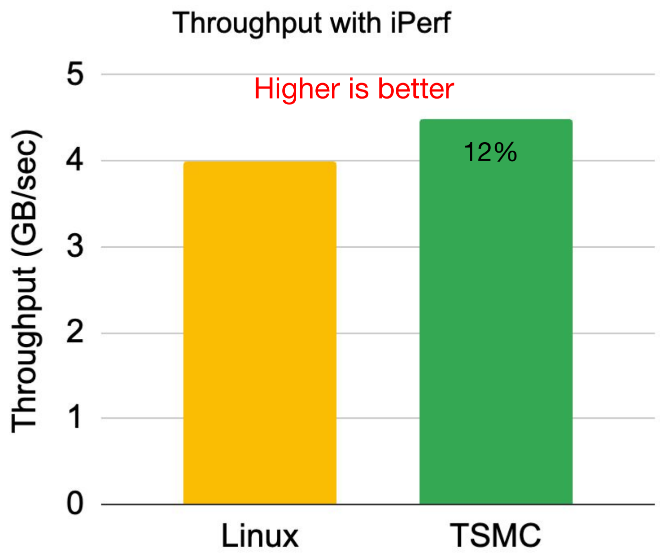
Optimizing sk_msg

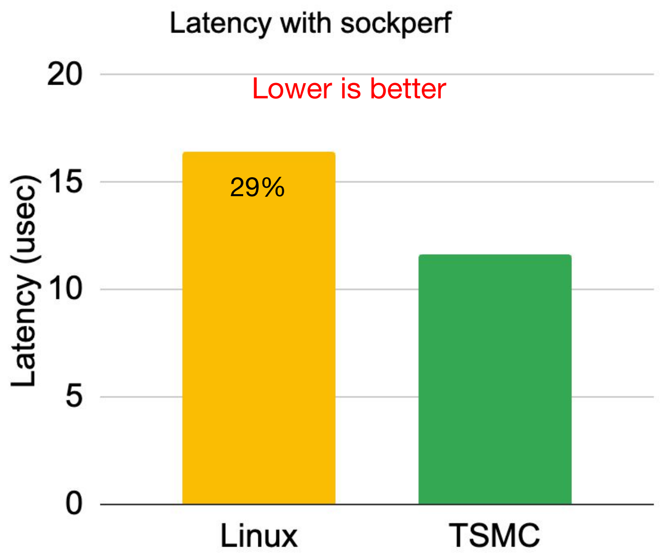


Optimizing sk_msg



Evaluation

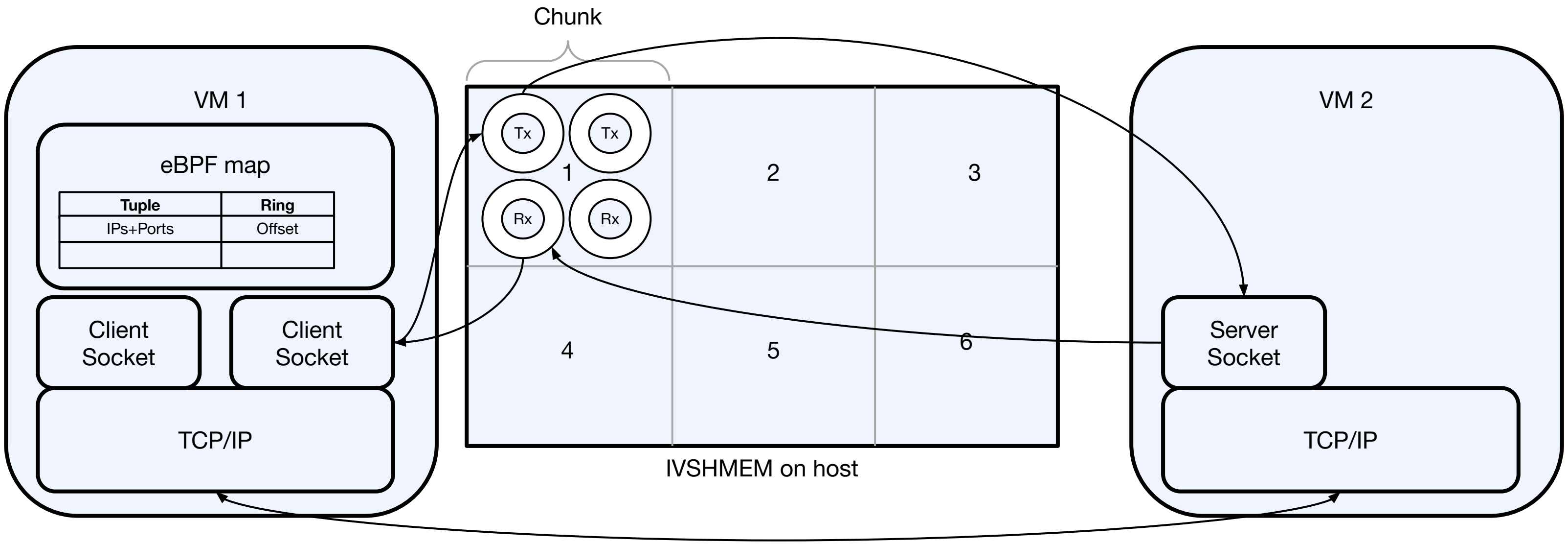

Throughput with iPerf
Higher is better
Throughput (GB/sec)
5
4
3
2
1
0
12%
Linux
TSMC


Latency with sockperf
Lower is better
Latency (usec)
20
15
10
5
0
29%
Linux
TSMC

VM Case

- Breaking it down into pieces:
  - Hijacking 3-way handshake: sockops with TCP option
  - Communication channel: IVSHMEM
  - sendmsg() and recvmsg() hooks: struct_ops for struct proto
- Security assumptions:
  - VMs run known workloads in our datacenter
  - Therefore, we trust all VMs running on the same host
- No prototype implementation available yet

VM case: overview



VM create

- Host IP
- Chunk offset and size

3-way handshake

- Host IP
- Tx and Rx offset

Sockops

- eBPF programs for handshake
    - Hook during SYN and ACK
    - Create Tx and Rx rings
    - Insert a new TCP option for discovery: Host IP and Ring info
    - Clean up rings if not on the same host

IVSHMEM

- IVSHMEM
  - Creates a shared memory segment
  - Exposes it to multiple VMs as PCI device
  - VMs can map into their address spaces
- An agent on hypervisor
  - Uses IVSHMEM
  - Divides that into chunks
  - Assigns chunks during VM initialization
  - Passes host IP to VM's
- BPF arena
  - Builds an eBPF storage on top of IVSHMEM chunks
  - Backend for ring buffers

struct_ops for struct proto

- We already have many hooks in tcp_bpf_sendmsg() for sockmap
- Introduce a new struct proto with struct_ops for more flexibilities
- Use eBPF programs to implement all TCP socket operations:
  - ->sendmsg()
  - ->recvmsg()
  - ->poll()
  - ->close()
- tcp_sendmsg_sm(), tcp_recvmsg_sm():
  - Use ring buffers for sending/receiving packets
- Build an infrastructure possibly for Homa/SMC/MPTCP too
  - Still retain TCP socket APIs

1. Background
2. Problem statement
3. State-of-the-art
4. Proposed solution
5. **Summary**

Summary

	Hardware Dependency	Application Transparency	Inter-Container Communication	Inter-VM Communication	Remote Communication
Unix Domain Socket	No	No	Yes (but requires shared filesystem)	No	No
Vsock	No	No	No	Yes	No
RDMA	Yes	No	No	No	Yes
SMC	No	Yes (but requires LD_PRELOAD)	No (still regular TCP)	No (but upstream is working on it)	Yes
Transparent Shared Memory Communication	No	Yes	Yes	Yes	No

Questions?

ByteDance字节跳动