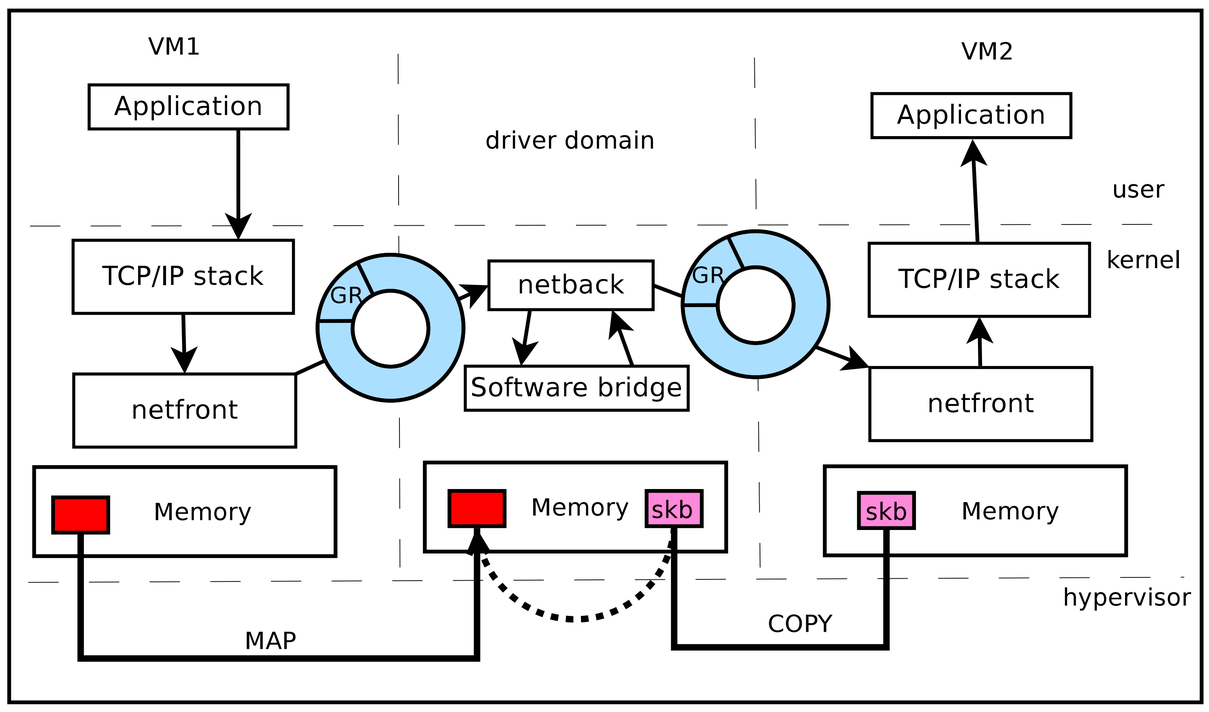
V4VSockets: low overhead intra-node communication in Xen

A. Nanos, S. Gerangelos, I. Alifieraki and N. Koziris
{ananos,sgerag,ialif,nkoziris}@cslab.ece.ntua.gr

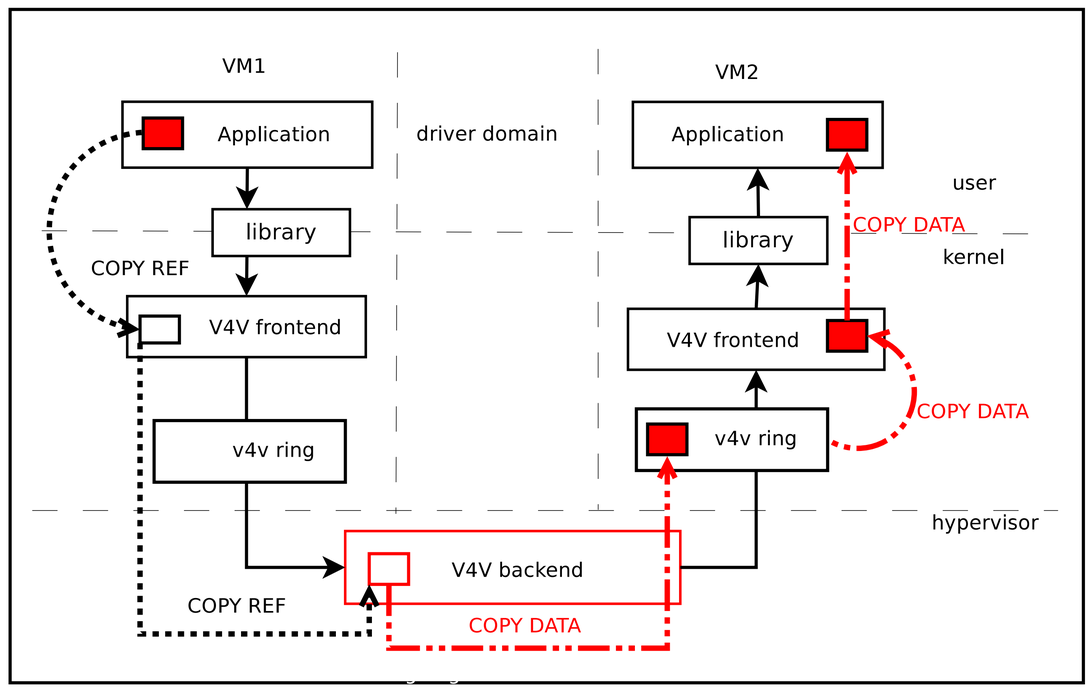
I/O Path in Xen (generic environment)



Intra-node communication suffers from severe overheads:

⇒ inefficient data paths
⇒ driver domain handles packet forwarding
⇒ unnecessary TCP/IP stack crossing and fragmentation

I/O Path in Xen with V4VSockets



V4VSockets is built as a full-stack protocol framework that supports p2p communication between VMs.

⇒ *Application layer*: the socket interface.
⇒ *Transport layer*: VM kernel driver.
⇒ *Network/Link layer*: the hypervisor, providing encapsulation of upper-layer messages to V4V messages, and packet delivery.

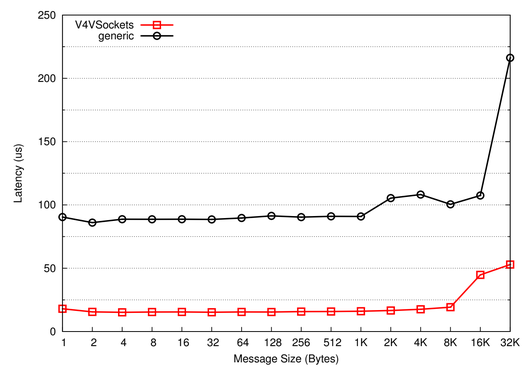
Key features

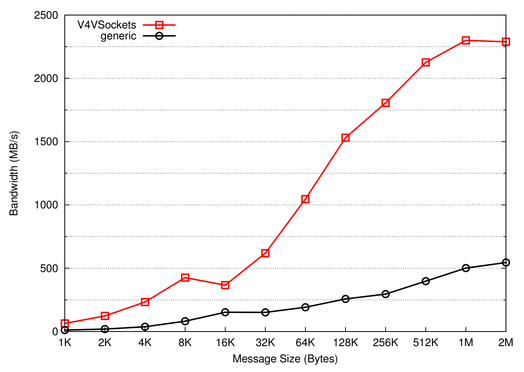
V4VSockets[a], is an efficient, socket-compliant, high performance intra node communication framework.
V4VSockets features:

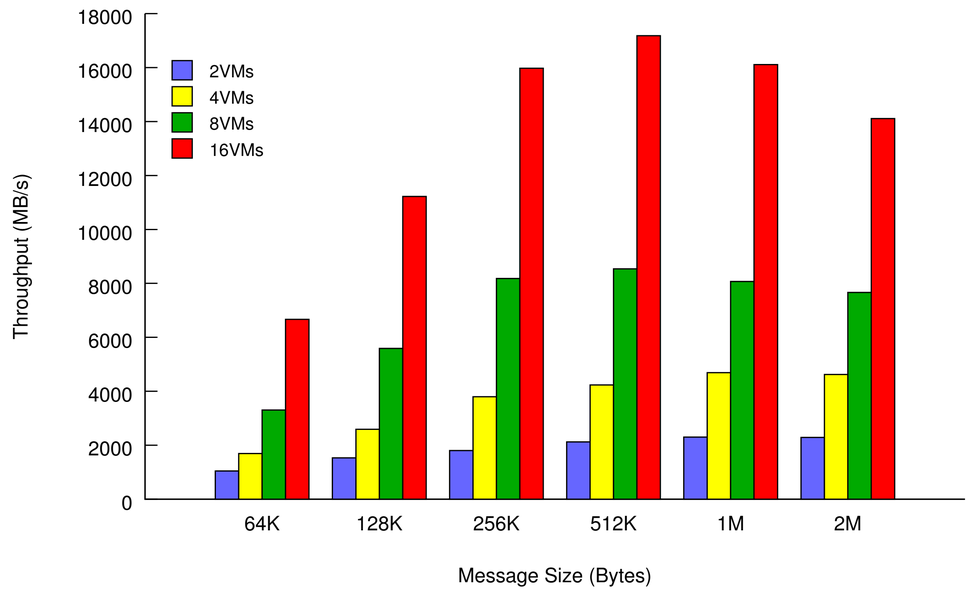
⇒ optimized data path. Data are copied from / to the VM kernel memory without the need to share pages between VMs.
⇒ no intermediary VM (driver domain), so no scheduling implications are involved.
⇒ no security implications, data cross the hypervisor and either get dropped or pushed forward through V4V semantics.
⇒ ultra low latency and high-bandwidth.

[a] https://github.com/HPSI/v4v

Ping-pong benchmark

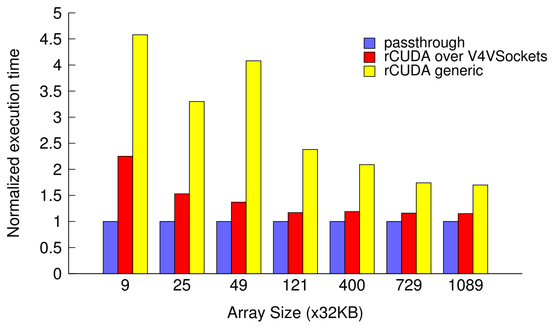






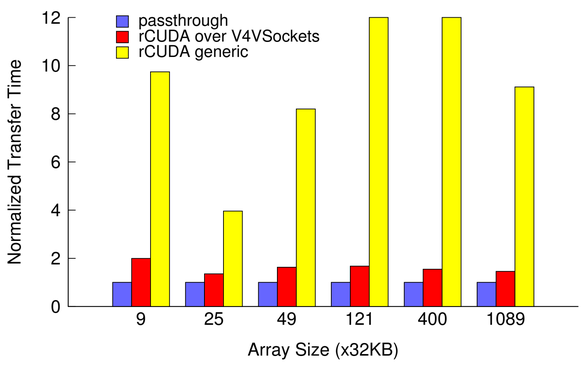
V4VSockets improves the latency for small messages by 81% compared to the generic case.
V4VSockets outperforms the default case for communication. V4VSockets is able to achieve 2299 MB/s, 4.5x better than the split driver, which performs poorly at 501 MB/s for 1 MB messages.
The aggregate throughput achieved by V4VSockets is shown below – we are able to reach more than half of the system's memory bandwidth (measured with stream @ 27GB/s) , bringing memory-copy-like bandwidth measurements to VM–to–VM message exchange.

GPU stencil performance through rCUDA



of V4VSockets to the improvement in the execution time, we plot the time needed to copy one of the input matrices from the machine's main memory to the GPU device memory. (essentially this is a `cudamemcpy()` call).

This experiment includes the following procedure: two copies of the input matrices from node's main memory to GPU device memory, the product execution on the GPU and finally one copy of the output matrix back to main memory. To elaborate more on the impact



Acknowledgments

The authors would like to thank the members of CSLab for the stimulating conversations that led to this work, especially Dr George Goumas, Dr Konstantinos Nikas and Nikela Papadopoulou. Additionally, many thanks to the anonymous reviewers for their useful comments and suggestions.

WiP

- Strengthen our implementation to a more user-friendly approach,
- Thoroughly examine the CPU utilization overheads imposed by V4VSockets,
- Polish the peer discovery framework to adaptively use V4VSockets over generic sockets.