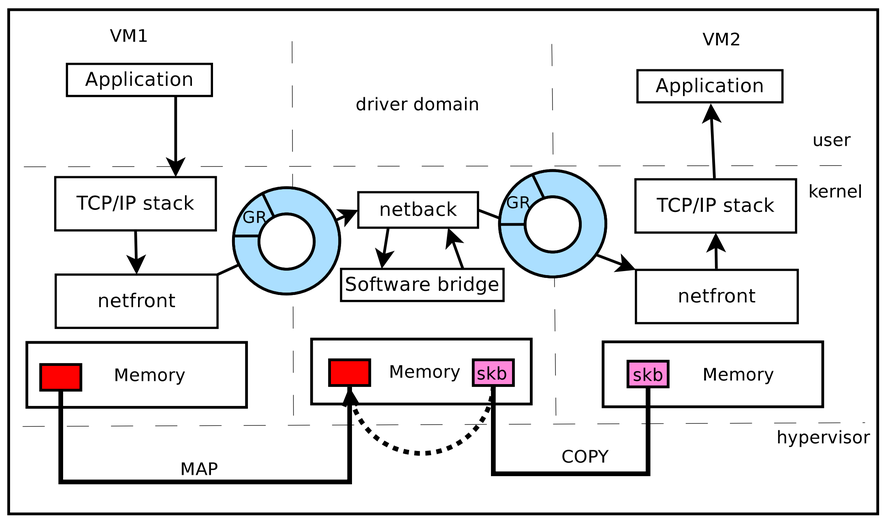
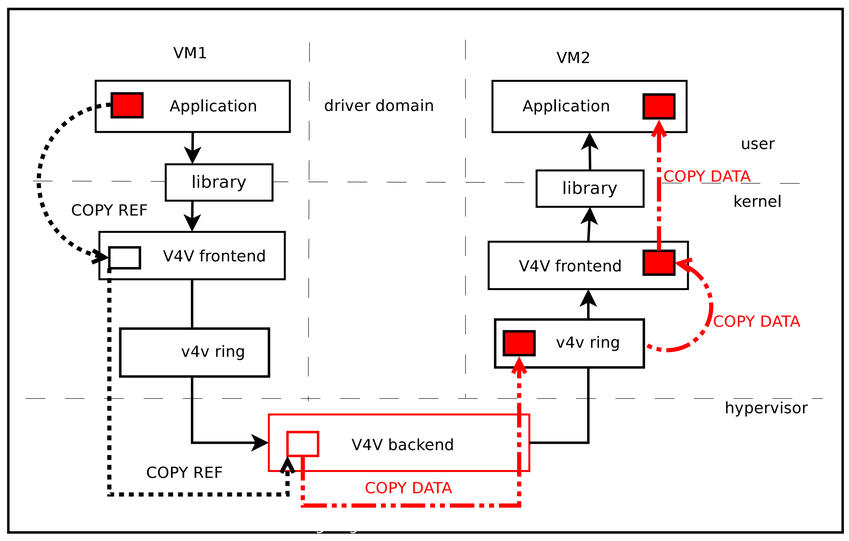
V4VSockets: low overhead intra-node communication in Xen

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

I/O Path in a Generic Xen environment



Intra-node communication suffers from severe overheads mostly due to:
⇒ inefficient data paths
⇒ driver domain intervention in packet forwarding / handling



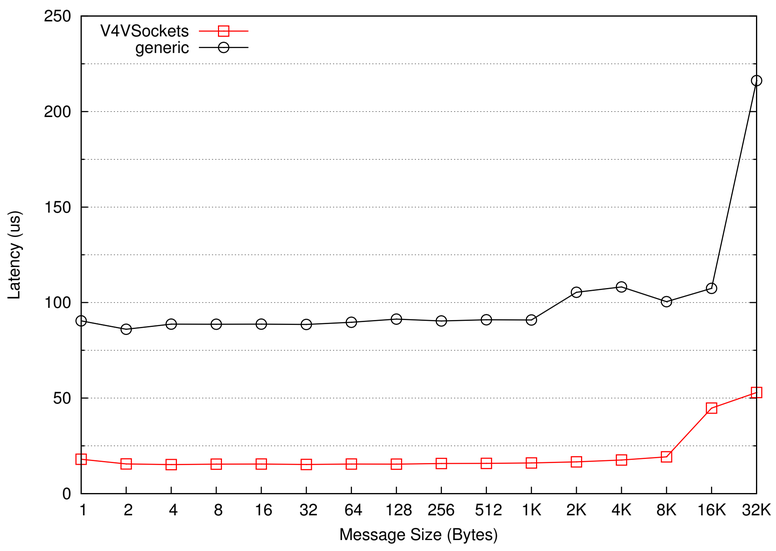
V4VSockets is built as a full-stack protocol framework that supports peer-to-peer communication between co-located VMs.
⇒ *Application layer*: the socket interface.
⇒ *Transport layer*: VM kernel driver.
⇒ *Network/Link layer*: the hypervisor, providing encapsulation of upper-layer messages to packets and packet delivery.

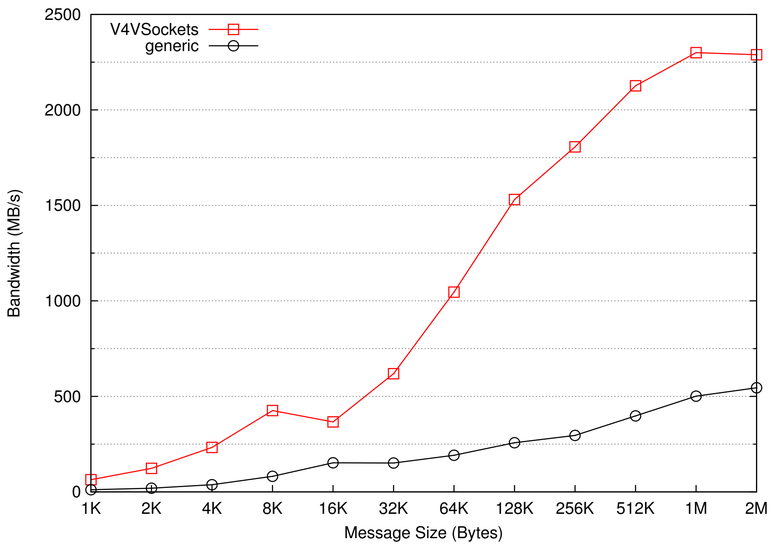
Contribution

⇒ V4VSockets[a], an efficient, socket-compliant, high–performance intra-node communication framework.
⇒ 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.
⇒ V4VSockets preliminary evaluation, using generic micro-benchmarks and compare it to conventional communication paths. V4VSockets outperforms the generic method of intra-node communication and scales efficiently with a large number of VMs both in terms of throughput as well as latency.
⇒ real-life use case: we deploy an HPC application stencil over rCUDA, a remote GPU execution stack over V4VSockets.

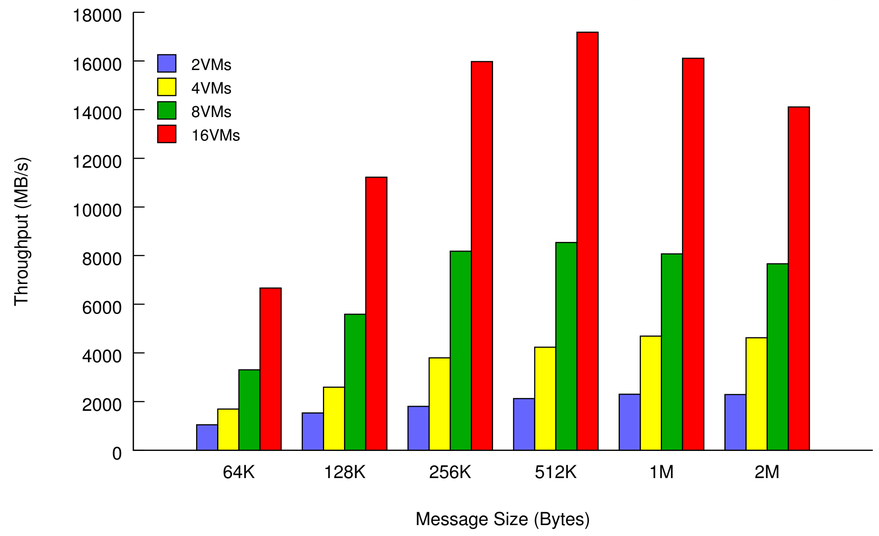
[a] https://github.com/HPSI/v4v

Preliminary results

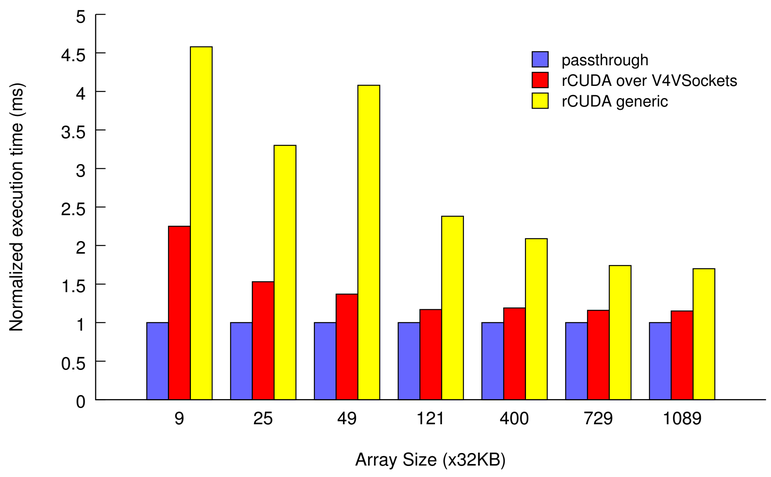


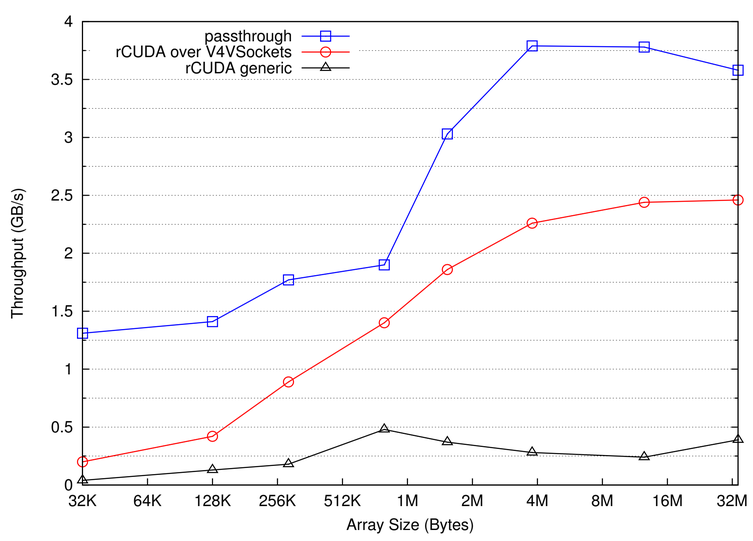


V4VSockets improves the latency for small messages by 81% compared to the generic case.
V4VSockets outperforms the default case for communication. V4VSockets is able to achieve 2299 MB/s, 4.5x better than the split driver, which performs poorly at 501 MB/s for 1 MB messages.
The aggregate throughput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



Total execution time of a generic GPU stencil and cudamemcpy throughput





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. We plot the normalized time of execution of the matrix-matrix product benchmark.
To elaborate more on the impact of V4VSockets to the improvement in the execution time, we plot the throughput achieved when copying one of the input matrices from the machine's main memory to the GPU device memory. (essentially this is a `cudamemcpy()` call).