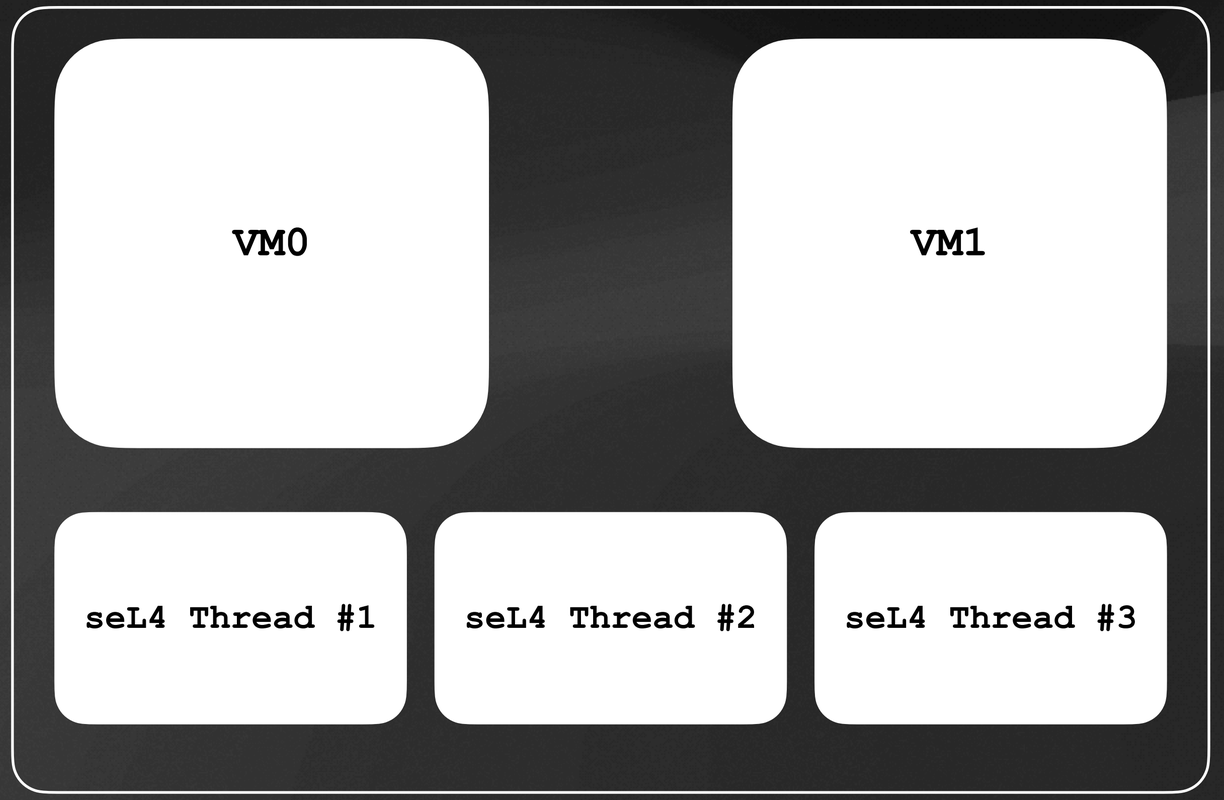
seL4: Experiences, Improvements, and Optimizations

Chris Guikema, DornerWorks

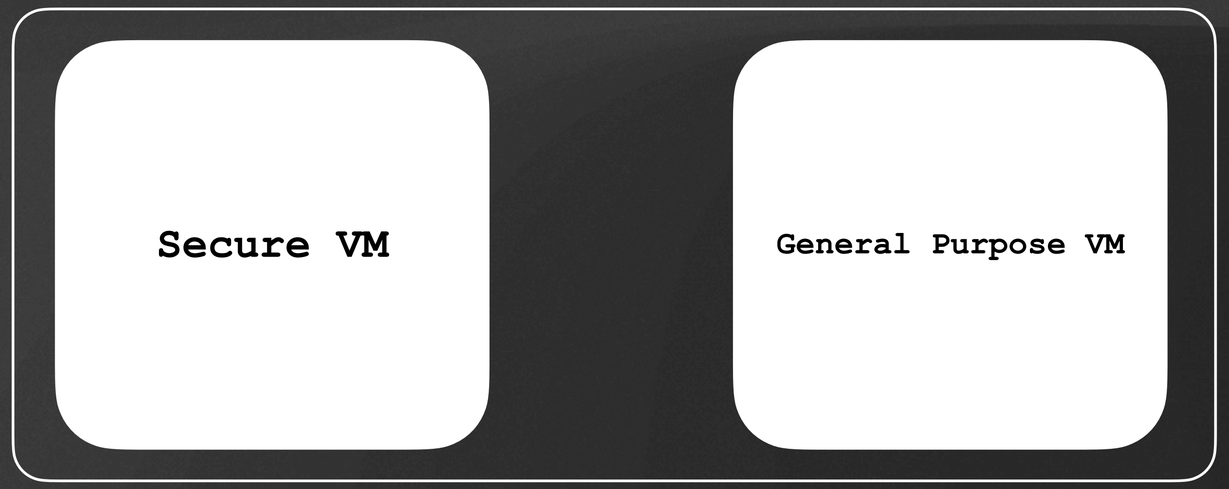
Common seL4 Use Cases

- Isolate legacy systems in Virtual Machines



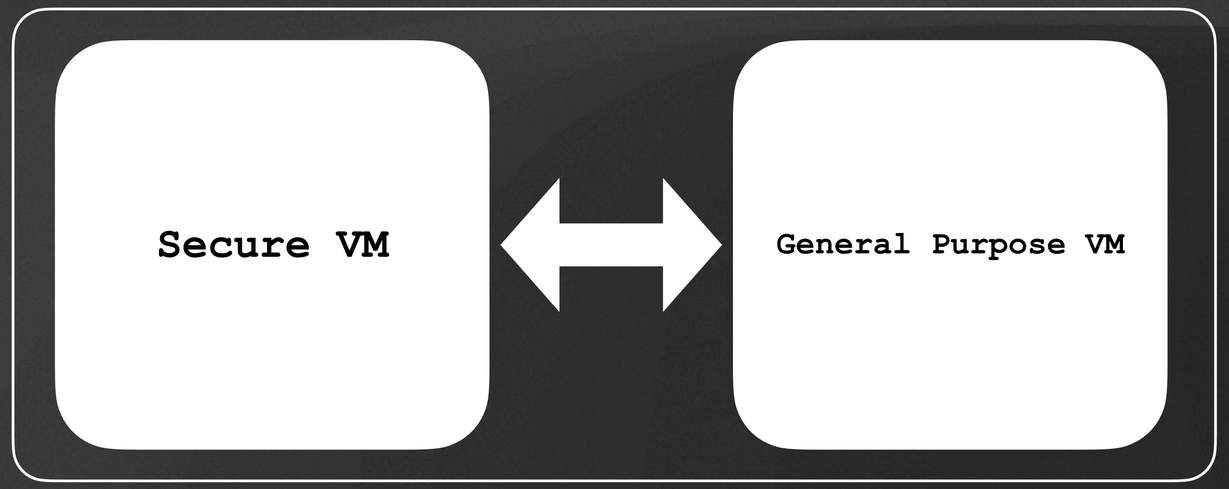
Common seL4 Use Cases

- Isolate legacy systems in Virtual Machines
- Cross-domain Solution



Common seL4 Use Cases

- Isolate legacy systems in Virtual Machines
- Cross-domain Solution



Virtio-Net

- Software implementation of a network device

Virtio-Net

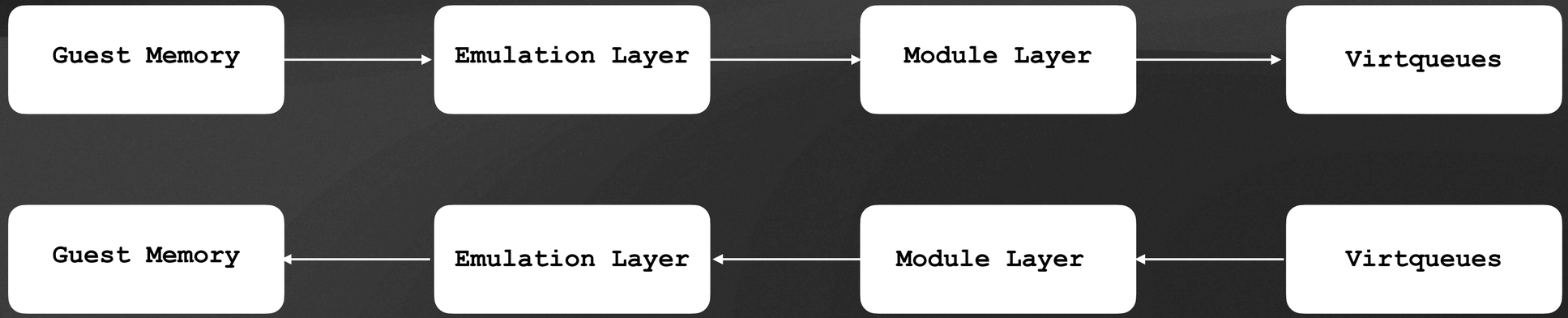
- Software implementation of a network device
- Leverage existing network stacks to communicate between VMs

Virtio-Net

- Software implementation of a network device
- Leverage existing network stacks to communicate between VMs
- Not Linux-specific

Virtio-Net

- Software implementation of a network device
- Leverage existing network stacks to communicate between VMs
- Not Linux-specific



How does it work?

From the guest’s perspective

- Host implements a virtual PCI bus with virtio-net device present

How does it work?

From the guest’s perspective

- Host implements a virtual PCI bus with virtio-net device present
- Guest scans PCI bus and loads virtio-net driver

How does it work?

From the guest's perspective

- Host implements a virtual PCI bus with virtio-net device present
- Guest scans PCI bus and loads virtio-net driver
- Standard network device available for use by any networking application

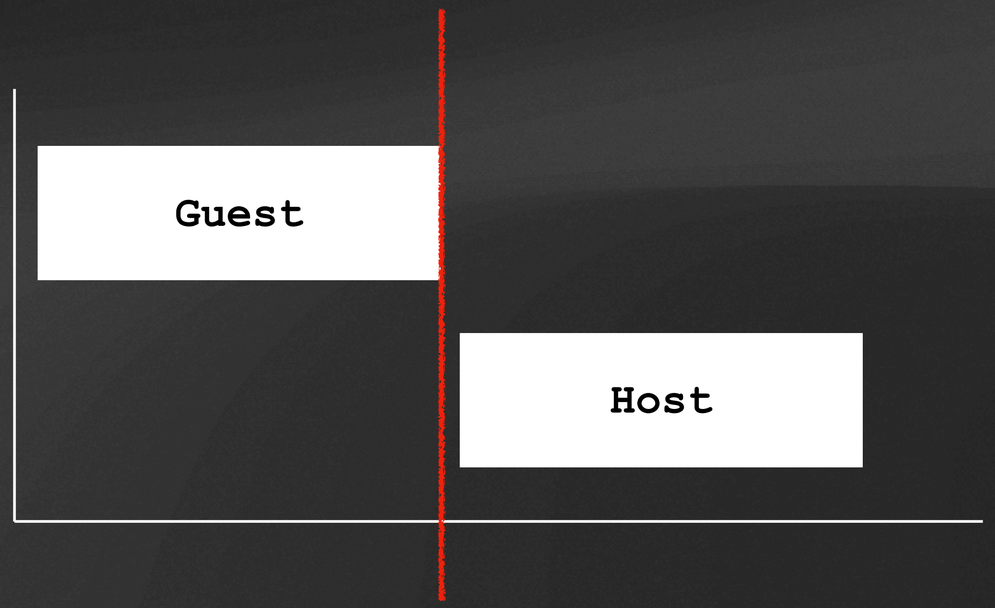
```
root@xilinx-zcu102-2021_1:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 15964 qdisc pfifo_fast qlen 1000
    link/ether 00:00:00:00:00:01 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:fe00:1/64 scope link
       valid_lft forever preferred_lft forever
3: sit0@NONE: <NOARP> mtu 1480 qdisc noop qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
```

How does it work?

- Host implements a virtual PCI bus with virtio-net device present

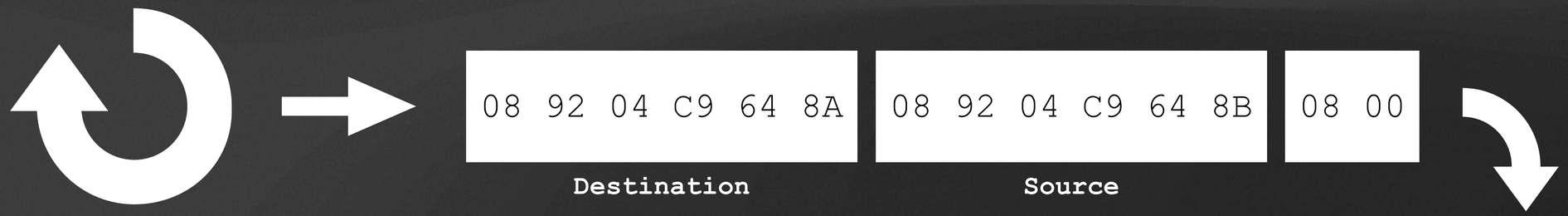
How does it work?

- Host implements a virtual PCI bus with virtio-net device present
- Guest accesses the PCI bus when transmitting a packet



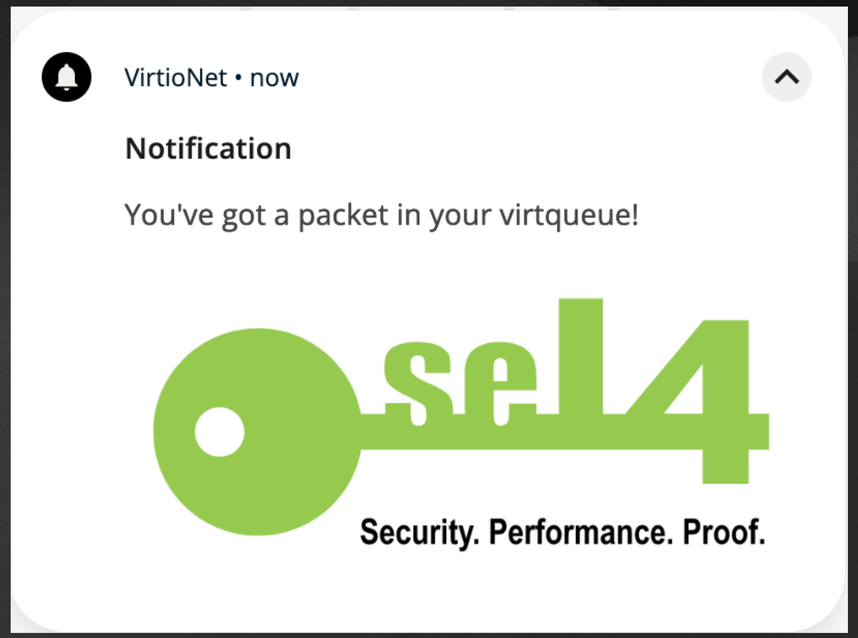
How does it work?

- Host implements a virtual PCI bus with virtio-net device present
- Guest accesses the PCI bus when transmitting a packet
- Host knows where the network packet is stored in memory



How does it work?

- Host receives a notification with a virtio-net specific badge

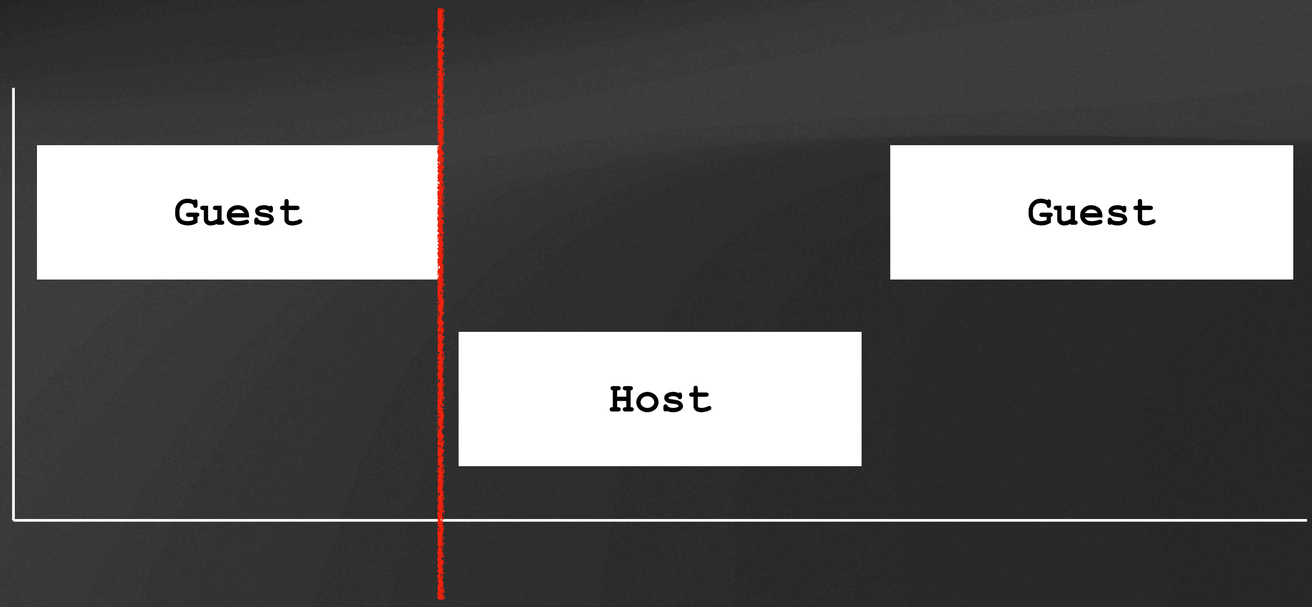


How does it work?

- Host receives a notification with a virtio-net specific badge
- Host reads the packet from the virtqueues and into guest memory

How does it work?

- Host receives a notification with a virtio-net specific badge
- Host reads the packet from the virtqueues and into guest memory
- Host injects an interrupt to the guest



How well does virtio-net work on seL4?

- Works great for simple communication path
  - pings, status updates, etc

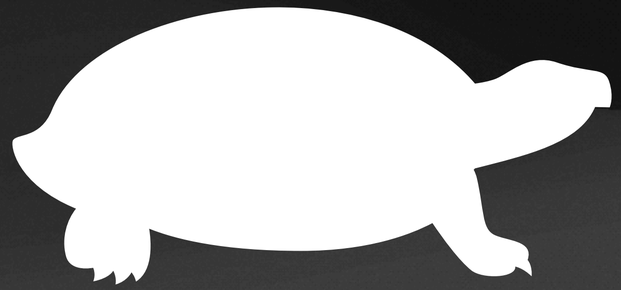
How well does virtio-net work on seL4?

- Works great for simple communication path
  - pings, status updates, etc
- Works poorly for throughput focused application

```
root@zcu102-zynqmp:~# iperf3 -c 192.168.1.2
Connecting to host 192.168.1.2, port 5201
[  5] local 192.168.1.1 port 39930 connected to 192.168.1.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  6.46 MBytes  54.2 Mbits/sec   14   60.8 KBytes
[  5]   1.00-2.00   sec  5.53 MBytes  46.4 Mbits/sec   26   43.8 KBytes
[  5]   2.00-3.00   sec  6.46 MBytes  54.2 Mbits/sec    5   35.4 KBytes
[  5]   3.00-4.00   sec  5.78 MBytes  48.5 Mbits/sec   13   39.6 KBytes
[  5]   4.00-5.00   sec  5.47 MBytes  45.9 Mbits/sec   28   7.07 KBytes
[  5]   5.00-6.00   sec  5.72 MBytes  48.0 Mbits/sec   29   31.1 KBytes
[  5]   6.00-7.00   sec  5.22 MBytes  43.8 Mbits/sec   37   14.1 KBytes
[  5]   7.00-8.00   sec  5.90 MBytes  49.5 Mbits/sec   13   32.5 KBytes
[  5]   8.00-9.00   sec  5.90 MBytes  49.5 Mbits/sec   14   65.0 KBytes
[  5]   9.00-10.00  sec  5.59 MBytes  46.9 Mbits/sec   24   8.48 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  58.0 MBytes  48.7 Mbits/sec  203             sender
[  5]   0.00-10.01  sec  57.4 MBytes  48.1 Mbits/sec                  receiver
```

Guest Memory Access

- Problem: accessing data from guest memory is *slow*



Guest Memory Access

- Problem: accessing data from guest memory is *slow*
- Upstream methodology:
  1. Find physical address to read from

Guest Memory Access

- Problem: accessing data from guest memory is *slow*
- Upstream methodology:
  1. Find physical address to read from
  2. Find capability associated with the Guest’s page

Guest Memory Access

- Problem: accessing data from guest memory is *slow*
- Upstream methodology:
  1. Find physical address to read from
  2. Find capability associated with the Guest’s page
  3. Use capability to map that page into Host’s address space

Guest Memory Access

- Problem: accessing data from guest memory is *slow*
- Upstream methodology:
  1. Find physical address to read from
  2. Find capability associated with the Guest’s page
  3. Use capability to map that page into Host’s address space
  4. Read/write information using that page

Guest Memory Access

- Problem: accessing data from guest memory is *slow*
- Upstream methodology:
  1. Find physical address to read from
  2. Find capability associated with the Guest’s page
  3. Use capability to map that page into Host’s address space
  4. Read/write information using that page
  5. Unmap the page from the Host’s address space

Virtio-Net Memory Access

- How often does the host need to access the guest's memory?
  - A *lot*!
- 4 functions:
  1. `ring_avail` reads guest memory once

```
uint16_t ring_avail(virtio_emul_t *emul, struct vring *vring, uint16_t idx)
{
    uint16_t elem;
    vm_guest_read_mem(emul->vm, &elem, (uintptr_t) & (vring->avail->ring[idx % vring->num]), sizeof(elem));
    return elem;
}
```

Virtio-Net Memory Access

- How often does the host need to access the guest's memory?
  - A *lot*!
- 4 functions:
  1. `ring_avail` reads guest memory once
  2. `ring_avail_idx` reads guest memory once

```
uint16_t ring_avail_idx(virtio_emul_t *emul, struct vring *vring)
{
    uint16_t idx;
    vm_guest_read_mem(emul->vm, &idx, (uintptr_t)&vring->avail->idx, sizeof(vring->avail->idx));
    return idx;
}
```

Virtio-Net Memory Access

- How often does the host need to access the guest's memory?
  - A *lot*!
- 4 functions:
  1. `ring_avail` reads guest memory once
  2. `ring_avail_idx` reads guest memory once
  3. `ring_desc` reads guest memory once

```
struct vring_desc ring_desc(virtio_emul_t *emul, struct vring *vring, uint16_t idx)
{
    struct vring_desc desc;
    vm_guest_read_mem(emul->vm, &desc, (uintptr_t) & (vring->desc[idx]), sizeof(desc));
    return desc;
}
```

Virtio-Net Memory Access

- How often does the host need to access the guest's memory?
  - A *lot*!
- 4 functions:
  1. `ring_avail` reads guest memory once
  2. `ring_avail_idx` reads guest memory once
  3. `ring_desc` reads guest memory once
  4. `ring_used_add` reads guest memory once, writes guest memory twice

```
void ring_used_add(virtio_emul_t *emul, struct vring *vring, struct vring_used_elem elem)
{
    uint16_t guest_idx;
    vm_guest_read_mem(emul->vm, &guest_idx, (uintptr_t)&vring->used->idx, sizeof(vring->used->idx));
    vm_guest_write_mem(emul->vm, &elem, (uintptr_t)&vring->used->ring[guest_idx % vring->num], sizeof(elem));
    guest_idx++;
    vm_guest_write_mem(emul->vm, &guest_idx, (uintptr_t)&vring->used->idx, sizeof(vring->used->idx));
}
```

Guest Memory Access - Transmit Path

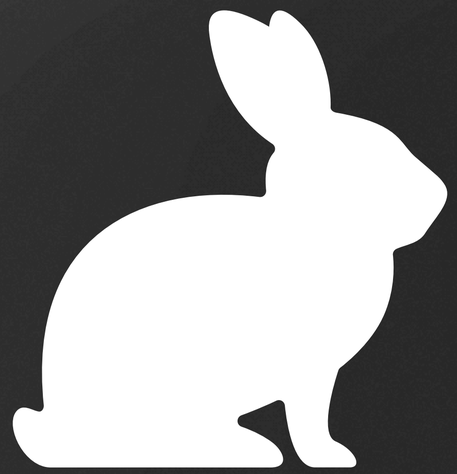
- `emul_notify_tx`:
  - 1x: `ring_avail_idx`, `ring_avail`, `ring_desc`
  - 1x: direct read of guest memory to get network packet
- `emul_tx_complete`:
  - 1x: `ring_used_add`
- Total:
  - 7 maps, 7 unmaps

Guest Memory Access - Receive Path

- `emul_rx_complete`:
  - 1x: `ring_avail_idx`, `ring_avail`, `ring_desc`
  - 1x: direct write of guest memory to give network packet
- Total:
  - 7 maps, 7 unmaps

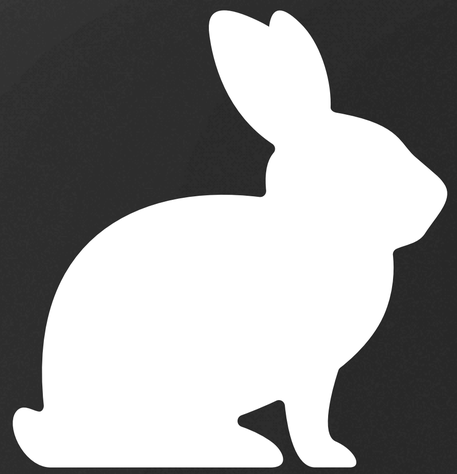
Can we speed this up?

- Old x86 libraries had a “Translation Vspace” implementation



Can we speed this up?

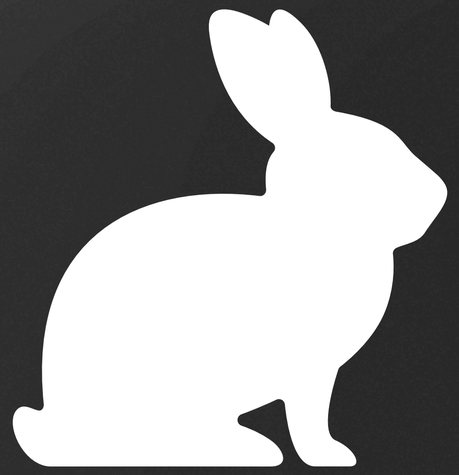
- Old x86 libraries had a “Translation Vspace” implementation
- Each page of guest memory remains mapped to the host



Can we speed this up?

- Old x86 libraries had a “Translation Vspace” implementation
- Each page of guest memory remains mapped to the host

How does this help?



Translation Vspace

Results

- Throughput improves by around 8x!

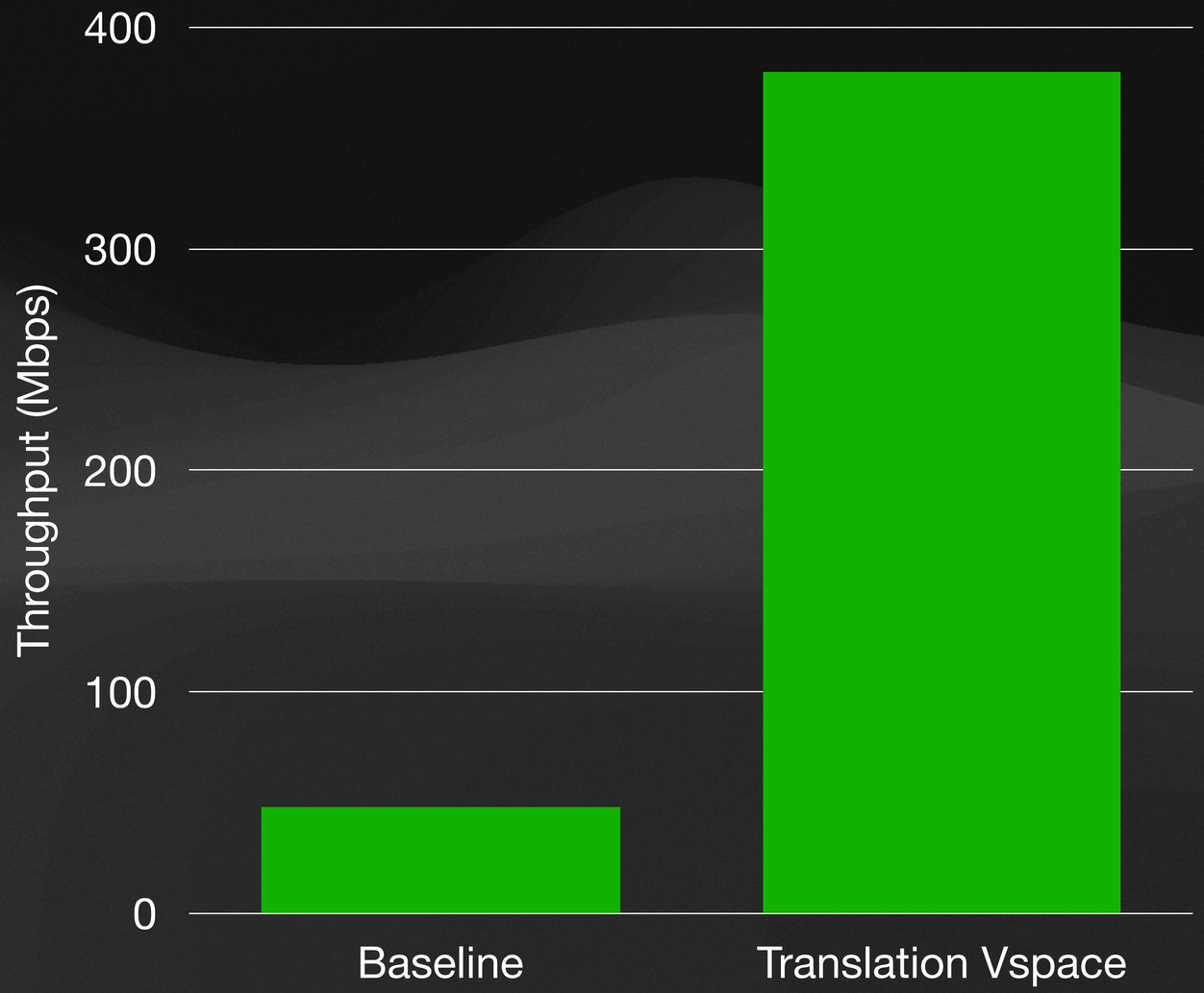
```
[  5] local 192.168.1.2 port 45814 connected to 192.168.1.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.01   sec  46.2 MBytes   385 Mbits/sec    0    199 KBytes
[  5]   1.01-2.01   sec  45.0 MBytes   378 Mbits/sec   10    132 KBytes
[  5]   2.01-3.02   sec  46.2 MBytes   382 Mbits/sec    0    164 KBytes
[  5]   3.02-4.01   sec  45.0 MBytes   381 Mbits/sec    1    156 KBytes
[  5]   4.01-5.01   sec  45.0 MBytes   381 Mbits/sec    4    150 KBytes
[  5]   5.01-6.02   sec  46.2 MBytes   381 Mbits/sec    1    147 KBytes
[  5]   6.02-7.02   sec  45.0 MBytes   381 Mbits/sec    0    189 KBytes
[  5]   7.02-8.02   sec  45.0 MBytes   375 Mbits/sec    5    163 KBytes
[  5]   8.02-9.01   sec  45.0 MBytes   381 Mbits/sec    0    195 KBytes
[  5]   9.01-10.01  sec  45.0 MBytes   380 Mbits/sec    0    198 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.01  sec   454 MBytes   380 Mbits/sec   21             sender
[  5]   0.00-10.01  sec   454 MBytes   380 Mbits/sec                  receiver
```

Translation Vspace

Results

- Throughput improves by around 8x!

```
[  5] local 192.168.1.2 port 45814 connected to 192.168.1.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.01   sec  46.2 MBytes   385 Mbits/sec    0    199 KBytes
[  5]   1.01-2.01   sec  45.0 MBytes   378 Mbits/sec   10    132 KBytes
[  5]   2.01-3.02   sec  46.2 MBytes   382 Mbits/sec    0    164 KBytes
[  5]   3.02-4.01   sec  45.0 MBytes   381 Mbits/sec    1    156 KBytes
[  5]   4.01-5.01   sec  45.0 MBytes   381 Mbits/sec    4    150 KBytes
[  5]   5.01-6.02   sec  46.2 MBytes   381 Mbits/sec    1    147 KBytes
[  5]   6.02-7.02   sec  45.0 MBytes   381 Mbits/sec    0    189 KBytes
[  5]   7.02-8.02   sec  45.0 MBytes   375 Mbits/sec    5    163 KBytes
[  5]   8.02-9.01   sec  45.0 MBytes   381 Mbits/sec    0    195 KBytes
[  5]   9.01-10.01  sec  45.0 MBytes   380 Mbits/sec    0    198 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.01  sec   454 MBytes   380 Mbits/sec   21             sender
[  5]   0.00-10.01  sec   454 MBytes   380 Mbits/sec                  receiver
```



With Great Throughput Comes…

More bugs

- Increasing the throughput introduced a number of bugs into the system that aren’t seen at lower speeds

With Great Throughput Comes…

More bugs

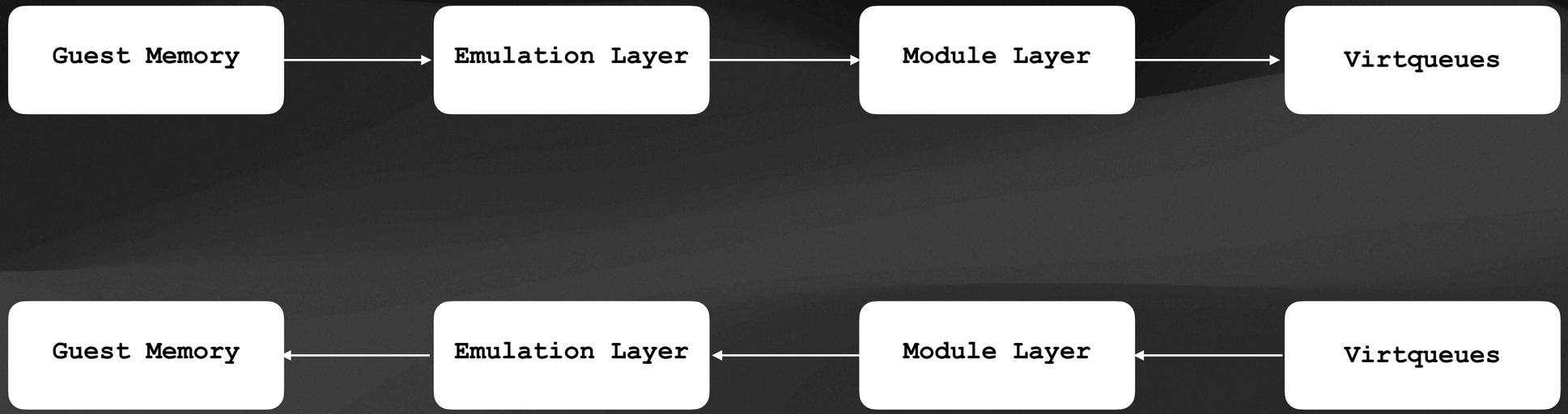
- Increasing the throughput introduced a number of bugs into the system that aren’t seen at lower speeds
- Most were simple to fix:
  - More data going through virtqueues -> increase virtqueue size, etc.

With Great Throughput Comes…

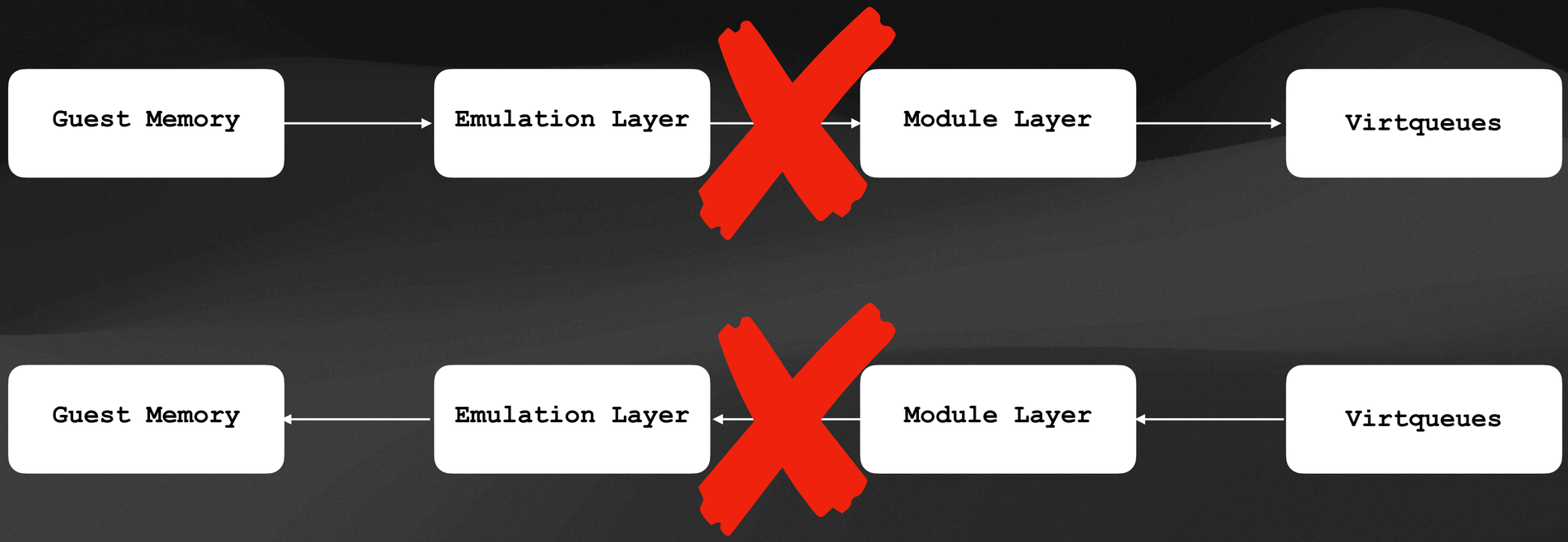
More bugs

- Increasing the throughput introduced a number of bugs into the system that aren’t seen at lower speeds
- Most were simple to fix:
  - More data going through virtqueues -> increase virtqueue size, etc.
- Others, not so much:
  - Increased throughput clobbers the cache and can lead to descriptor corruption

Optimizing the Virtio-Net Driver



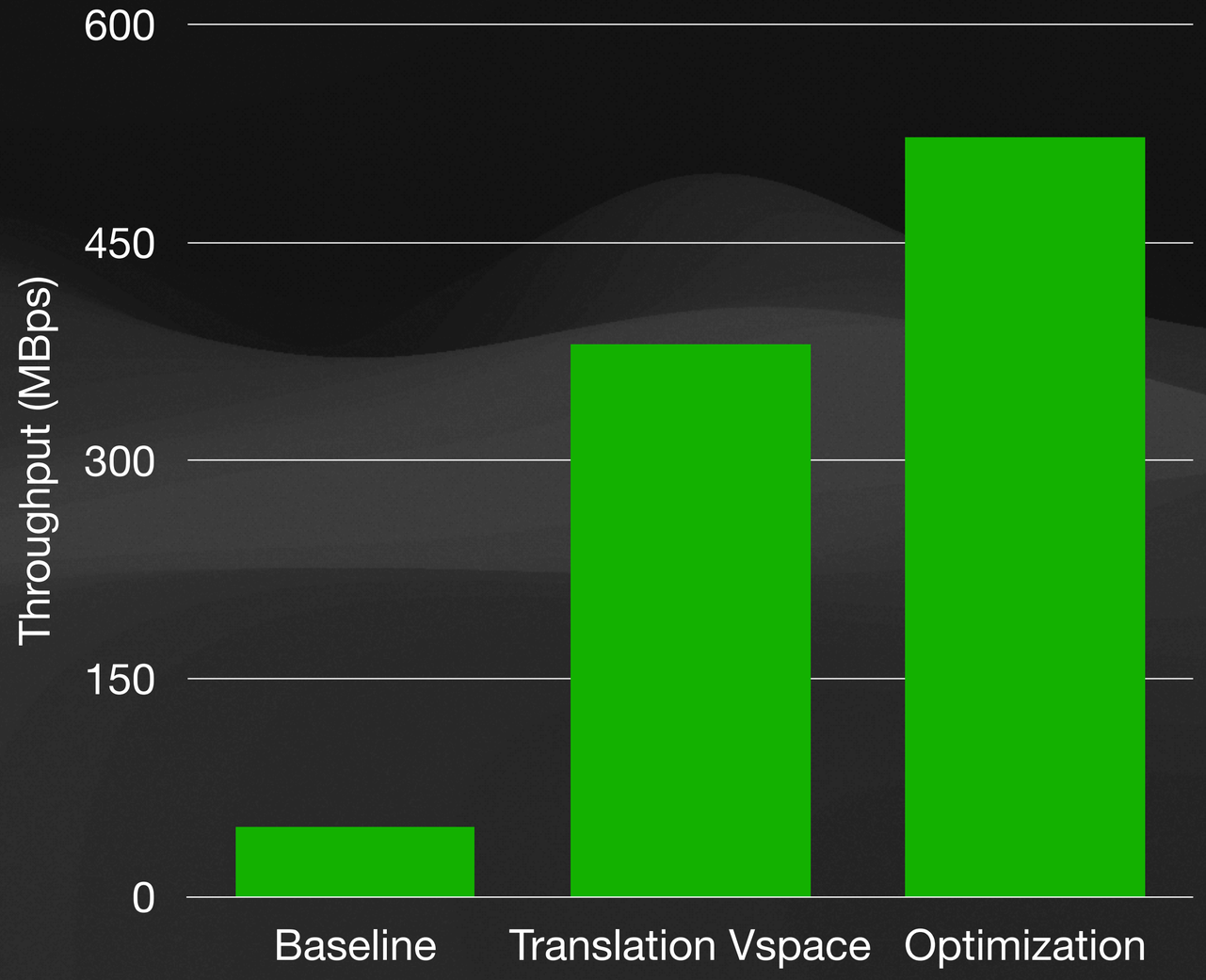
Optimizing the Virtio-Net Driver



Optimized Memcpy

Results

```
Connecting to host 192.168.1.100, port 5201
[  5] local 192.168.1.101 port 36190 connected to 192.168.1.100 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.01   sec  64.9 MBytes   541 Mbits/sec    0    260 KBytes
[  5]   1.01-2.02   sec  62.4 MBytes   518 Mbits/sec    0    273 KBytes
[  5]   2.02-3.01   sec  61.2 MBytes   521 Mbits/sec    0    287 KBytes
[  5]   3.01-4.02   sec  62.5 MBytes   519 Mbits/sec    0    287 KBytes
[  5]   4.02-5.00   sec  61.2 MBytes   519 Mbits/sec    0    287 KBytes
[  5]   5.00-6.01   sec  62.5 MBytes   520 Mbits/sec    0    287 KBytes
[  5]   6.01-7.02   sec  62.5 MBytes   521 Mbits/sec    0    287 KBytes
[  5]   7.02-8.01   sec  61.2 MBytes   520 Mbits/sec    0    287 KBytes
[  5]   8.01-9.01   sec  62.5 MBytes   520 Mbits/sec    0    287 KBytes
[  5]   9.01-10.00  sec  61.2 MBytes   520 Mbits/sec    0    287 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec   622 MBytes   522 Mbits/sec    0             sender
[  5]   0.00-10.01  sec   622 MBytes   521 Mbits/sec                  receiver
```

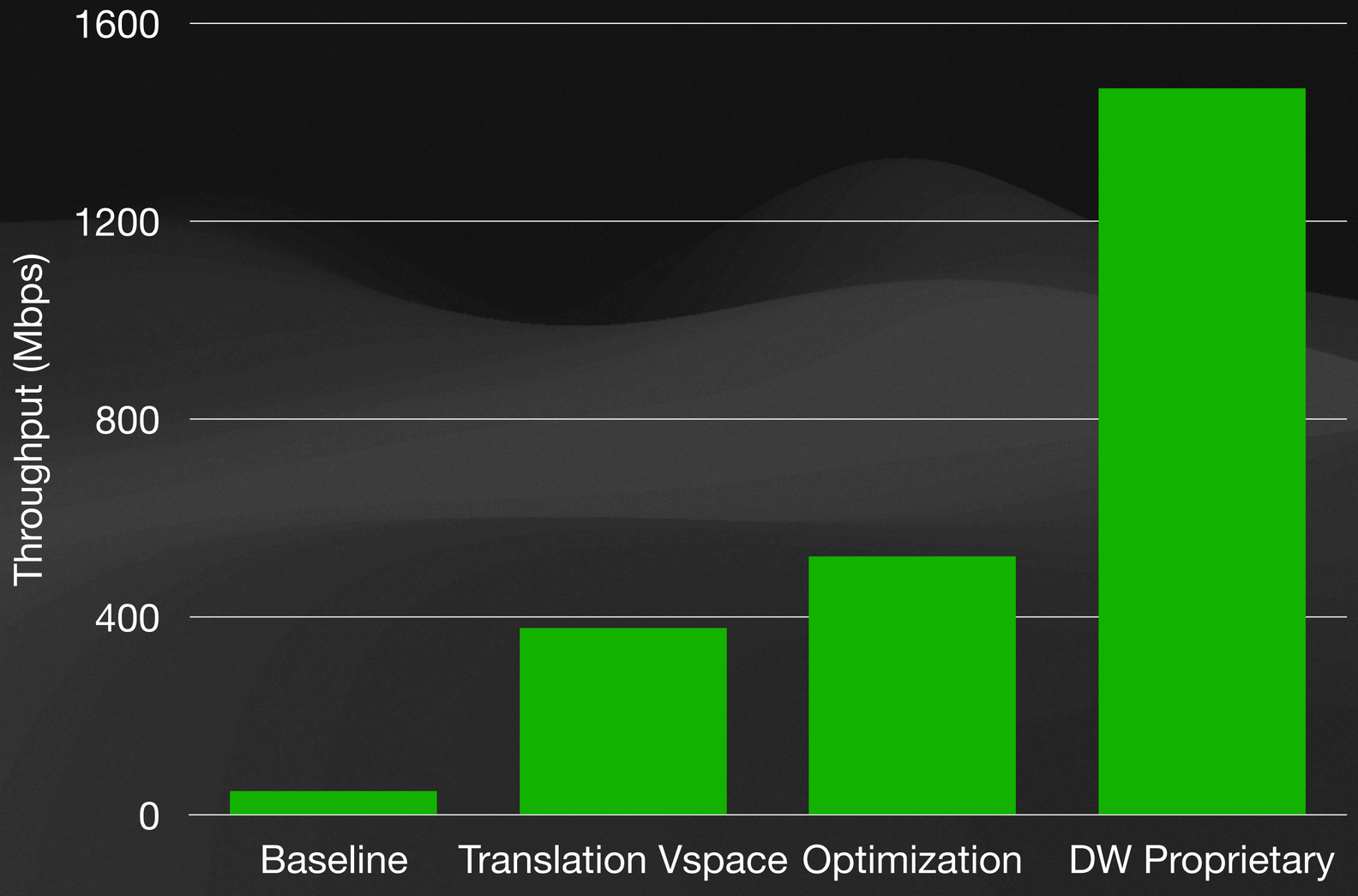


What’s next?

- DornerWorks is open-sourcing the Translation Vspace and `memcpy` optimizations
- We also developed a further improvement available for purchase…

What’s next?

```
root@xilinx-zcu102-2021_1:~# iperf3 -c 192.168.1.2
Connecting to host 192.168.1.2, port 5201
[  5] local 192.168.1.1 port 58794 connected to 192.168.1.2 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.01   sec   179 MBytes  1.49 Gbits/sec    6    357 KBytes
[  5]   1.01-2.00   sec   172 MBytes  1.46 Gbits/sec   19    202 KBytes
[  5]   2.00-3.00   sec   176 MBytes  1.48 Gbits/sec    8    295 KBytes
[  5]   3.00-4.01   sec   176 MBytes  1.47 Gbits/sec    2    280 KBytes
[  5]   4.01-5.00   sec   176 MBytes  1.49 Gbits/sec   11    264 KBytes
[  5]   5.00-6.00   sec   176 MBytes  1.47 Gbits/sec    4    373 KBytes
[  5]   6.00-7.00   sec   175 MBytes  1.47 Gbits/sec    6    295 KBytes
[  5]   7.00-8.01   sec   176 MBytes  1.47 Gbits/sec    4    280 KBytes
[  5]   8.01-9.00   sec   175 MBytes  1.47 Gbits/sec    4    295 KBytes
[  5]   9.00-10.01  sec   176 MBytes  1.47 Gbits/sec    4    280 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.01  sec  1.72 GBytes  1.47 Gbits/sec   68             sender
[  5]   0.00-10.02  sec  1.72 GBytes  1.47 Gbits/sec                  receiver
```




Thank You!