パブリッククラウドにおけるネットワーク品質の向上：パケット落ちの実態とその原因調査の実例

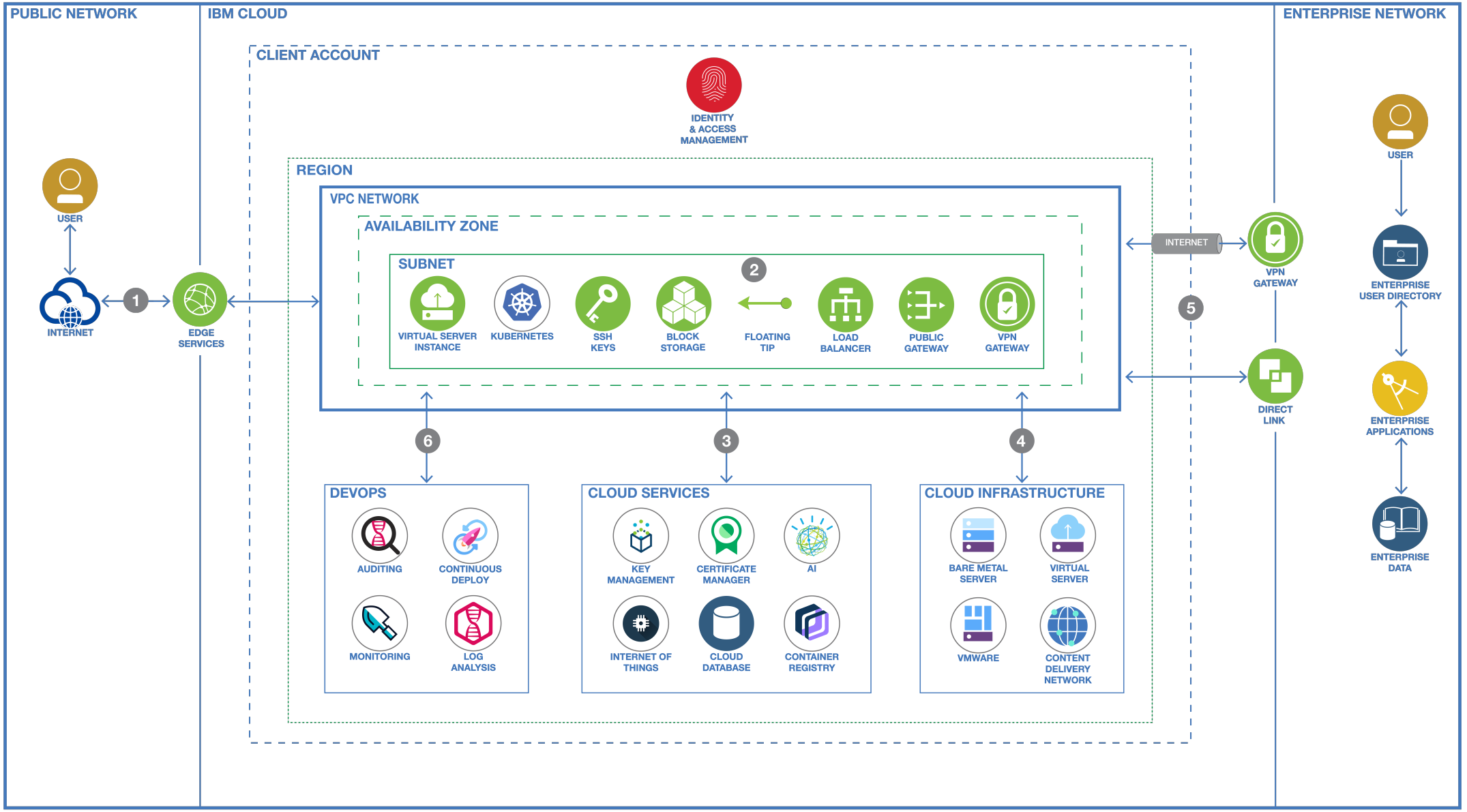
大平　怜
Infrastructure Performance
IBM Cloud

本講演の目的

- 商用パブリッククラウドで起きた性能上の問題を紹介
  - ネットワークのパケット落ちを題材として
- クラウドのネットワーク基盤の構成の一部を紹介
- パケット落ちの原因調査の方法論を紹介

- 何か皆さんの研究ネタにでもなれば……？

Virtual Private Cloud in IBM Cloud



ネットワーク品質とパケット落ち

- ネットワーク品質
  - バンド幅
  - レイテンシ
- パケット落ちはレイテンシ重視のワークロードへの影響大
  - バンド幅重視のワークロードではキュー（バッファ）のサイズが影響
    →メモリ使用量とのトレードオフで一定のパケット落ちは許容
  - キューのサイズ以外が原因のパケット落ちは原因調査に時間がかかる

仮想化ネットワークにおけるパケット落ち

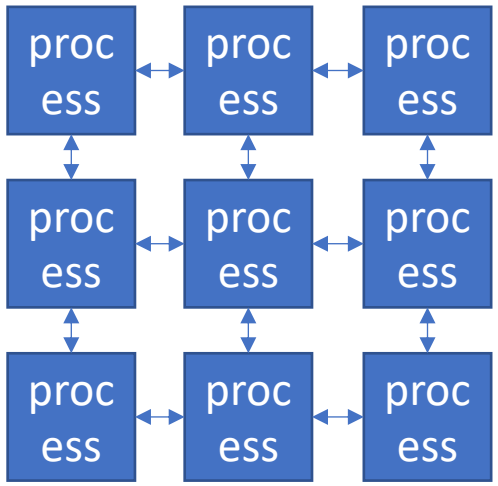
- クラウドの仮想化されたネットワークに特有のパケット落ちを紹介
  1. Linux Macvtapレイヤーにおけるパケット落ち
     - https://www.ibm.com/cloud/blog/diagnosing-packet-loss-in-linux-network-virtualization-layers-part-1
  2. Software Defined Overlay Networkが関連するパケット落ち
  - スイッチやネットワークカードでもパケットは落ちるが今回は除外

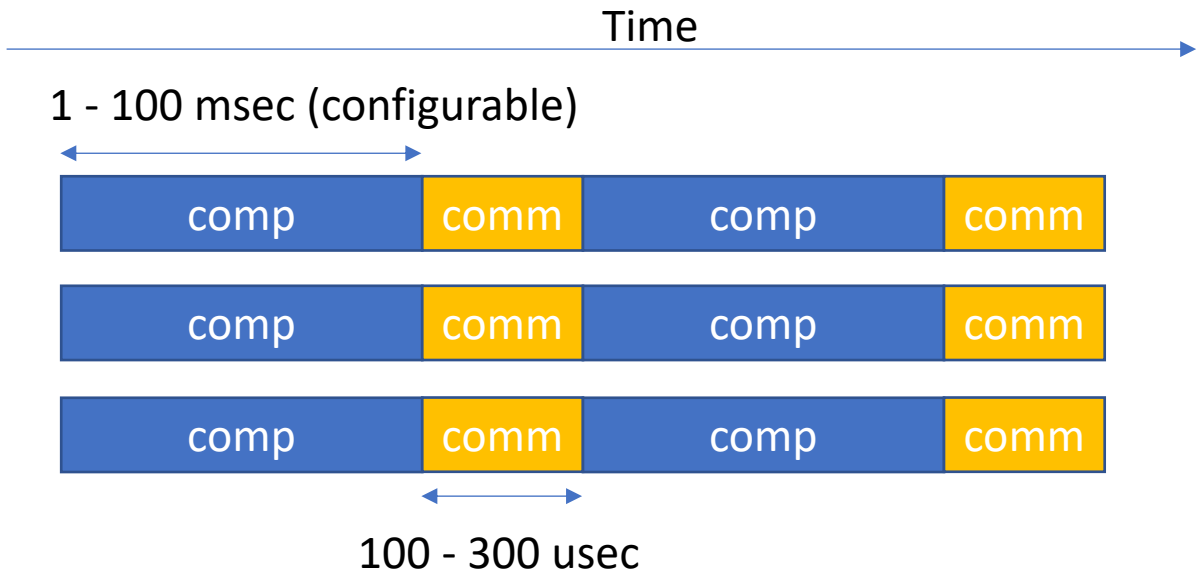
Diagnosing Macvtap Packet Loss

Rei Odaira, Saju Mathew, Kean Kuiper
Infrastructure Performance
IBM Cloud

Microbenchmark

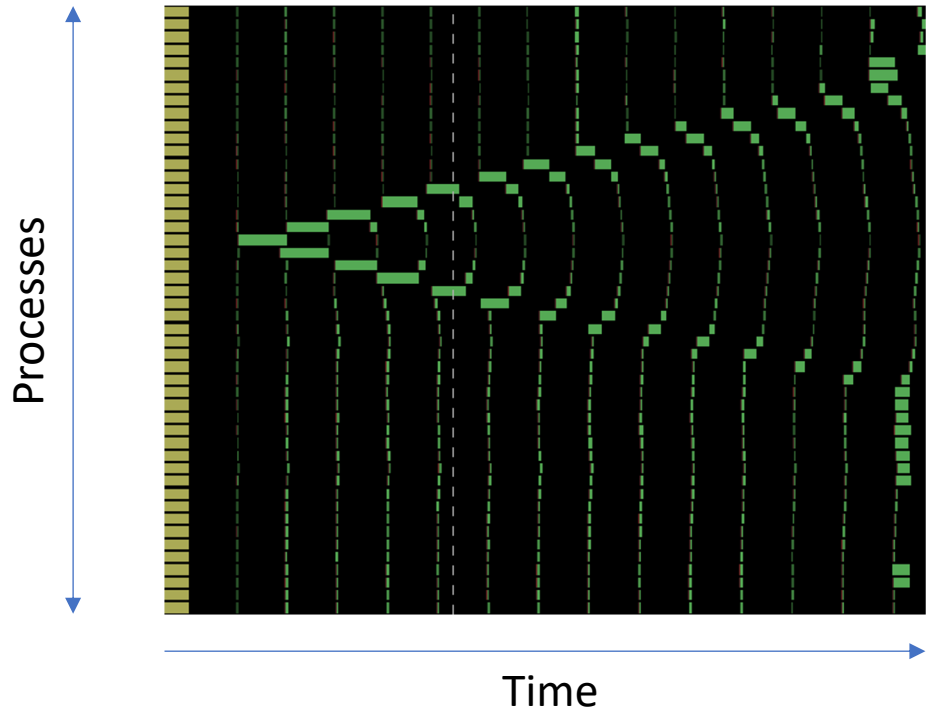
- MPI-based C program
  - 2D grid
- Alternates between computation and communication





Cascading Effect of Noise

- Delays propagate to neighbors
- It is essential to minimize noise in the communication phase



Black = computation
Green = communication

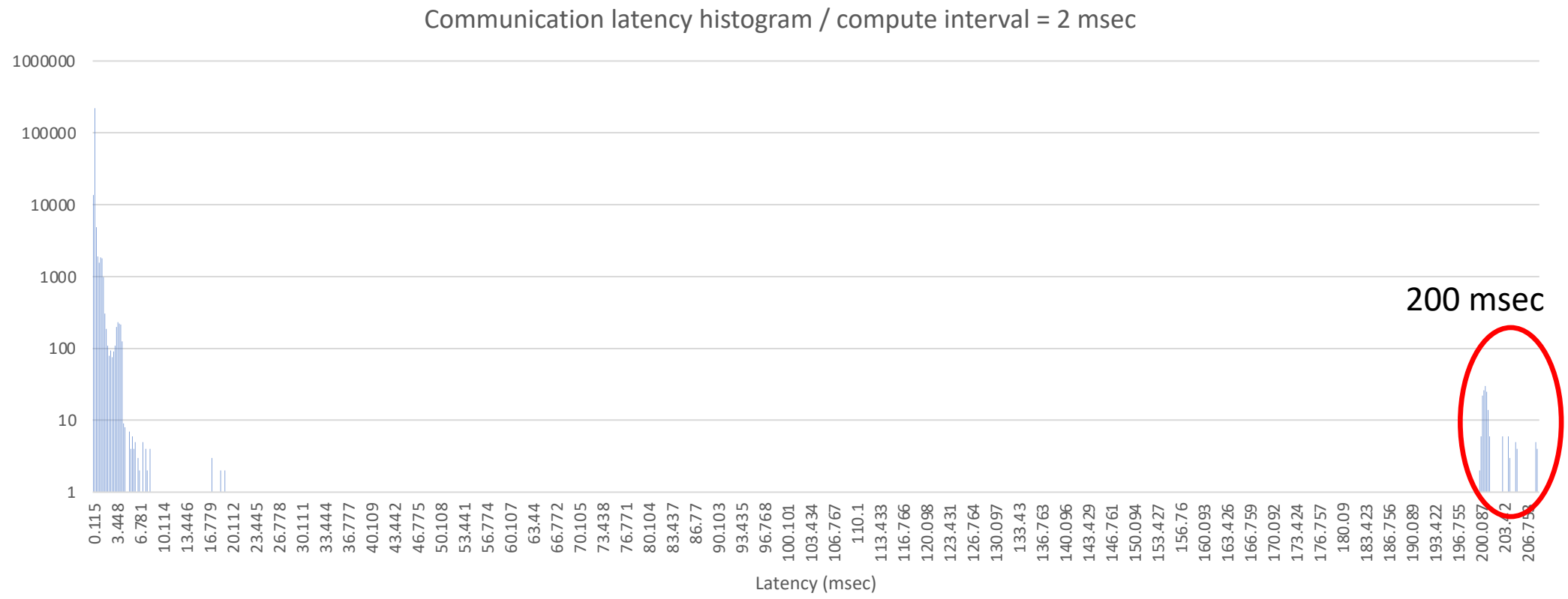
Experimental Environment

- Pre-prod, us-south-2
- 4x cx2-16x32
  - Scheduled on different hosts
- centos-7-amd64

- Open MPI
- 8 processes (ranks) per VSI, 32 processes in total
- Each process bound to a vCPU core by the `mpirun` command

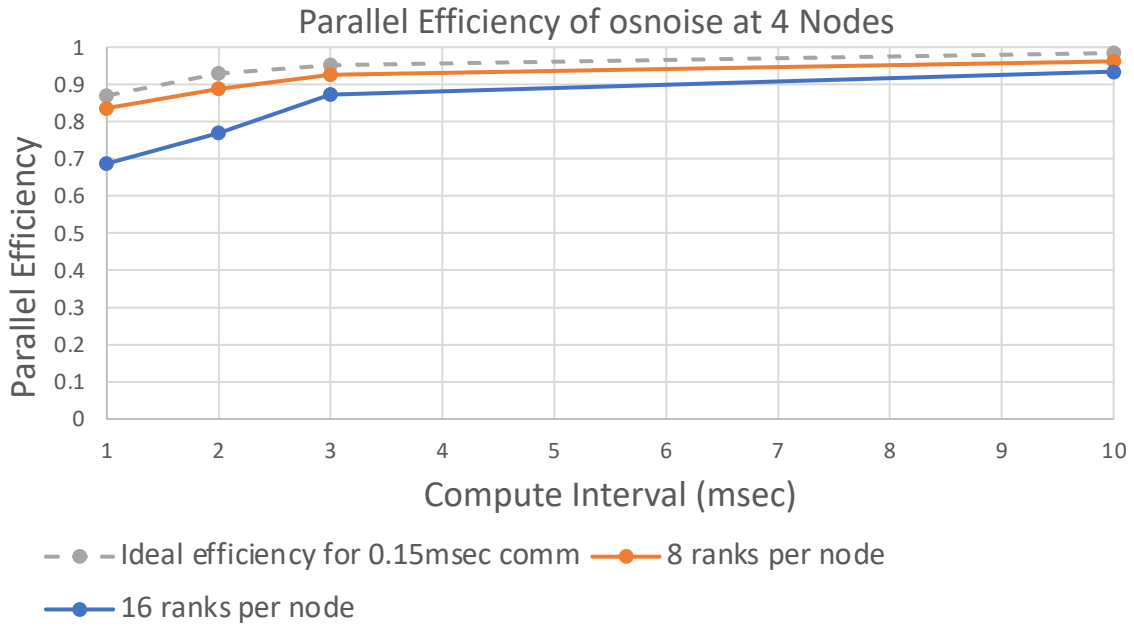
Communication Latency Histogram

- Compute interval 2 msec
- 500-sec run = 250,000 steps
- Max communication latency observed at `MPI_Waitall()` across the 32 processes, at each step



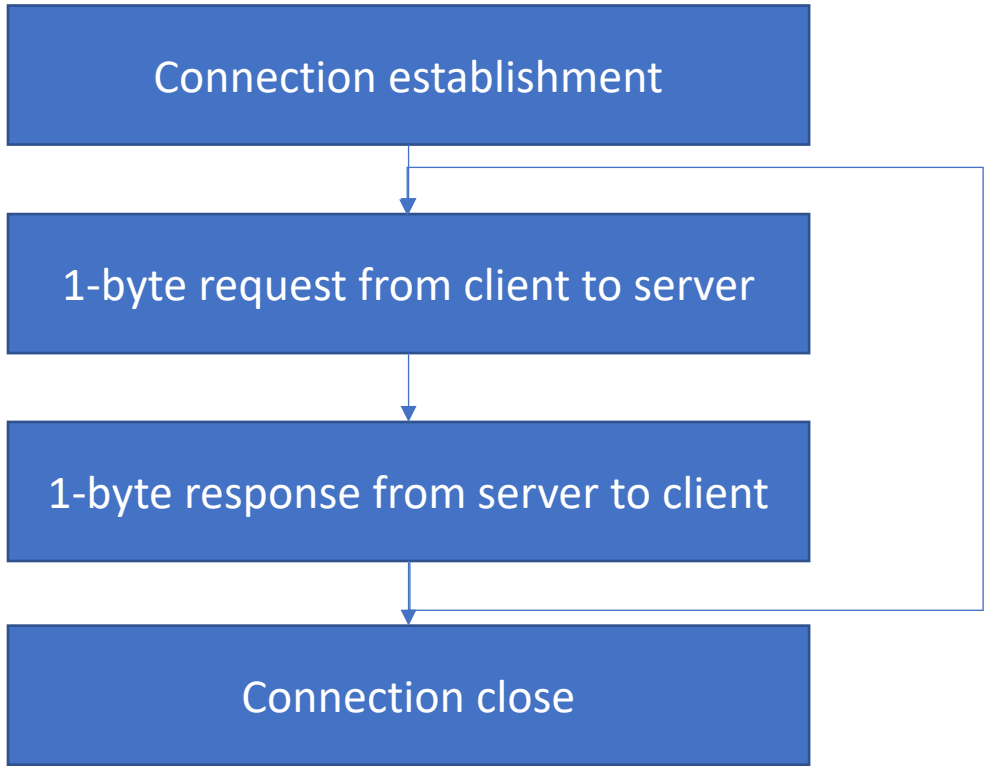
Parallel Efficiency

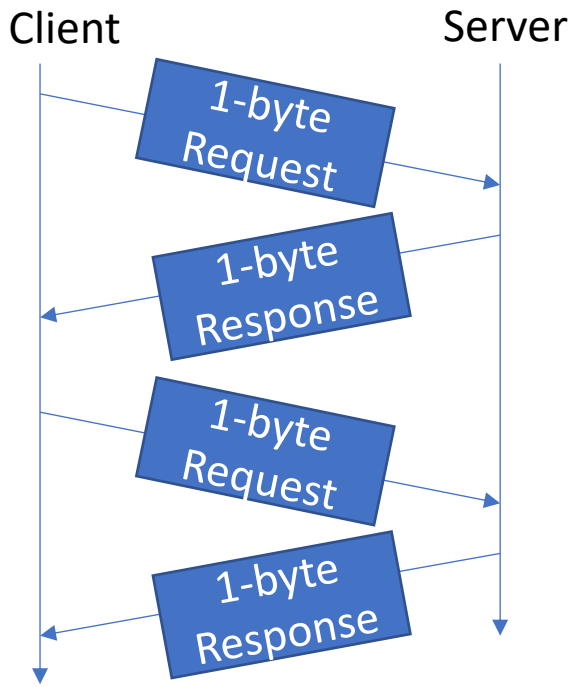
- Parallel efficiency = target time / measured time
  - = computation time / (computation time + communication time)
- Our measurement (see figure below): 92.6% at 3-msec compute interval with 4 nodes and 8 ranks per node
  - = 7.4% communication overhead
- Measurement by an internal customer: 75.5% at 3-msec compute interval with 32 nodes and 24 ranks per node



Netperf TCP_RR Benchmark

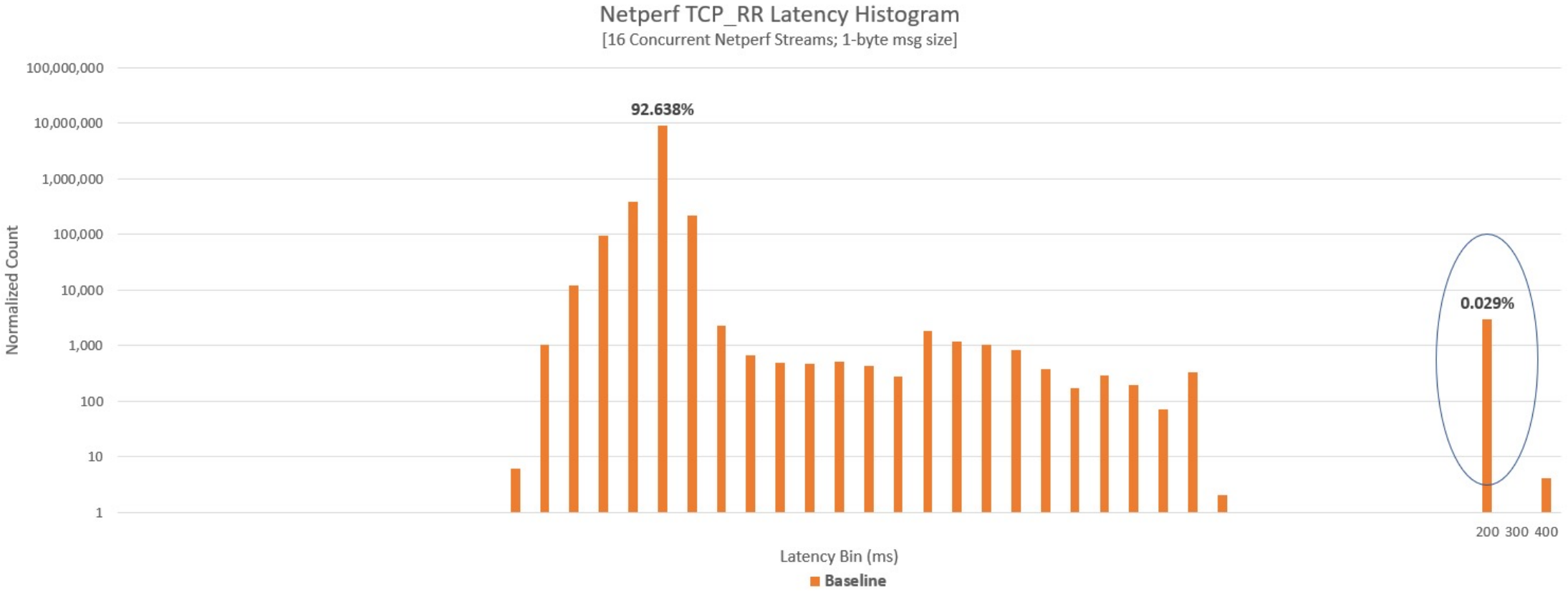
- TCP request-response





Netperf TCP_RR Latency Histogram

- Communication patter of osnoise corresponds to netperf TCP_RR
  - Osnoise retains established TCP connections throughout the execution
- Ran 8 netperf-netserver pairs
  - Used 2 of the 4 VSIs
  - Couldn’t reproduce the delays with 1 netperf-netserver pair

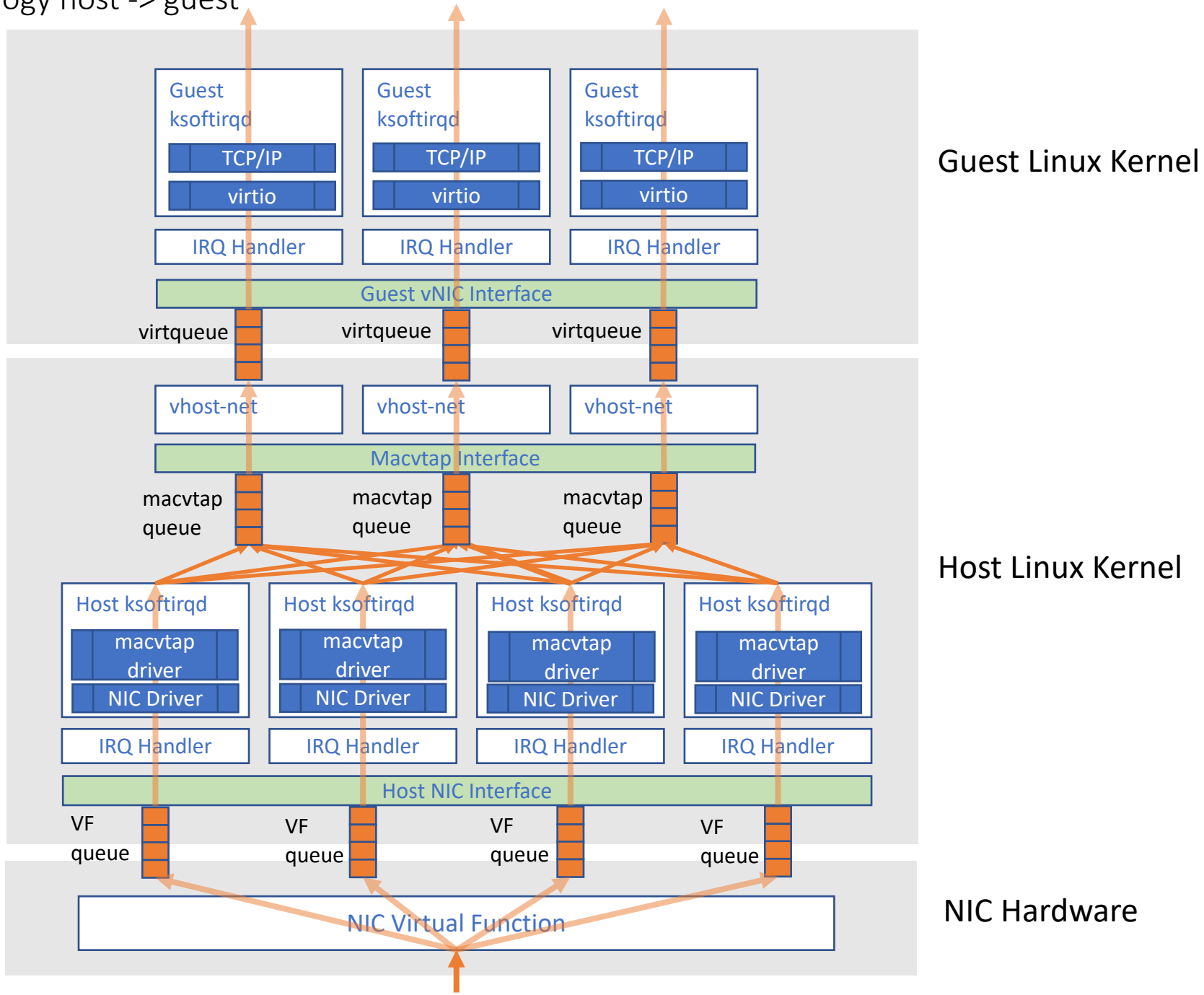


TCP Retransmission Analysis

- Number of 200-msec delays roughly matched TcpRetransSegs reported by nstat on the client guest VSI
- *TCP retransmits an unacknowledged packet up to tcp_retries2 sysctl setting times (defaults to 15) using an exponential backoff timeout for which each retransmission timeout is between TCP_RTO_MIN (200 ms) and TCP_RTO_MAX (120 seconds)*
  - *https://pracucci.com/linux-tcp-rto-min-max-and-tcp-retries2.html*

```
[root@reiperf-sysnoise-vsi1-0-gnnzq ~]# ip route
default via 10.240.64.1 dev eth0
10.240.64.0/24 dev eth0 proto kernel scope link src 10.240.64.5
169.254.0.0/16 dev eth0 scope link metric 1002
[root@reiperf-sysnoise-vsi1-0-gnnzq ~]# ip route change 10.240.64.0/24 dev eth0 rto_min 100ms proto kernel
scope link src 10.240.64.5
[root@reiperf-sysnoise-vsi1-0-gnnzq ~]# ip route
default via 10.240.64.1 dev eth0
10.240.64.0/24 dev eth0 proto kernel scope link src 10.240.64.5 rto_min lock 100ms
169.254.0.0/16 dev eth0 scope link metric 1002
```

Network emulation topology host -> guest

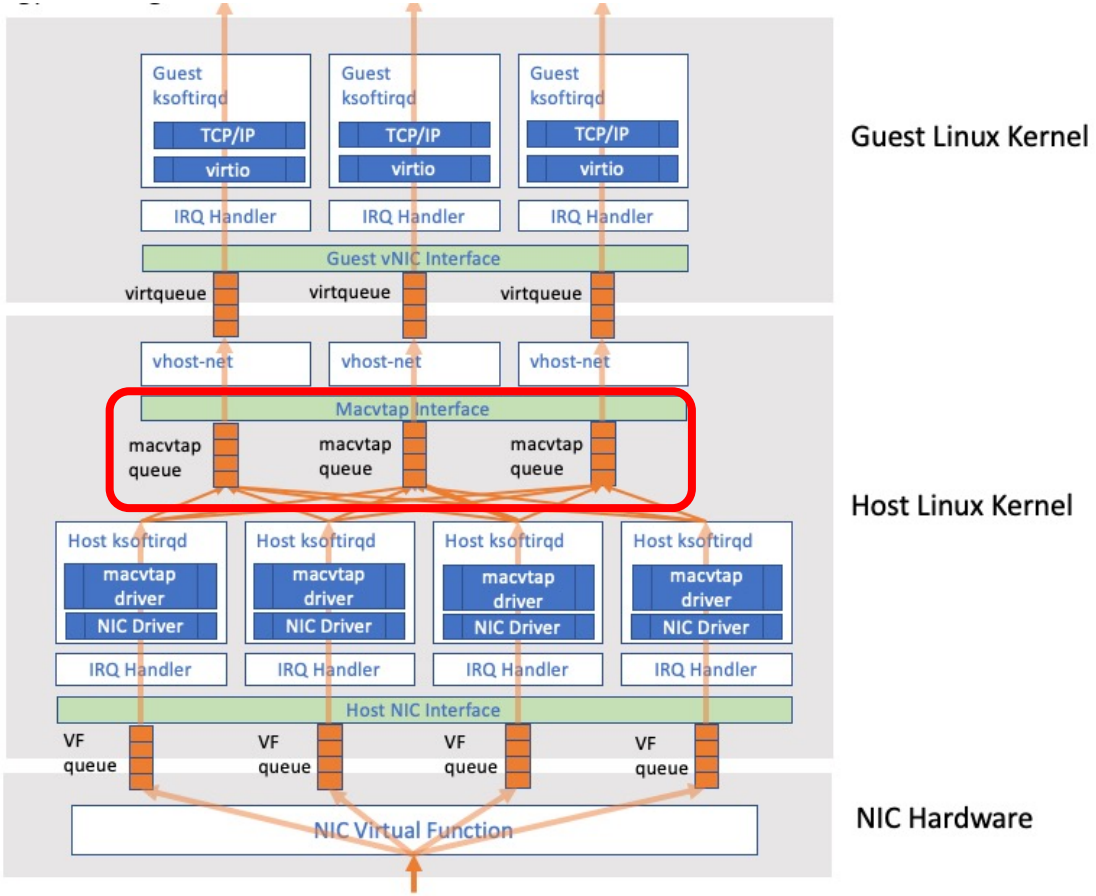


Packet Loss Analysis of 200 msec Delays

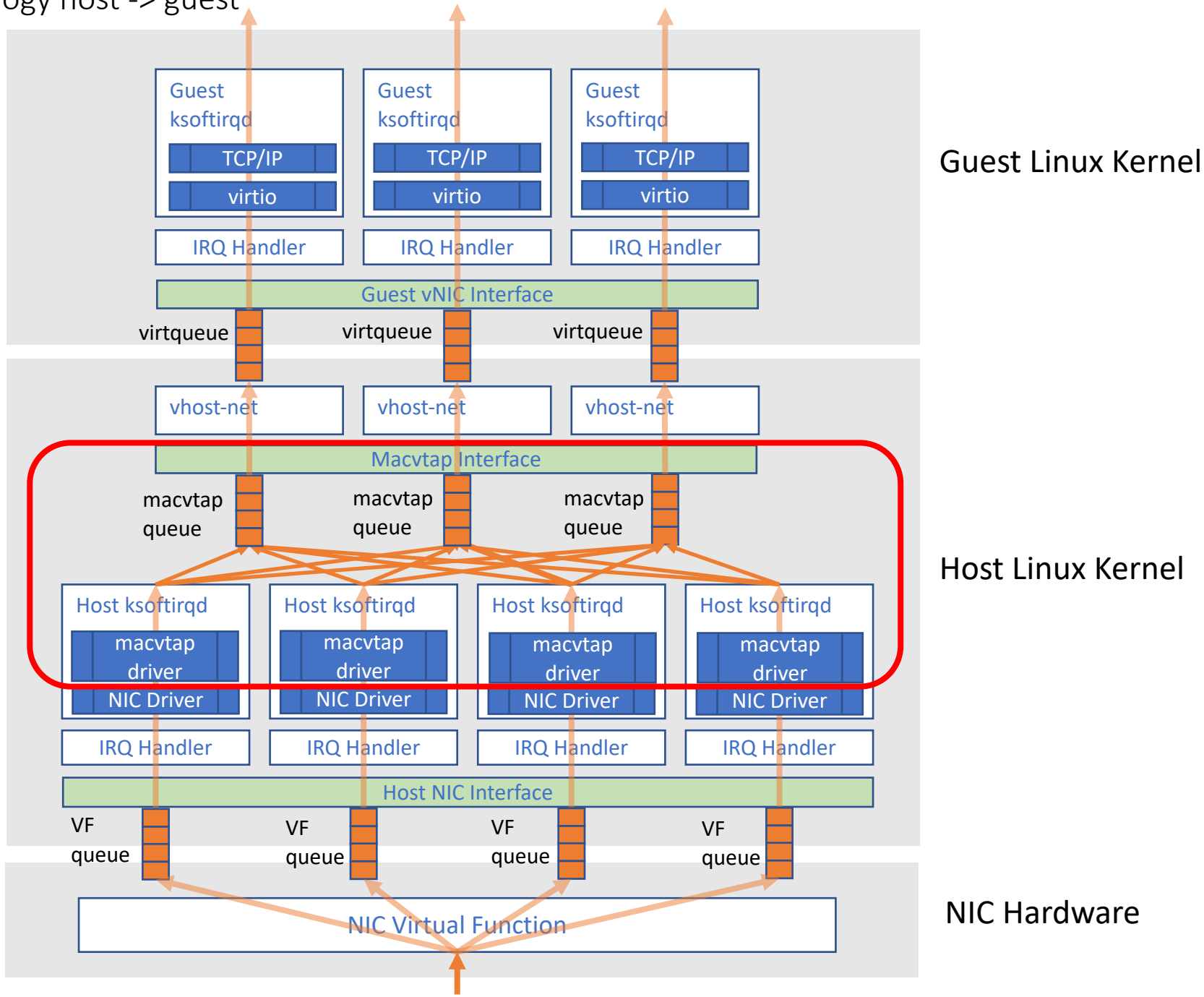
- We collected host statistics by running 8x netperf TCP_RR
- Number of 200-msec delays exactly matched RX dropped packets at macvtap

```
ifconfig macvtap2  (delta)
macvtap3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 0
        inet6 fe80::0ff:fe00:0c  prefixlen 0  scopeid 0x0<link>
        ether 0:0:0:0:0:0c  txqueuelen 0  (Ethernet)
        RX packets 4140032  bytes 248426110 (0.3 GB)
        RX errors 152  dropped 152  overruns 0  frame 0
        TX packets 4140168  bytes 227804160 (0.2 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ifconfig macvtap2  (delta)
macvtap2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 0
        inet6 fe80::0ff:fe00:0c  prefixlen 0  scopeid 0x0<link>
        ether 0:0:0:0:0:0c  txqueuelen 0  (Ethernet)
        RX packets 4139997  bytes 248429813 (0.2 GB)
        RX errors 160  dropped 160  overruns 0  frame 0
        TX packets 4139941  bytes 227729333 (0.2 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```



Network emulation topology host -> guest



Source Code Analysis (drivers/net/tap.c in Linux 4.15.0)

```
 1: rx_handler_result_t tap_handle_frame(struct sk_buff **pskb)
 2: {
    ...
 3:         q = tap_get_queue(tap, skb);
    ...
 4:         if (__skb_array_full(&q->skb_array))
 5:                 goto drop;
    ...
 6:         if (netif_needs_gso(skb, features)) {
 7:                 struct sk_buff *segs = __skb_gso_segment(skb, features, false);
 8:                 if (IS_ERR(segs))
 9:                         goto drop;
    ...
10:         } else {
    ...
11:                 if (... && ... && skb_checksum_help(skb))
12:                         goto drop;
13:                 if (skb_array_produce(&q->skb_array, skb))
14:                         goto drop;
15:         }
    ...
16: drop:
17:         if (tap->count_rx_dropped)
18:                 tap->count_rx_dropped(tap);
    ...
19: }
```

Source Code Analysis (drivers/net/tap.c in Linux 4.15.0)

```
 1: rx_handler_result_t tap_handle_frame(struct sk_buff **pskb)
 2: {
    ...
 3:         q = tap_get_queue(tap, skb);
    ...
 4:         if (__skb_array_full(&q->skb_array))
 5:                 goto drop;
    ...
 6:         if (netif_needs_gso(skb, features)) {
 7:                 struct sk_buff *segs = __skb_gso_segment(skb, features, false);
 8:                 if (IS_ERR(segs))
 9:                         goto drop;
    ...
10:         } else {
    ...
11:                 if (... && ... && skb_checksum_help(skb))
12:                         goto drop;
13:                 if (skb_array_produce(&q->skb_array, skb))
14:                         goto drop;
15:         }
    ...
16: drop:
17:         if (tap->count_rx_dropped)
18:                 tap->count_rx_dropped(tap);
    ...
19: }
```

Kernel Instrumentation with SystemTap

- Simple CLI and scripting language to inject a probe into the running Linux kernel

SystemTap script

```
probe module("macvtap").function("macvtap_count_rx_dropped").call
{
    printf("dropped\n")
}

probe kernel.function("skb_checksum_help").call
{
    printf("checksum\n")
}

probe kernel.function("__skb_gso_segment").call
{
    printf("gso\n")
}
```

Execution example

```
# stap -v macvtap.stp
dropped
dropped
dropped
dropped
dropped
dropped
dropped
dropped
dropped
dropped
...
```

No packet drop on the checksum or gso paths

Source Code Analysis (drivers/net/tap.c in Linux 4.15.0)

```
 1: rx_handler_result_t tap_handle_frame(struct sk_buff **pskb)
 2: {
    ...
 3:         q = tap_get_queue(tap, skb);
    ...
 4:         if (__skb_array_full(&q->skb_array))
 5:                 goto drop;
    ...
 6:         if (netif_needs_gso(skb, features)) {
 7:                 struct sk_buff *segs = __skb_gso_segment(skb, features, false);
 8:                 if (IS_ERR(segs))
 9:                         goto drop;
    ...
10:         } else {
    ...
11:                 if (... && ... && skb_checksum_help(skb))
12:                         goto drop;
13:                 if (skb_array_produce(&q->skb_array, skb))
14:                         goto drop;
15:         }
    ...
16: drop:
17:         if (tap->count_rx_dropped)
18:                 tap->count_rx_dropped(tap);
    ...
19: }
```

Source Code Analysis (drivers/net/tap.c in Linux 4.15.0)

```
rx_handler_result_t tap_handle_frame(struct sk_buff **pskb)
{
    ...
        q = tap_get_queue(tap, skb);
    ...
        if (__skb_array_full(&q->skb_array))
                goto drop;
    ...
        if (skb_array_produce(&q->skb_array, skb))
                goto drop;
    ...
}
```

Instrumenting Jump Instruction with SystemTap

- Identify which jump instruction to instrument in assembly code
- Obtain the absolute memory address of the jump instruction
  - /proc/kallsyms
- Determine what condition to check at the jump instruction

```
rx_handler_result_t tap_handle_frame(struct sk_buff **pskb)
{
    ...
        if (__skb_array_full(&q->skb_array))
                goto drop;
```

```
2349:       movslq 0x380(%r13),%rdx
2350:       mov    0x408(%r13),%rax
2357:       cmpq   $0x0,(%rax,%rdx,8)
235c:       je     23bb <tap_handle_frame+0xfb>
```

Memory address of je

```
probe kprobe.statement(0xFFFFFFFFC1F1B35C).absolute
{
    if (! (register("flags") & 0x40)) {
        printf("full\n")
    }
}
```

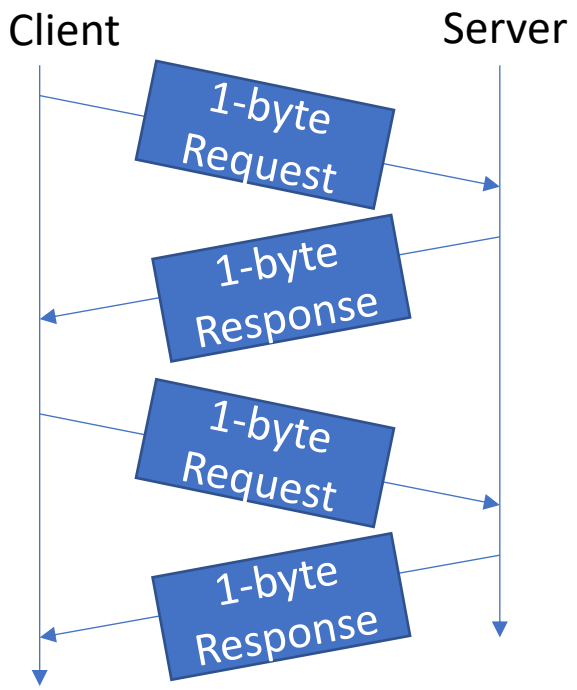
Bit position of the Zero flag in the flag register

Macvtap Queue is (Considered) Full

- Confirmed that the number of packet drops matched the number of times the goto in red was taken

```
rx_handler_result_t tap_handle_frame(struct sk_buff **pskb)
{
    ...
        q = tap_get_queue(tap, skb);
    ...
        if (__skb_array_full(&q->skb_array))
                goto drop;
    ...
        if (skb_array_produce(&q->skb_array, skb)
                goto drop;
    ...
}
```

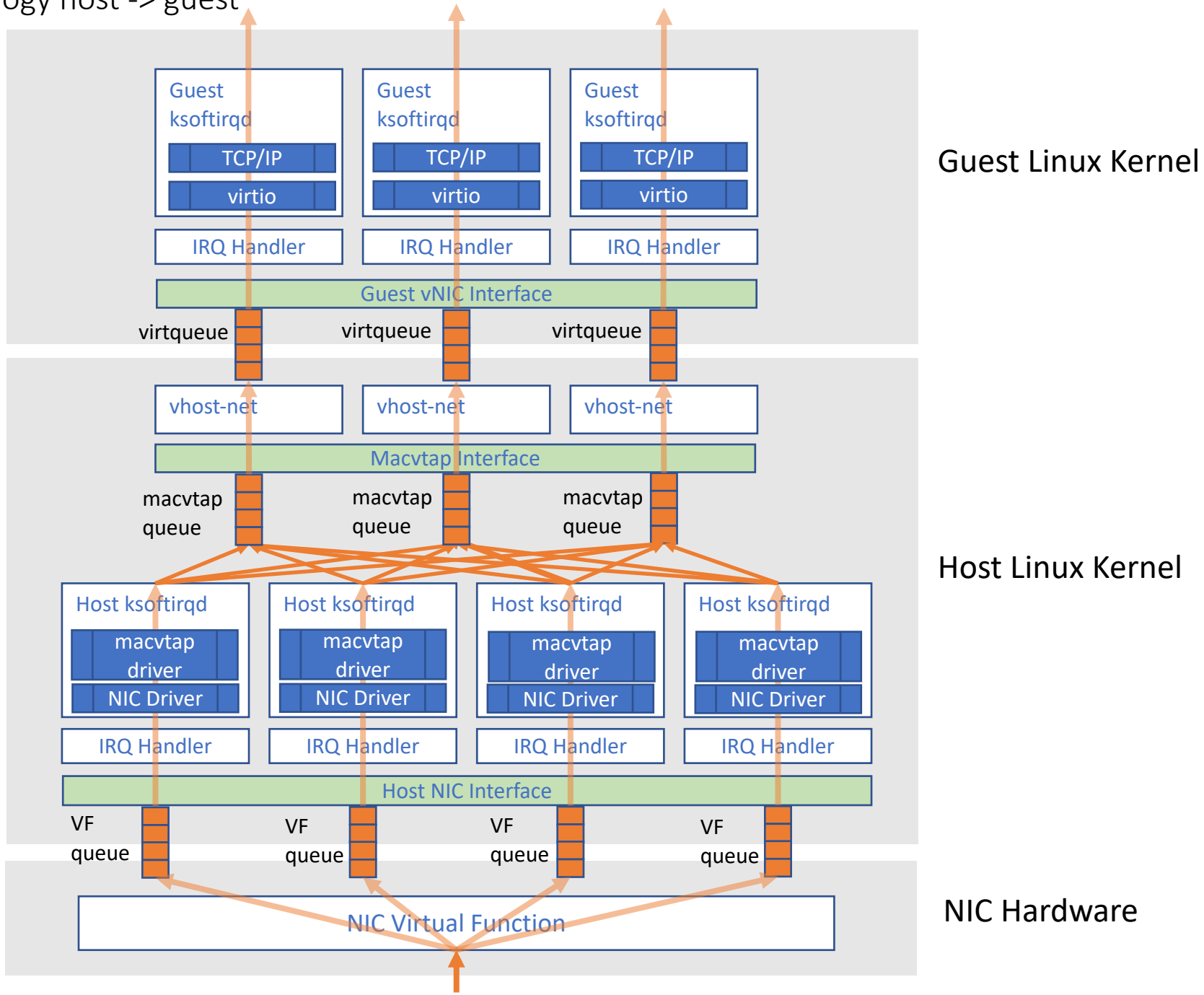
Is the Macvtap Queue Really Full?



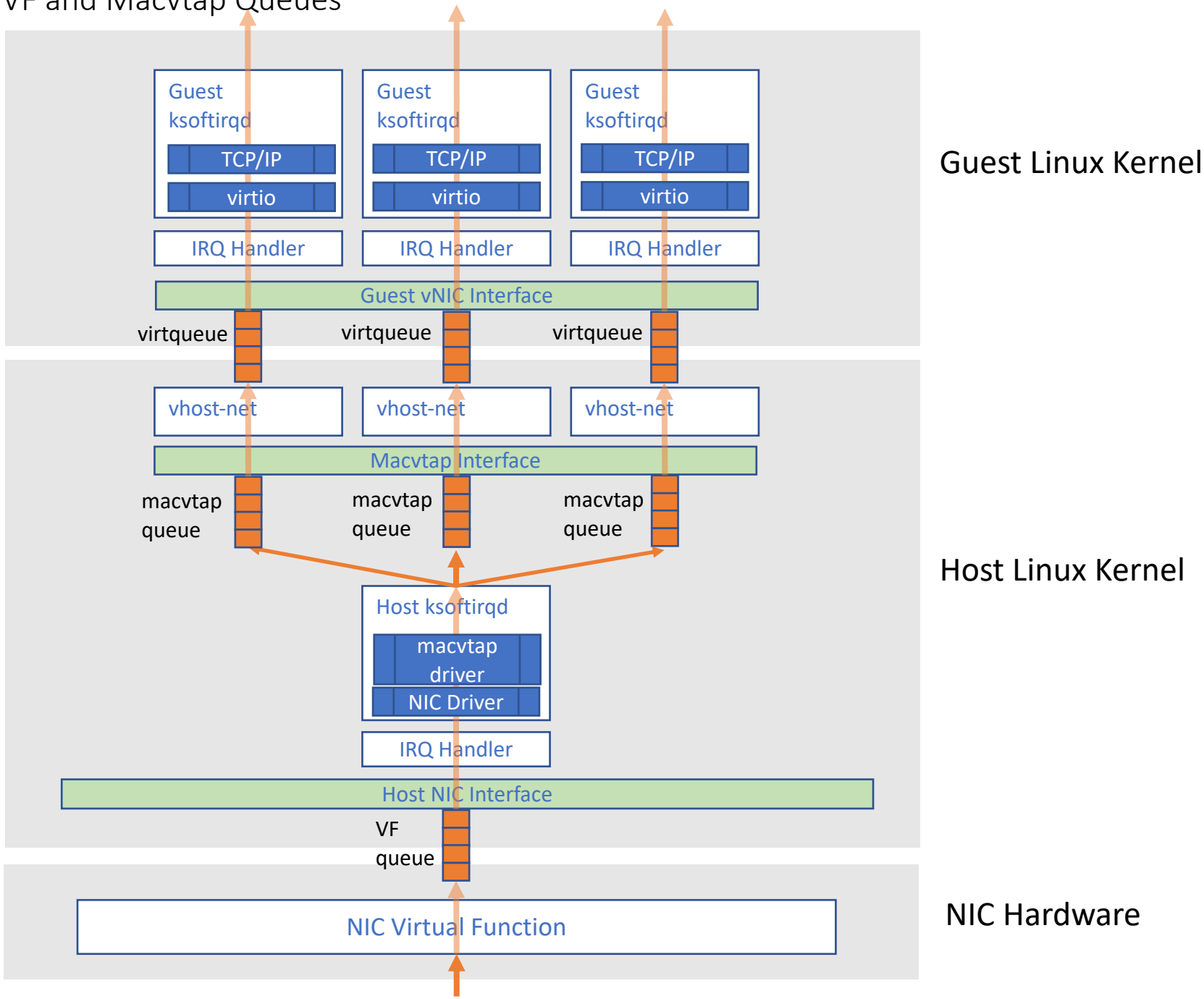
- It is very unlikely
- Netperf TCP_RR is a synchronous benchmark
  - Eight parallel netperf processes can have at most eight packets on the network
- According to SystemTap, none of the queues had more than one packet when a packet was dropped

```
probe module("macvtap").function("macvtap_count_rx_dropped").call
{
  for (i = 0; i < $tap->numqueues; i++) {
    printf("q%d %d %d %p\n", i,
           $tap->taps[i]->skb_array->ring->producer,
           $tap->taps[i]->skb_array->ring->consumer_head,
           $tap->taps[i]->skb_array->ring->queue[$tap->taps[i]->skb_array->ring->producer])
    }
}
```

Network emulation topology host -> guest

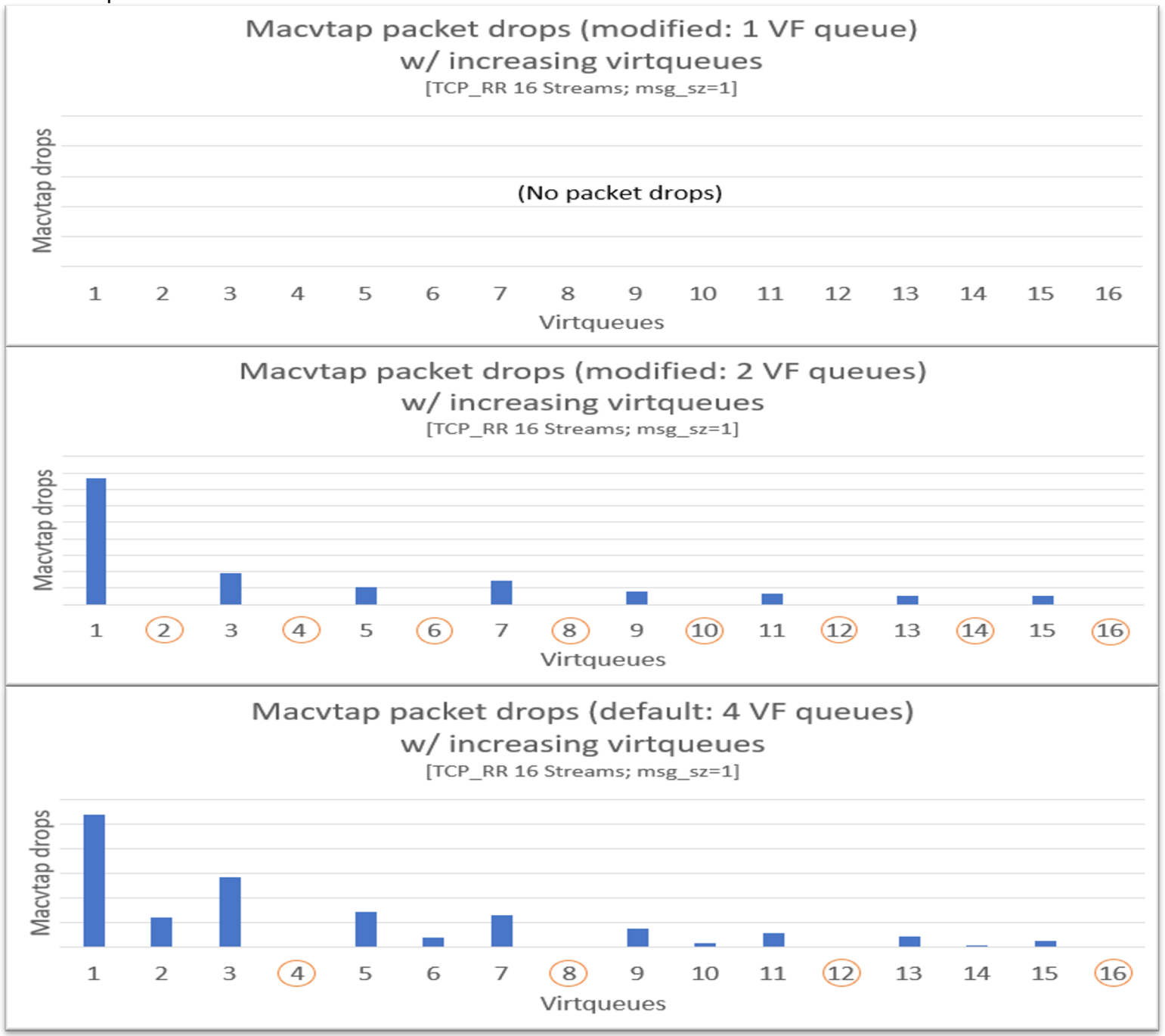


Changing the Number of VF and Macvtap Queues

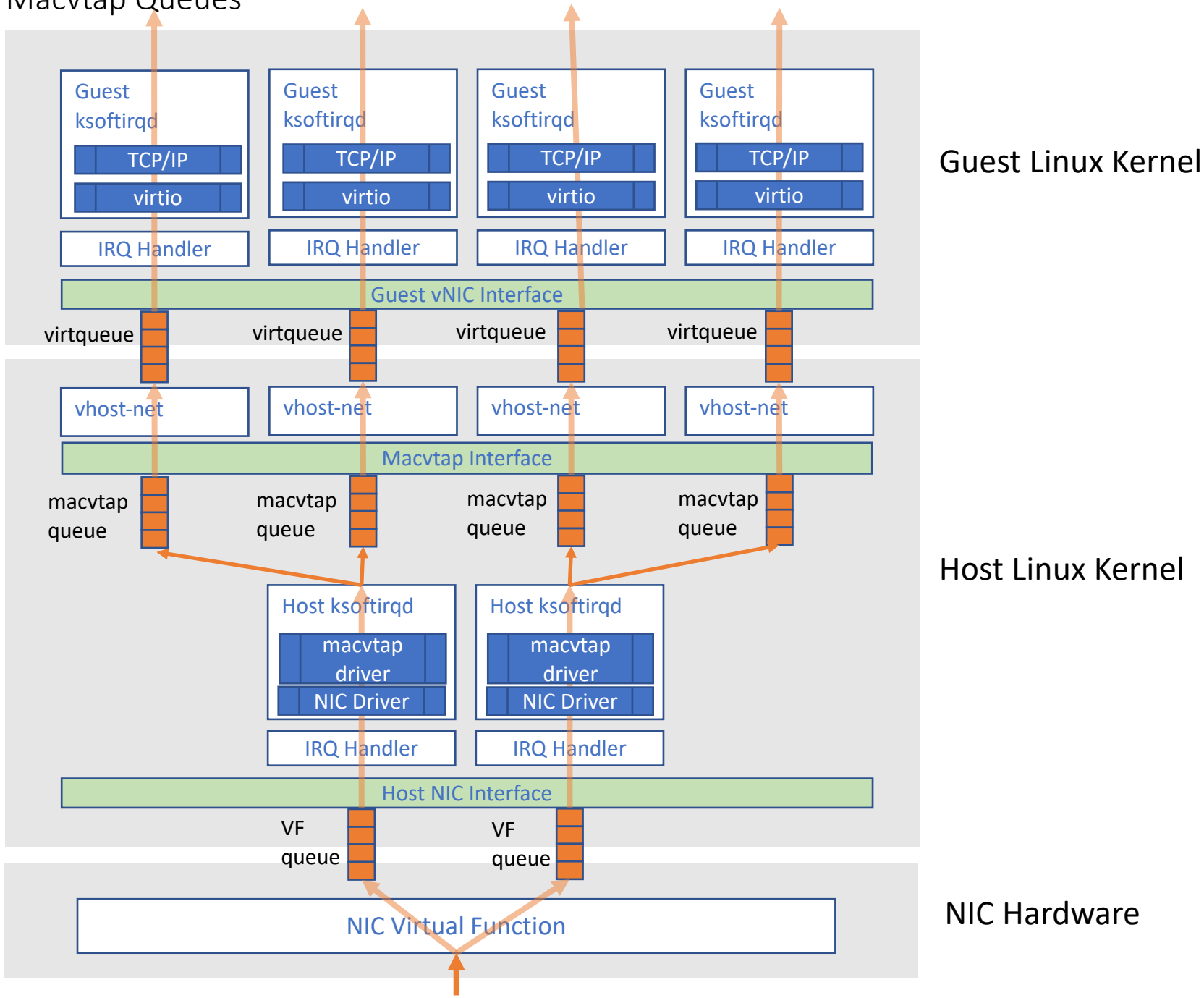


Loss rate as a function of #VF and #virtqueue

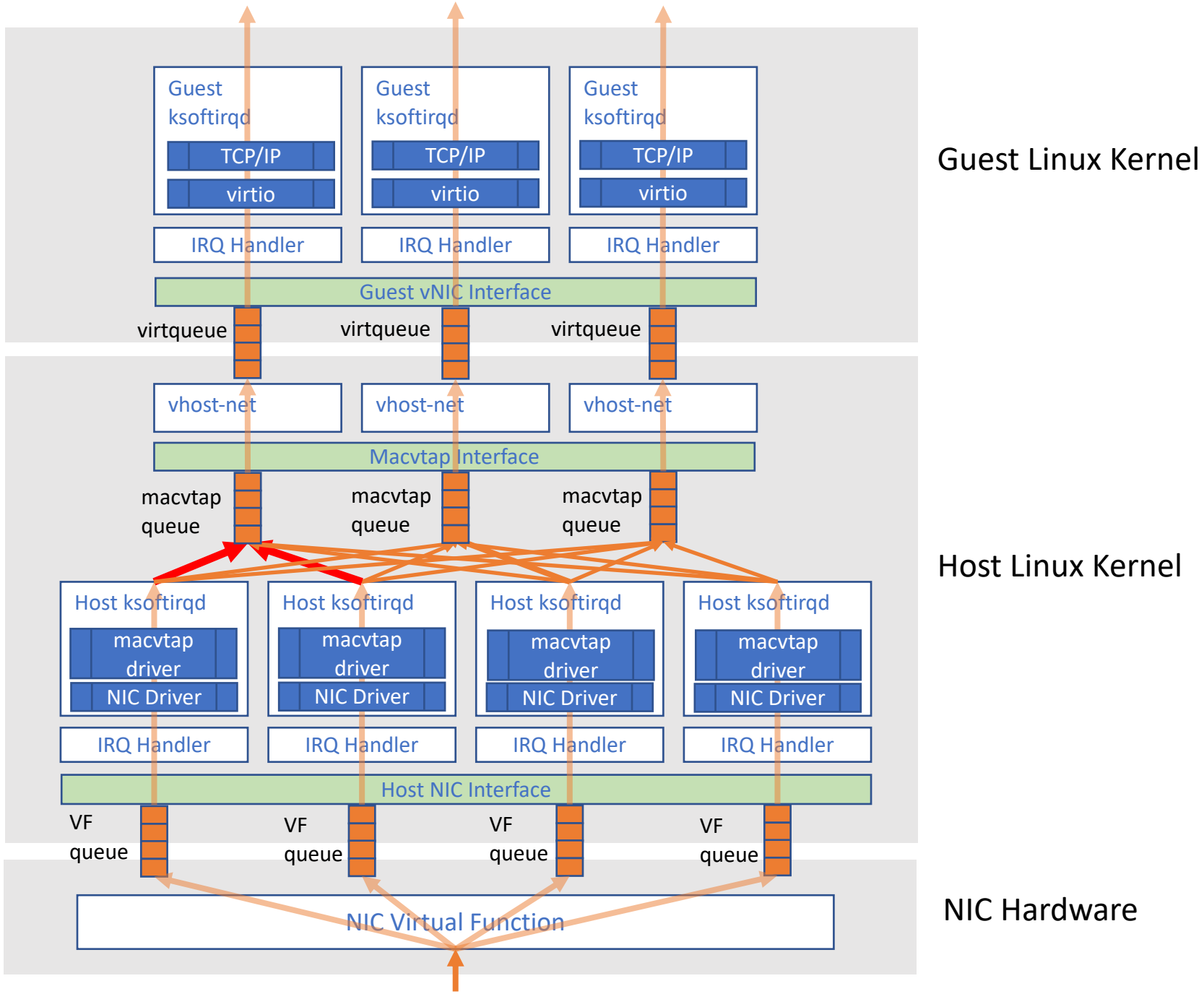
When the #virtqueues is a multiple of the #VF queues there are no drops



Two VF Queues and Four Macvtap Queues



Concurrency Bug…?



Source Code Analysis

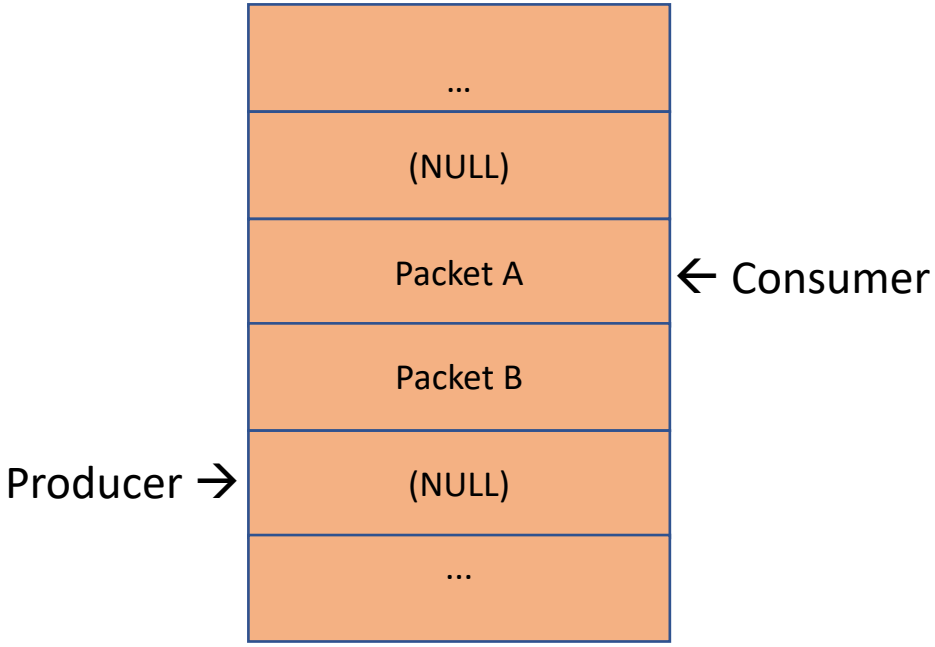
```
rx_handler_result_t tap_handle_frame(struct sk_buff **pskb)
{
    ...
        if (__skb_array_full(&q->skb_array))
                goto drop;
    ...
        if (skb_array_produce(&q->skb_array, skb))
                goto drop;
    ...
}
```

```
static inline int skb_array_produce(struct skb_array *a, struct sk_buff *skb)
{
        return ptr_ring_produce(&a->ring, skb);
}
```

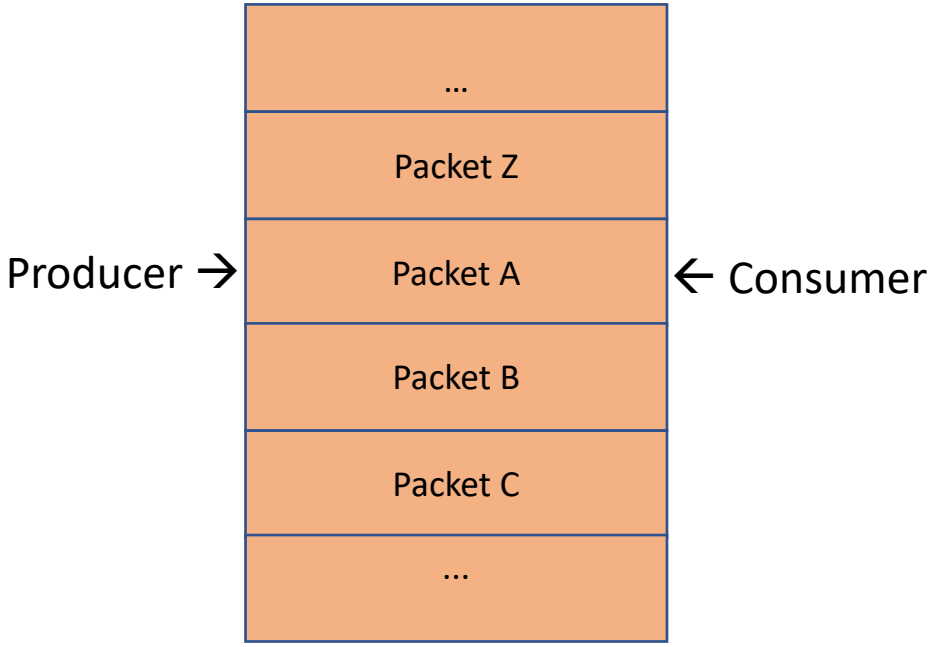
```
static inline int ptr_ring_produce(struct ptr_ring *r, void *ptr)
{
        int ret;
        spin_lock(&r->producer_lock);
        ret = __ptr_ring_produce(r, ptr);
        spin_unlock(&r->producer_lock);
        return ret;
}
```

Macvtap Queue Structure

- Logically structured as a ring
- Entry is either NULL or points to a packet



Macvtap queue with two packets



A full Macvtap queue

Source Code Analysis: `__ptr_ring_produce()`

```
 static inline int __ptr_ring_produce(struct ptr_ring *r, void *ptr)
{
        if (unlikely(!r->size) || r->queue[r->producer])
                return -ENOSPC;
        smp_wmb();

        r->queue[r->producer++] = ptr;
        if (unlikely(r->producer >= r->size))
                r->producer = 0;
        return 0;
}
```

Source Code Analysis: Early Check

```
rx_handler_result_t tap_handle_frame(struct sk_buff **pskb)
{
    ...
        if (__skb_array_full(&q->skb_array))
                goto drop;
    ...
        if (skb_array_produce(&q->skb_array, skb))
                goto drop;
    ...
}
```

```
static inline bool __skb_array_full(struct skb_array *a)
{
        return __ptr_ring_full(&a->ring);
}
```

```
static inline bool __ptr_ring_full(struct ptr_ring *r)
{
        return r->queue[r->producer];
}
```

Concurrency Bug

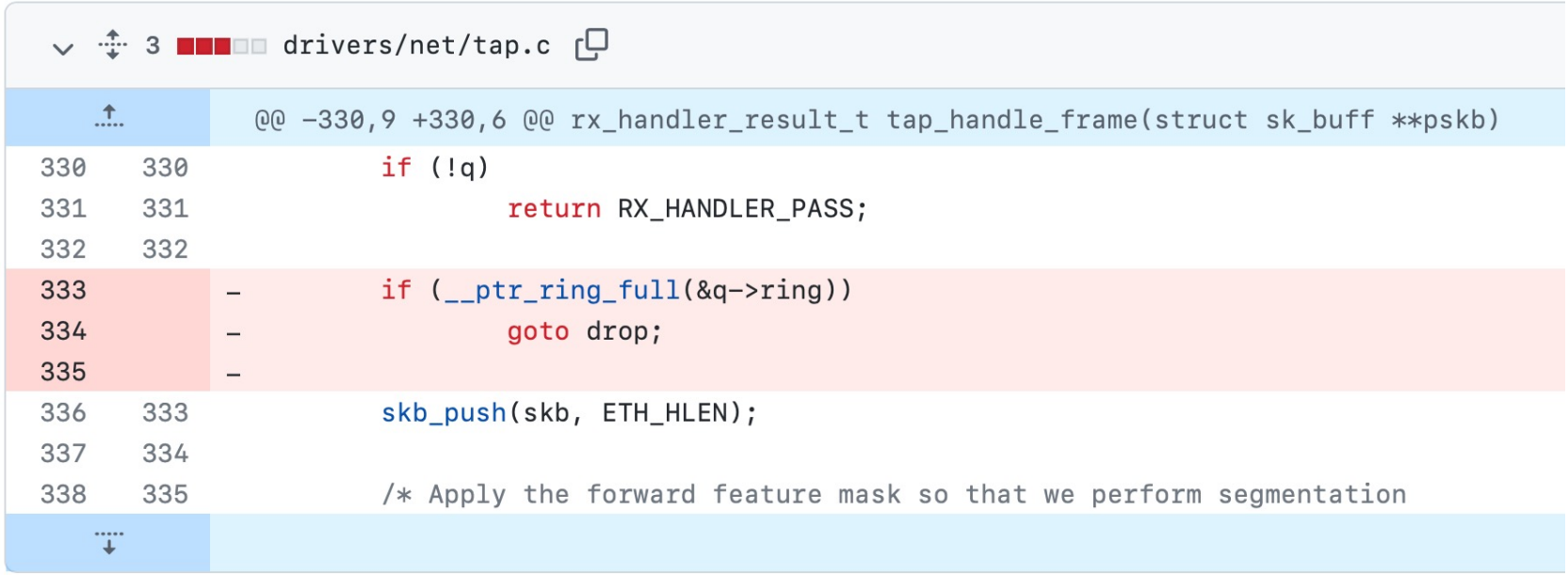
```
r->queue[r->producer++] = ptr;
```


```
r->queue[r->producer]
```

```
                                    read r->producer              ; old value
write r->producer = producer + 1                                  ; new value
write r->queue[producer]
                                    read r->queue[producer]       ; **seems full**
```

Upstream Kernel Already Removed the Early Check For Different Reason

- https://github.com/torvalds/linux/commit/88fae87327a2261cf8db078f6ce4e5a3e55b30b1#diff-14c2f5b9b9139ab07400b51cd5f691e5
  - *Lockless access to __ptr_ring_full is only legal if ring is never resized, otherwise it might cause use-after free errors. Simply drop the lockless test, we'll drop the packet a bit later when produce fails.*

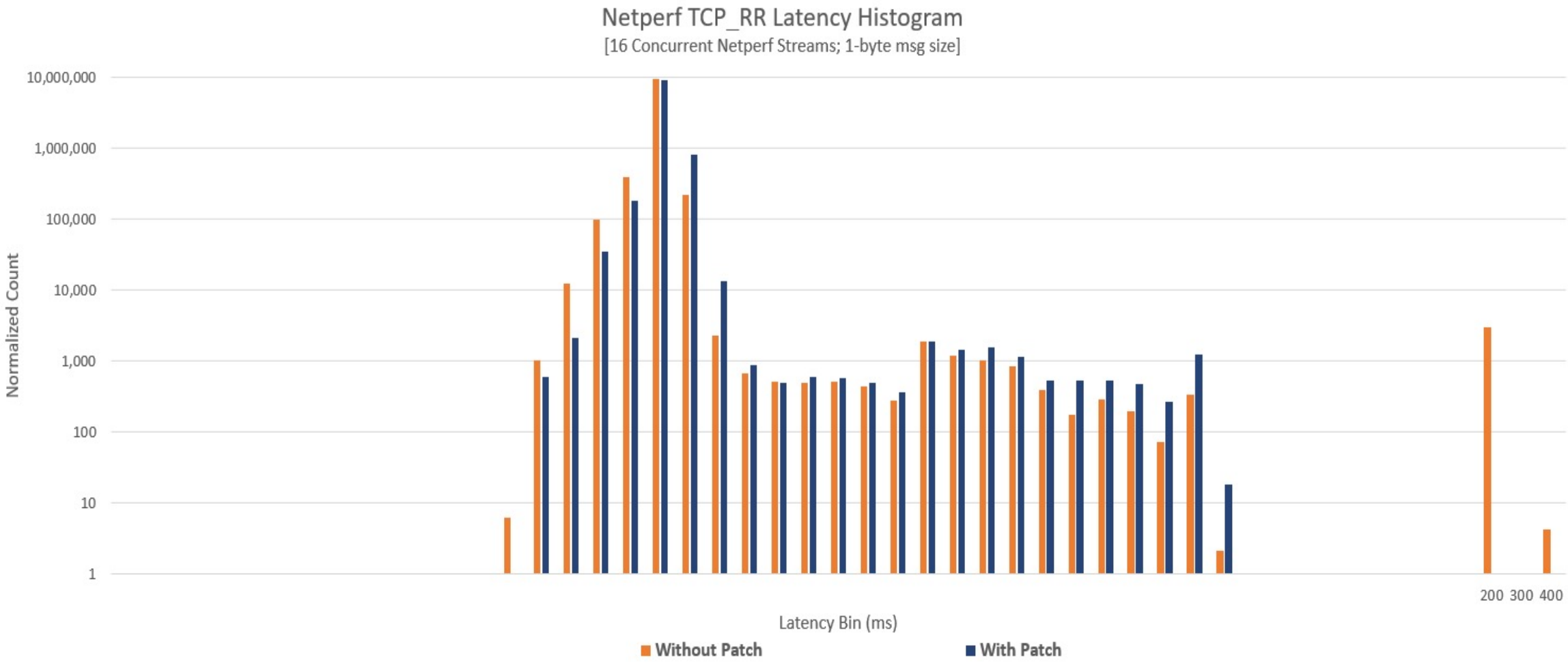


Updating Prod Environments

- Confirmed that packets were not dropped after applying the patch
  - Kernel Live Patching
- Created a new host Linux kernel image
- Performed a rolling update of the host kernel on all nodes
  - Live migration

- All production data centers have been updated

Tail Latency Significantly Improved in Netperf TCP_RR

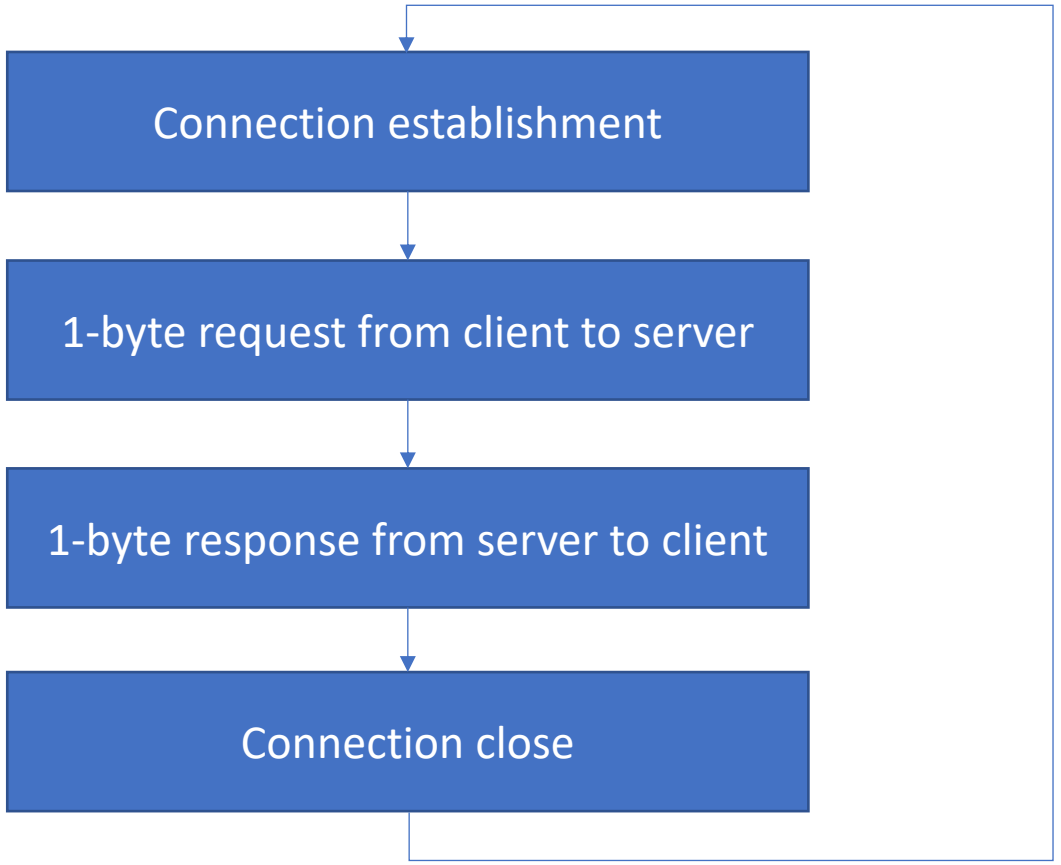


Fixing Connection Errors

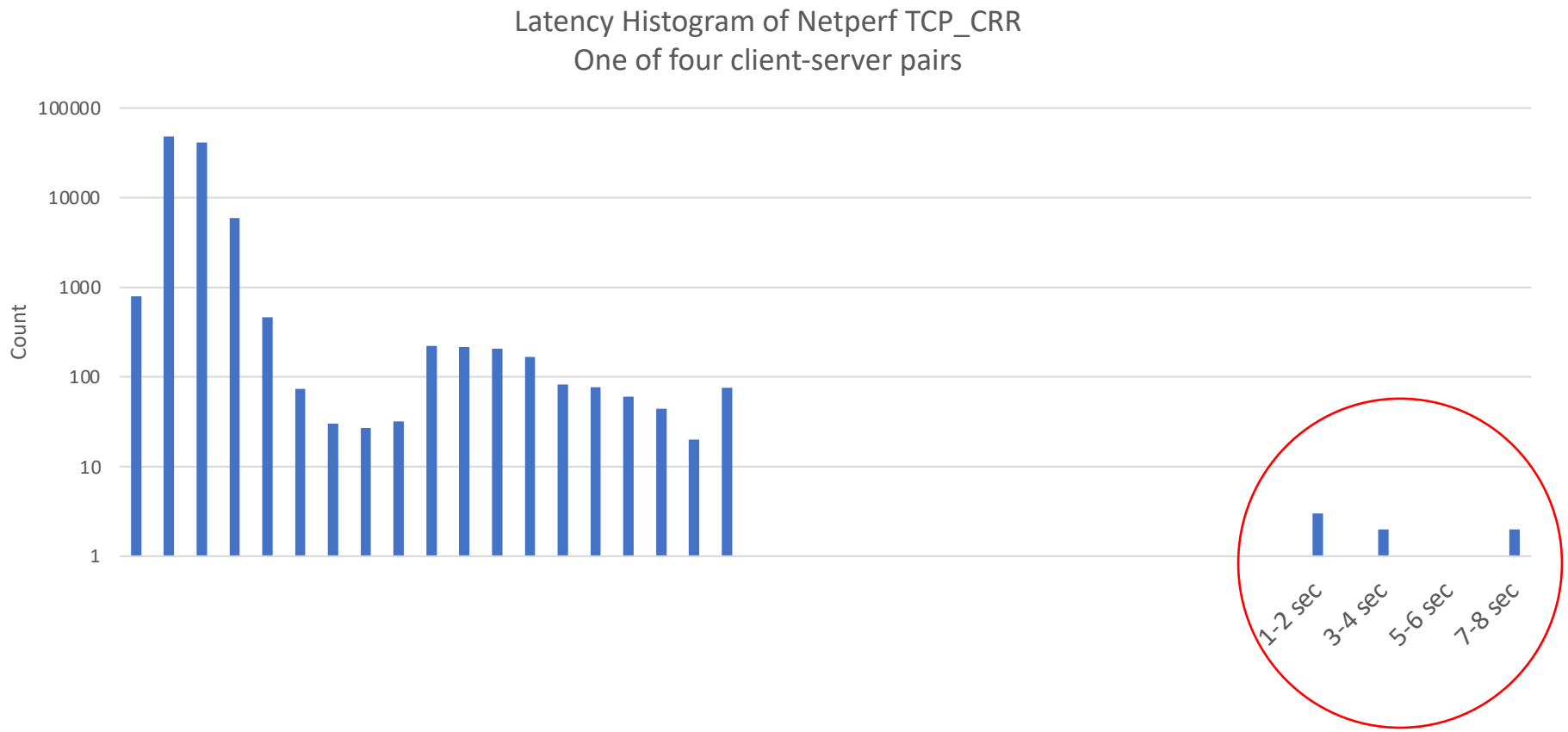
Rei Odaira
Infrastructure Performance
IBM Cloud

Netperf TCP_CRR Benchmark

- TCP connection-request-response

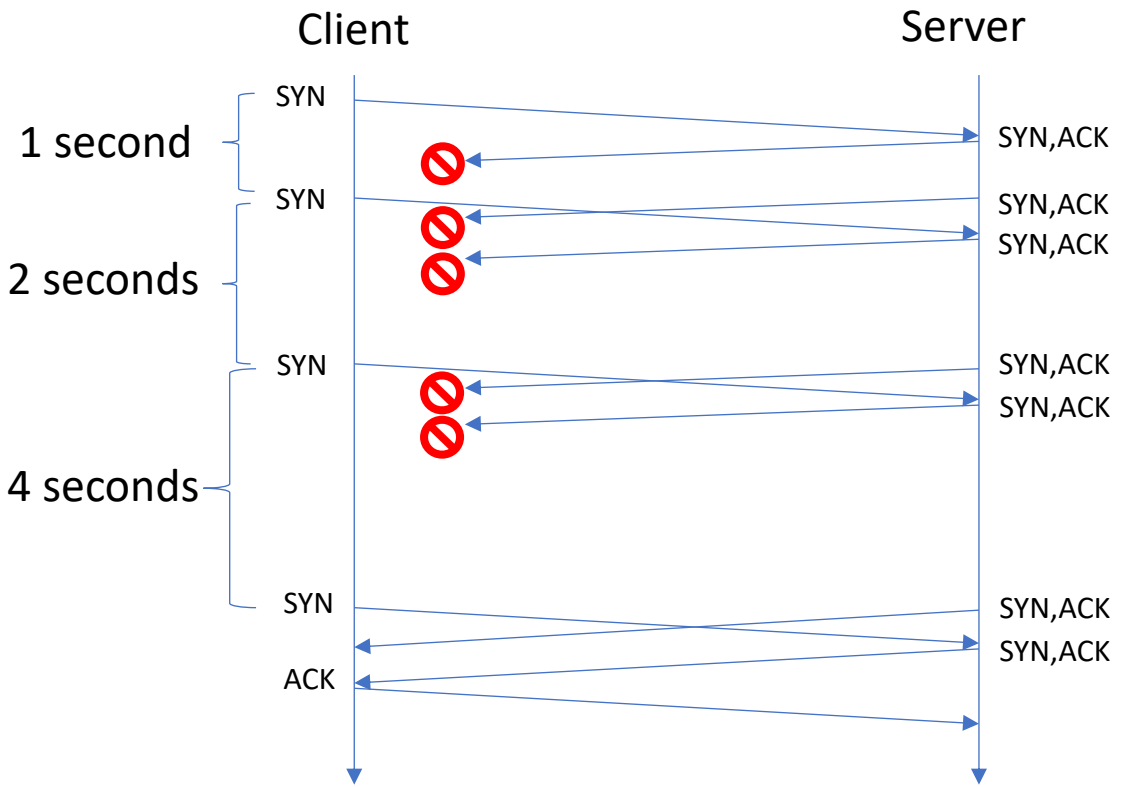


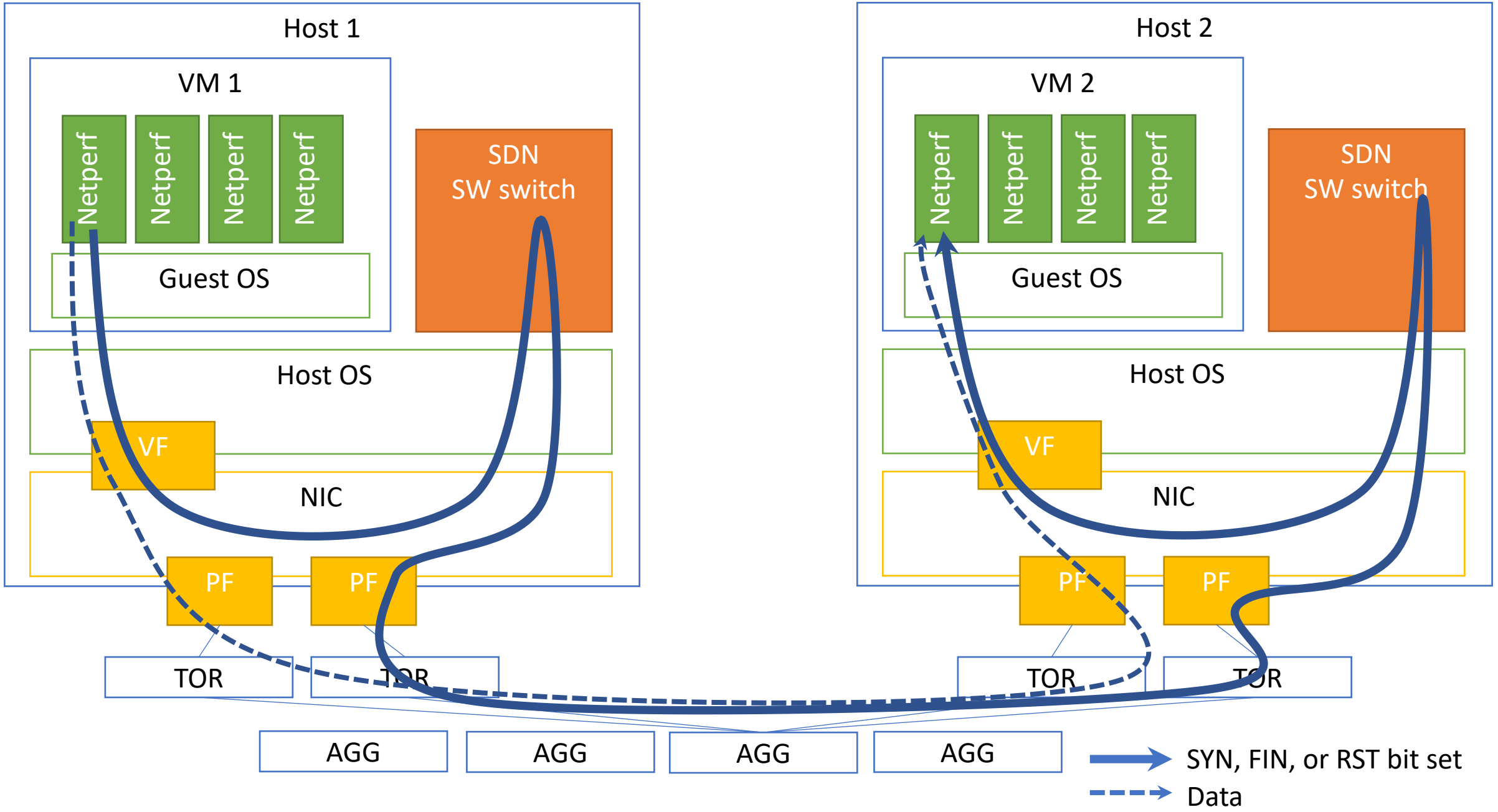
>1 sec Tail Latency of Netperf TCP_CRR



Packets captured on the server

No.	Time (sec)	Source	Destination	Protocol		Packet
1255960	11.586010	100.100.0.7	100.100.0.8	TCP	66	33043 → 12866 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1255961	11.586017	100.100.0.8	100.100.0.7	TCP	66	12866 → 33043 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258593	12.596504	100.100.0.8	100.100.0.7	TCP	66	[TCP Retransmission] 12866 → 33043 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258594	12.611286	100.100.0.7	100.100.0.8	TCP	66	[TCP Retransmission] 33043 → 12866 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258595	12.611301	100.100.0.8	100.100.0.7	TCP	66	[TCP Retransmission] 12866 → 33043 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258605	14.612525	100.100.0.8	100.100.0.7	TCP	66	[TCP Retransmission] 12866 → 33043 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258606	14.627292	100.100.0.7	100.100.0.8	TCP	66	[TCP Retransmission] 33043 → 12866 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258607	14.627312	100.100.0.8	100.100.0.7	TCP	66	[TCP Retransmission] 12866 → 33043 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258613	18.644526	100.100.0.8	100.100.0.7	TCP	66	[TCP Retransmission] 12866 → 33043 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258614	18.723345	100.100.0.7	100.100.0.8	TCP	66	[TCP Retransmission] 33043 → 12866 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258615	18.723375	100.100.0.8	100.100.0.7	TCP	66	[TCP Retransmission] 12866 → 33043 [SYN, ACK] Seq=0 Ack=1 Win=42340 Len=0 MSS=1460 SACK_PERM=1 WS=8192
1258622	18.723574	100.100.0.7	100.100.0.8	TCP	60	33043 → 12866 [ACK] Seq=1 Ack=1 Win=49152 Len=0

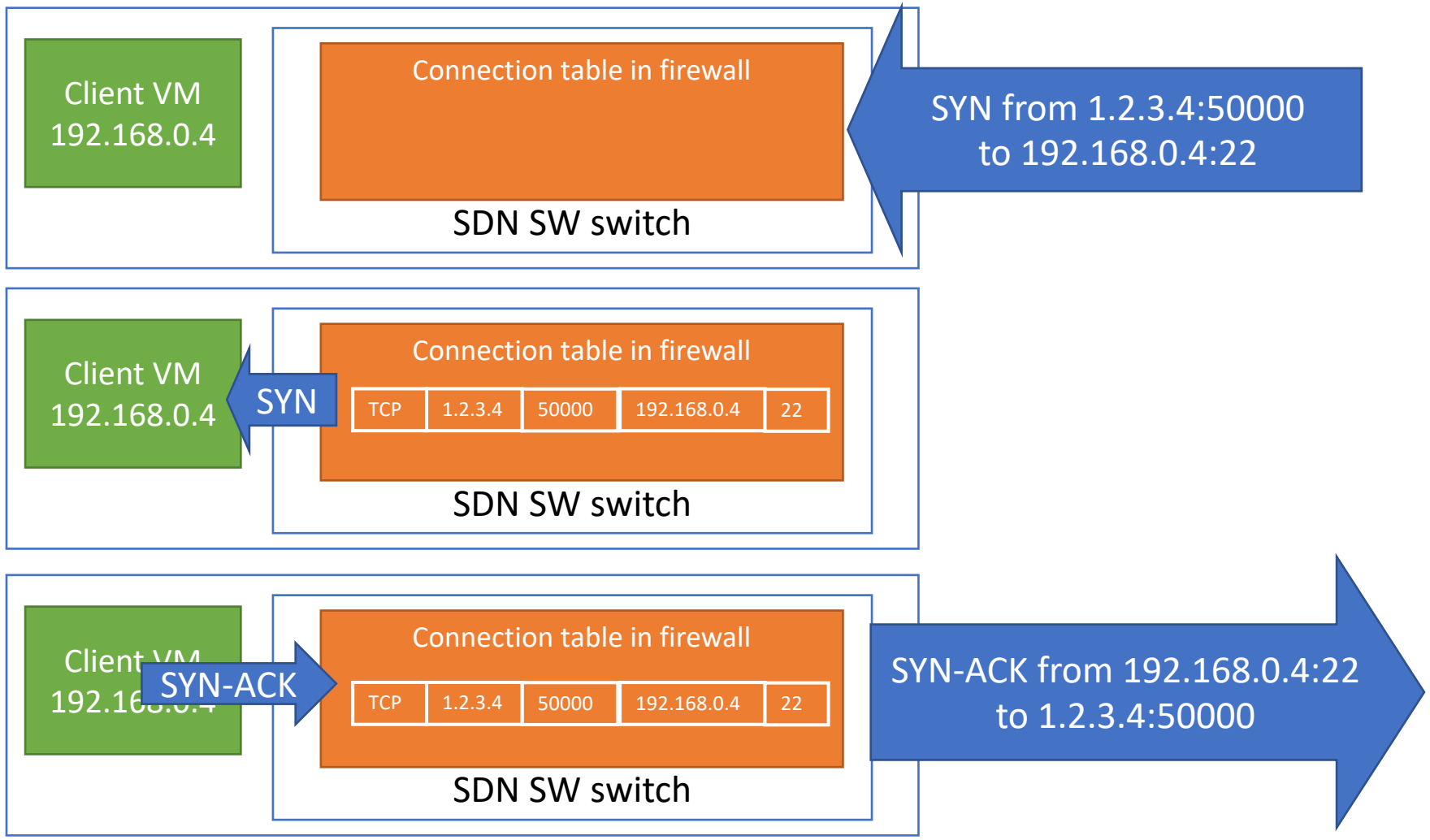




SDN: Software Defined Network

Connection Table in Firewall Module to Implement Stateful Security Group

Example of security group rule: inbound, remote 0.0.0.0/0, TCP port 22



Connection Table Overflow

- Connection table is a hash table with 128K entries
- An entry is retained even after it is closed
  - To allow packets in the TIME-WAIT state

- In Ubuntu, default local (ephemeral) port range has 28232 ports
  - net.ipv4.ip_local_port_range = 32768 60999
- Four netperf clients, each targeting a different server port
- 4x 28232 = 112928 5-tuples =~ 128K

- Mitigated the issue by increasing the table size, but need a fundamental solution

パケット落ちの解析は非常に時間がかかることが多い

- どこでパケットが落ちているかは各キューのパケット落ちカウンタを見ればわかる
  - しかしプロダクション環境では各ノード、スイッチの統計情報を集めるだけで大変
    - 特権オペレータ以外は直接ログインできない
    - ログ基盤に自動的に集計される統計情報も多いが全てではない
  - 包括的なパケット落ち情報の集計、異常検知機構が求められる
- どこでパケットが落ちているかわかっても、原因は自明ではない
  - 単にキューのサイズが小さすぎることもあるが、そういった原因は既に取り除かれていることが多い
  - 今回はプロダクション環境から隔離された実験環境でも再現できたのでマシだった
- パケット落ちの症状は自然と消えてしまうことがある
  - 環境の変化、コードのアップデートなど
  - 潜在的な問題として再び発現するので困る
  - 迅速な原因究明が求められる

カーネルの並列性バグ

- 並列性バグを検出する研究は数多い
- カーネルの開発過程や検証過程で適用可能な手法が求められる

Software Defined Overlay Network におけるコネクション管理

- クラウドのネットワーク基盤ではコネクション管理が至る所で必要とされる
  - ファイアウォール
  - Network Address Translation (NAT)
    - Public Gateway, Service Gateway, etc.
- コネクション管理テーブルの容量や置き換えアルゴリズムに起因するパケット落ちがよく見られる
  - テーブルのエントリーをいつ再利用するか
  - データパスのクリティカルパスに入っているので高速に処理したい
  - NATではポートの割り当てアルゴリズムも重要
  - 裏で不要エントリーを回収するゴミ集めスレッドのようなものは、なるべく走らせたくない

パブリッククラウドにおける性能上の課題

- Control Plane
  - Time to create/destroy resources
    - How long does it take to provision 1000 VMs?
- Data Plane
  - Compute
  - Network
    - Packet drops, Throughput bottleneck
  - Storage
    - IOPS bottleneck, Encryption
  - Hardware
    - Stability

まとめ

- パブリッククラウドでは様々な性能上の課題が生じる
- パブリッククラウドの仮想化ネットワークにおけるパケット落ちの迅速な原因調査には多くの困難がある
  - Linux Macvtapレイヤーにおける並列性バグに起因するパケット落ち
  - SDNのコネクション管理に起因するパケット落ち