



Possibility of Userspace Networking Drivers: TUNTAP, VDUSE and What's More?

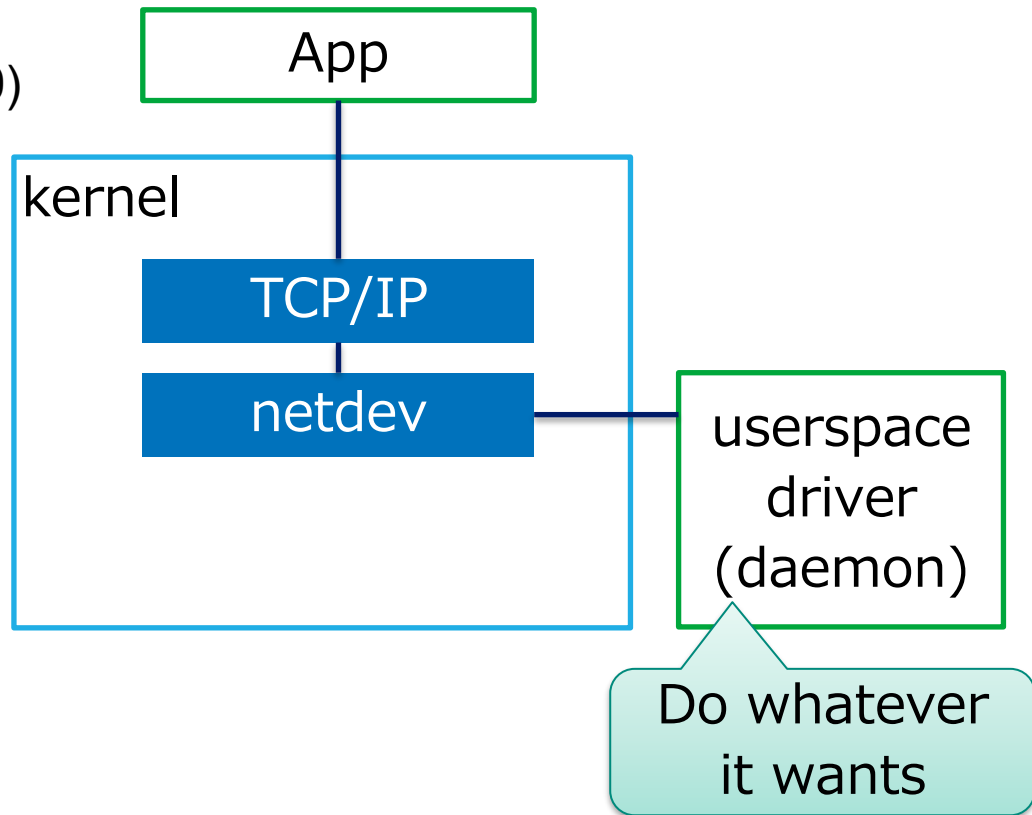
Toshiaki Makita

Oct 7, 2026

© NTT, Inc. 2026

What is a “userspace networking driver” here?

- Linux network device (e.g., eth0)
- Driver logic in userspace (daemon)
- Not DPDK; does not bypass kernel network stack



Use case: for real NIC

No kernel crash, no upstreaming

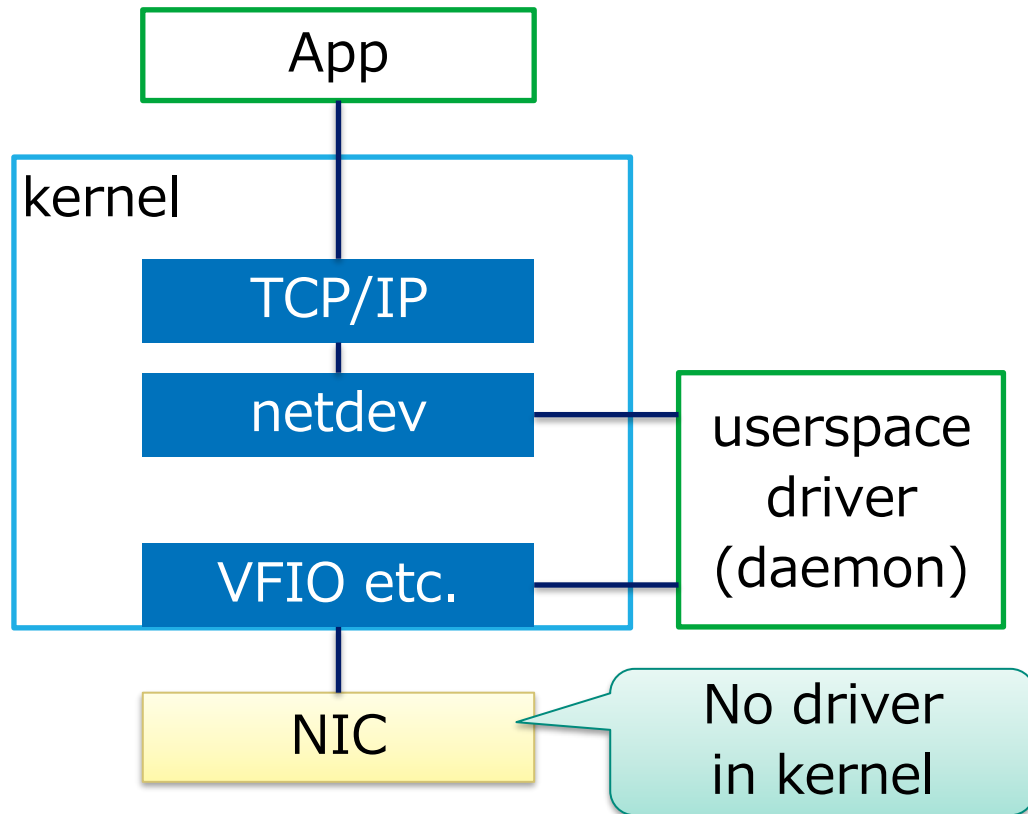
1. New NICs for:

- Small business
- Private use

E.g. FPGA NICs
(free layout of registers/
descriptors)

2. Obsolete (deleted) old drivers

Cons: low performance?



How to do it?

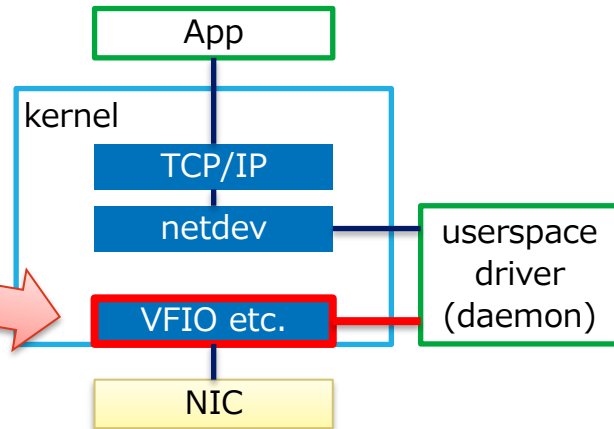
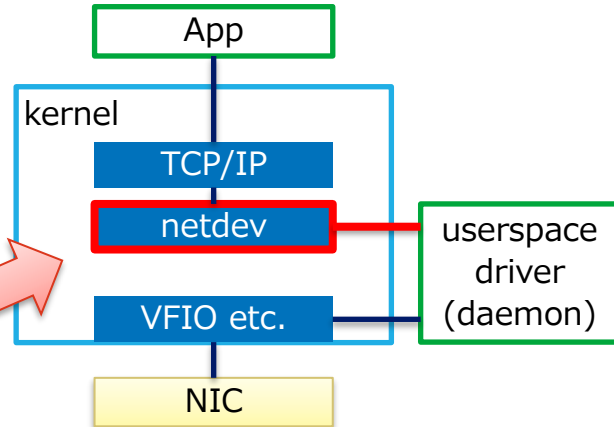
The current kernel already has building blocks

- Network stack $\leftrightarrow$ userspace driver

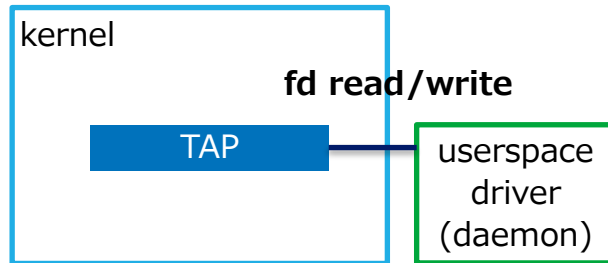
- TAP
- VirtIO-VDPA + VDUSE

- Userspace driver $\leftrightarrow$ NIC

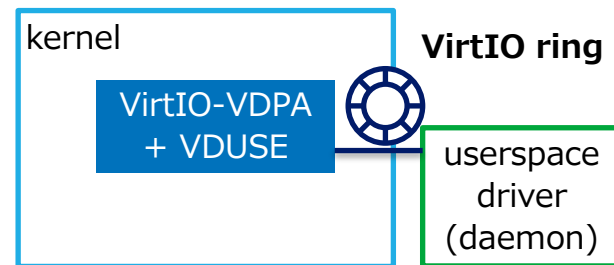
- VFIO/IOMMUFD
- UIO



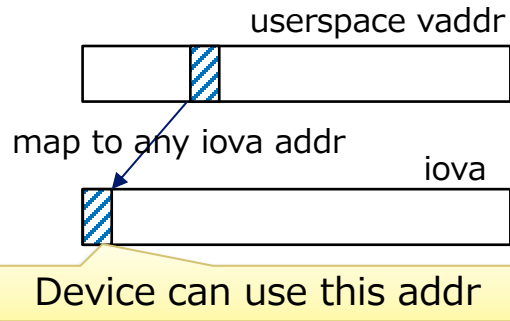
- Classic file-based read/write interface
- No zerocopy, No recvmmsg/sendmmsg (batch recv/send)
 - Socket IF and ZC for only in-kernel users (vhost-net)
- io_uring allows multishot write (batch write)
- io_uring multishot read (batch read) is possible, but not with readv
 - 64KB buffers are always required for multishot read



- VDPA: framework to implement VirtIO device on hardware with VirtIO ring
- VirtIO-VDPA: VDPA for host virtio drivers, e.g. virtio-net
- VDUSE: VDPA backend for userspace instead of hardware
- VirtIO-VDPA + VDUSE: virtio device with userspace backend

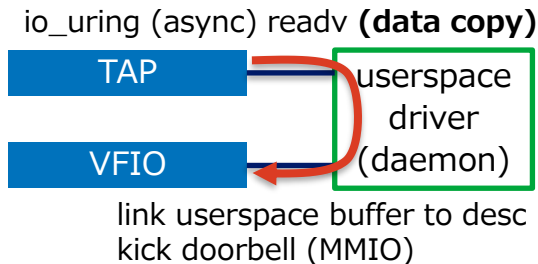


- IOMMU-based framework for userspace device drivers
- VFIO-PCI provides:
 - PCI configuration space via fd pread/pwrite
 - BAR (MMIO/PIO) access via mmap/pread/pwrite
- DMA map interface
- IRQ notification via eventfd
- Example existing usage:
 - QEMU: guest device assignment
 - DPDK: kernel bypass networking

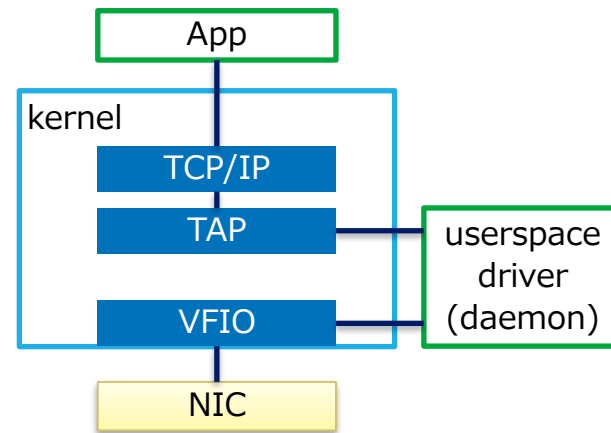
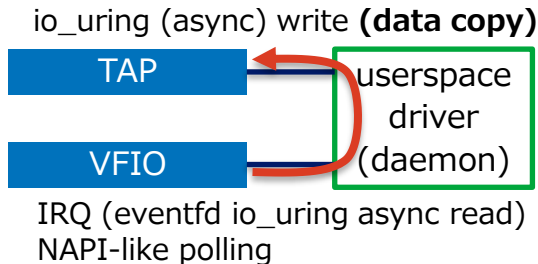


Combined solution 1: TAP + VFIO

TX:



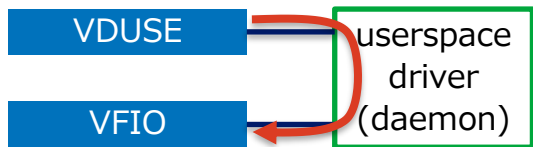
RX:



Combined solution 2: VirtIO-VDPA + VDUSE + VFIO

TX:

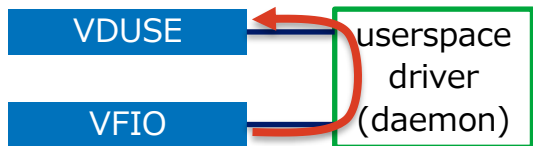
buffer from vring (**data copy** in dma_map)



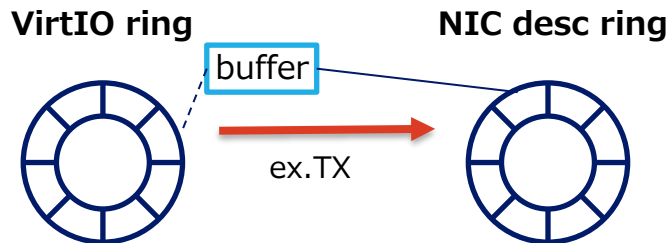
link userspace buffer to desc
kick doorbell (MMIO)

RX:

link buffer to vring (**data copy** in dma_unmap)



IRQ (eventfd io_uring async read)
NAPI-like polling



Relink buffers between rings
No copy here
But a copy still happens in virtio_net
(dma_map/unmap) between skb
data and bounce buffer

Performance measurement environment



- HPE ProLiant DL385 Gen10 Plus v2
 - FPGA NIC: Silicom N6010 (100GbE)
 - Requires proprietary IP / internal project; not suitable for community discussion

Performance measurement environment

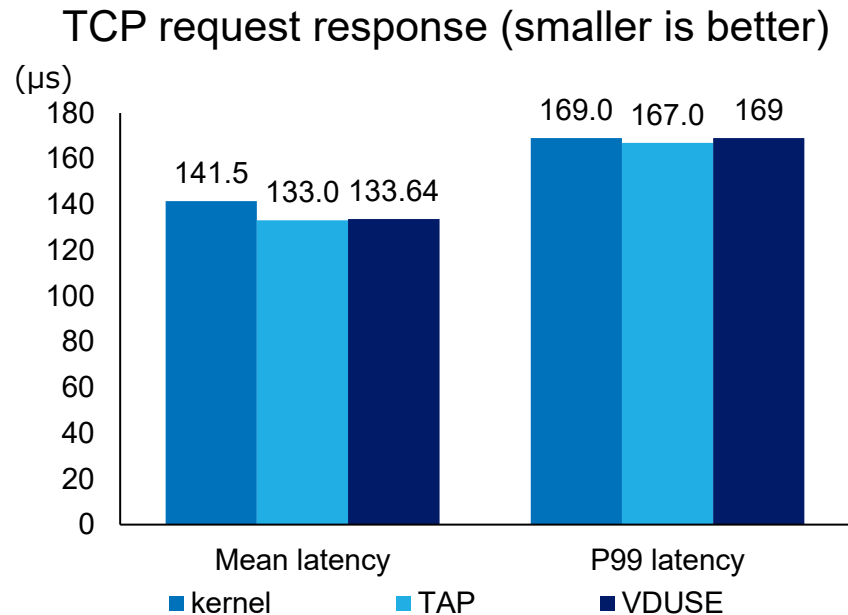
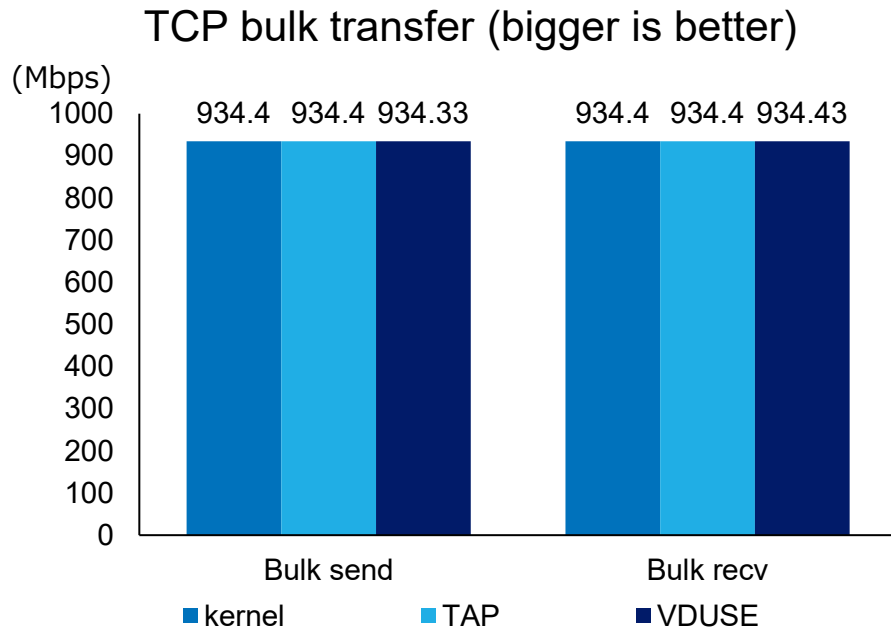


- HPE ProLiant DL385 Gen10 Plus v2
 - FPGA NIC: Silicom N6010 (100GbE)
 - Requires proprietary IP / internal project; not suitable for community discussion
- KVM Guest on HPE ProLiant DL360 Gen11
 - NIC: emulated Intel 82574L (e1000e)
 - Little difference between kernel e1000e and userspace driver: MMIO (VMExit) dominates

- HPE ProLiant DL385 Gen10 Plus v2
 - FPGA NIC: Silicom N6010 (100GbE)
 - Requires proprietary IP / internal project; not suitable for community discussion
- KVM Guest on HPE ProLiant DL360 Gen11
 - NIC: emulated Intel 82574L (e1000e)
 - Little difference between kernel e1000e and userspace driver: MMIO (VMExit) dominates
- HP Z4 G4 workstation
 - CPU: Xeon W-2223 4 cores, 1 socket @3.60GHz
 - NIC: Intel I219-LM2 (1GbE, e1000e, No TSO 🤔) ... let's make the best of it

- Flows: single
- Protocol: TCP
- Metrics: bulk transfer throughput and request response latency
- Tool: netperf
- CPU affinity: core 2 → NIC IRQ and userspace driver daemon; core 3 → netperf
- Offload: TX/RX csum, GSO and GRO (no TSO, HW-GRO, or LRO)
- OS: Rocky Linux 10.2
- Linux kernel: net-next 91ec20351349 (“Merge tag ‘net-next-7.3’ of ...”), Aug 20, 2026
- Driver:
 - kernel: e1000e (baseline)
 - TAP: TAP + VFIO
 - VDUSE: Virtio-VDPA + VDUSE + VFIO

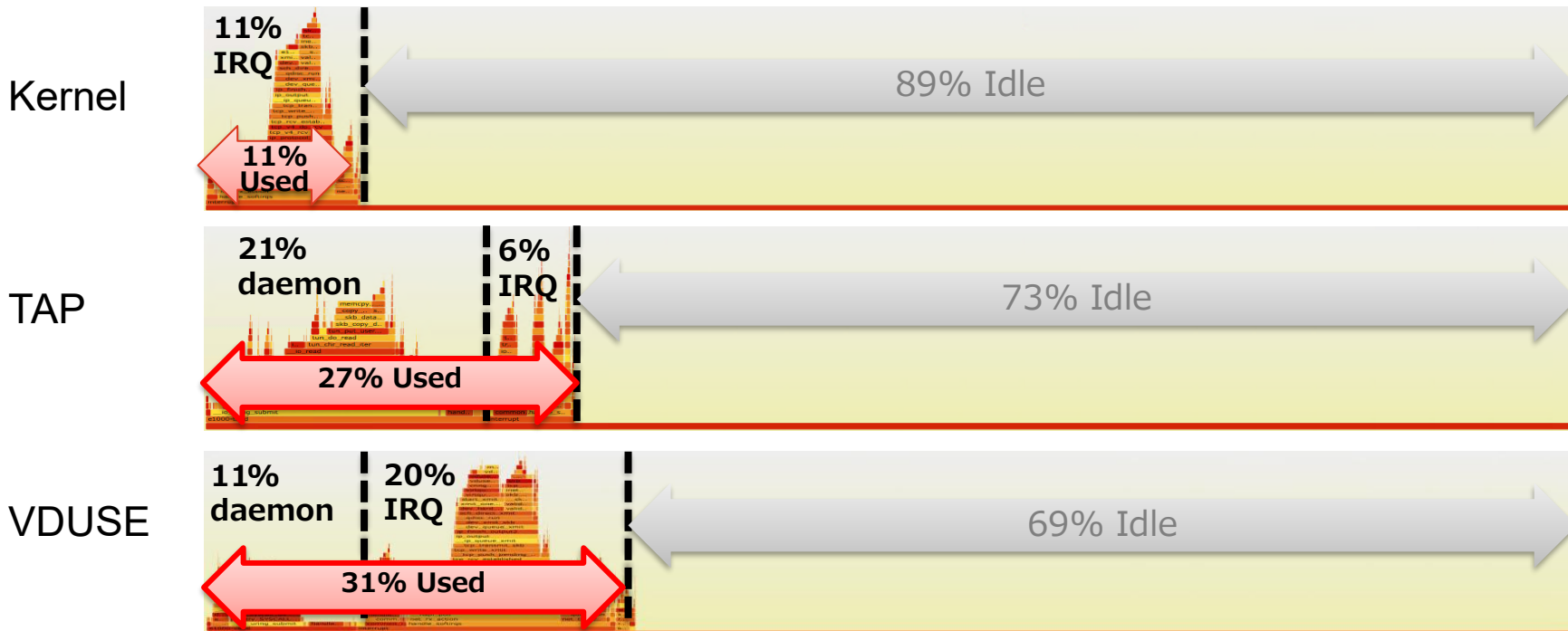
Performance comparison



Throughput hits line rate, userspace driver's latency is on par with or better than the kernel driver
(Probably the latency difference comes from device configuration, not the kernel)

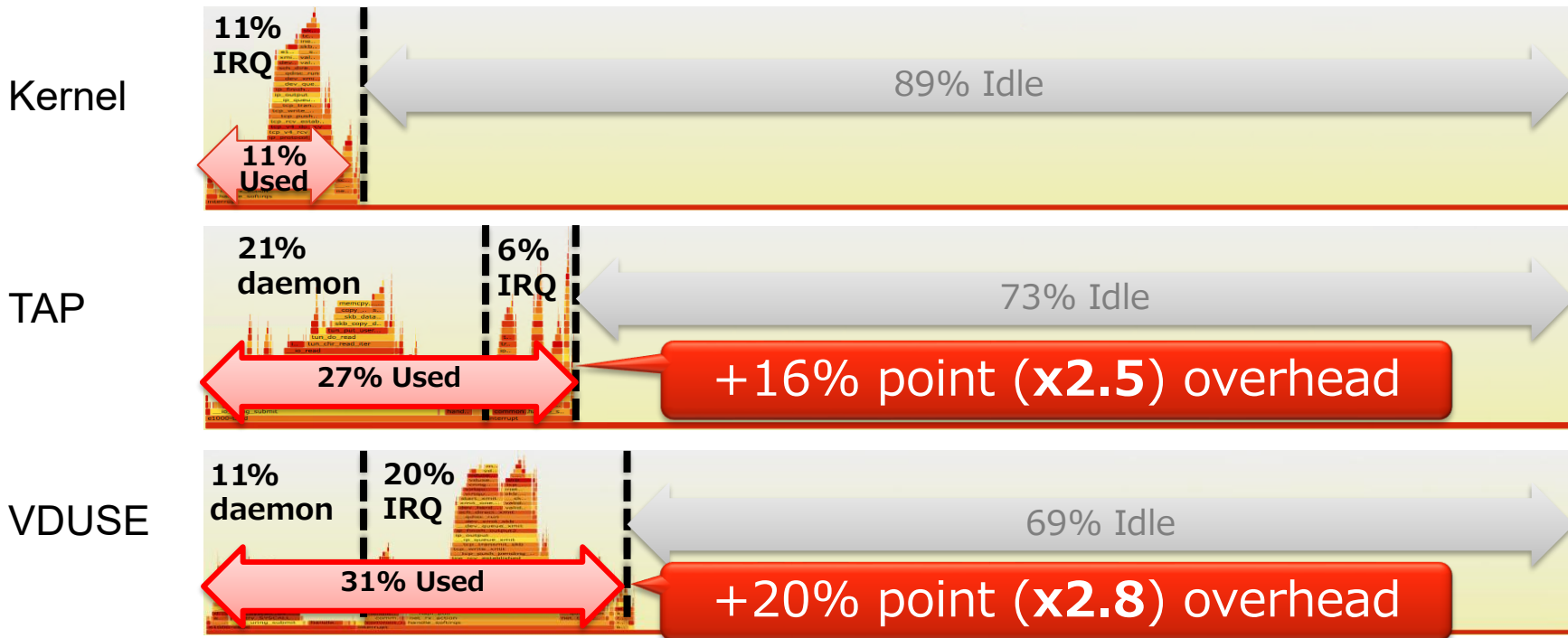
CPU Usage (flamegraph) Bulk Send

- CPU usage of core 2, where SoftIRQ and the userspace daemon are running



CPU Usage (flamegraph) Bulk Send

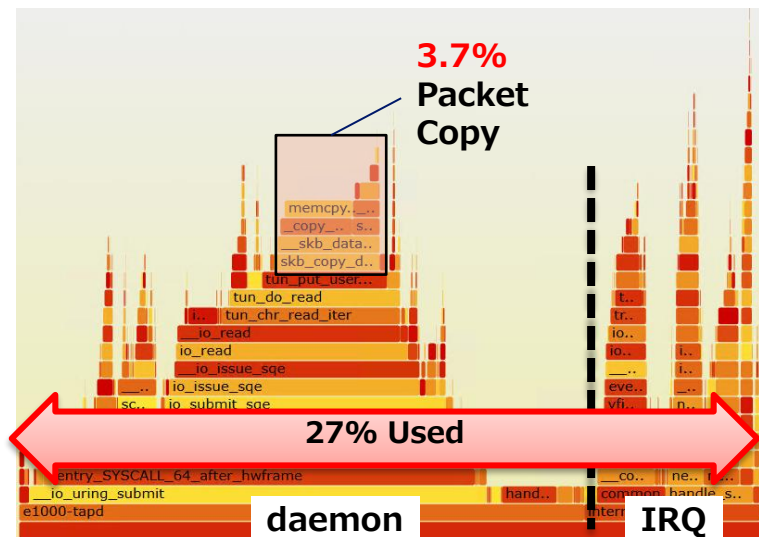
- Much larger overhead with userspace driver



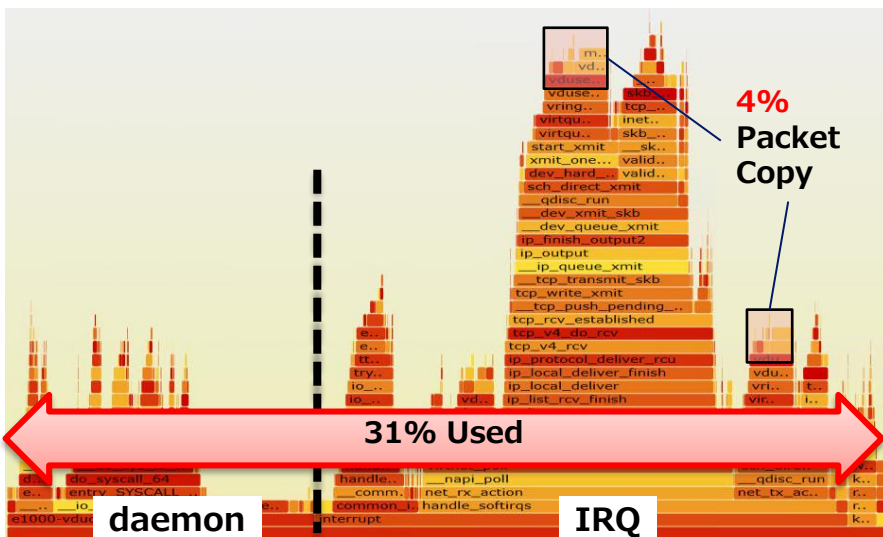
CPU Usage Bulk Send: Zoom up

- Packet copy is costly, but the rest (per-packet cost) is far heavier (without TSO)
 - TAP/VDUSE processing, syscall / context switches, io_uring, ...

TAP

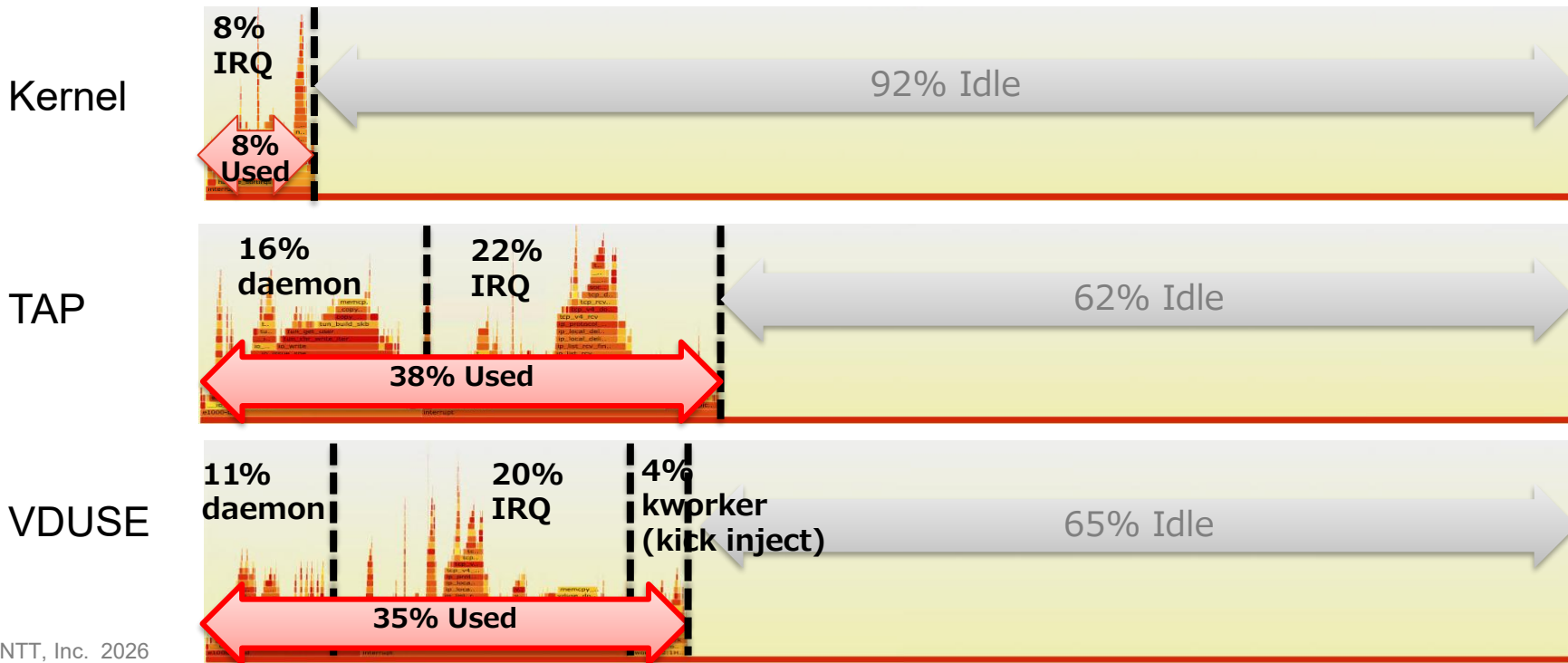


VDUSE



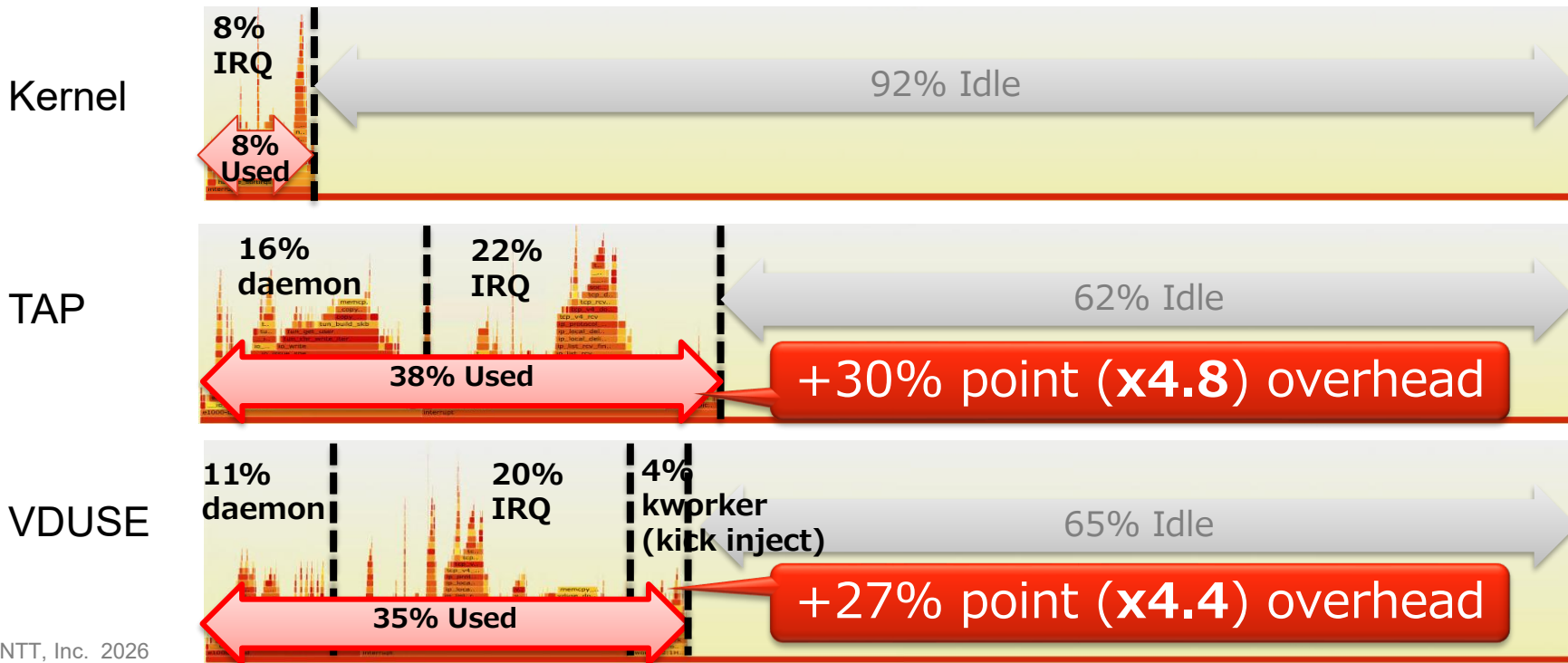
CPU Usage (flamegraph) Bulk Recv

- CPU usage of core 2, where SoftIRQ and the userspace daemon are running



CPU Usage (flamegraph) Bulk Recv

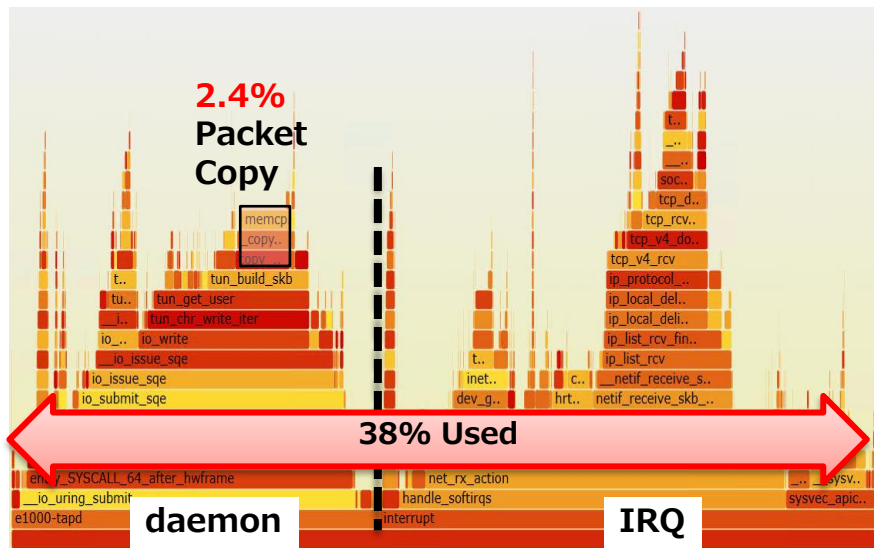
- Even worse than send



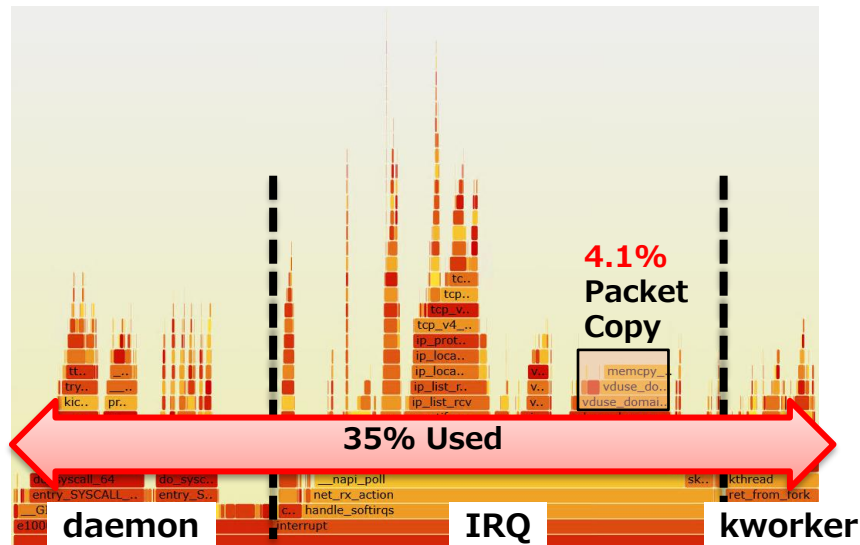
CPU Usage Bulk Recv: Zoom up

- Again, packet copy is not cheap but not dominant (without HW-GRO/LRO)

TAP



VDUSE



Eliminate per-packet overhead (TAP/VDUSE processing, syscall / context switch, io_uring...)

1. Network device with unified interface (VirtIO, IDPF, or some other ones)

- Devices can use existing drivers; no userspace drivers are needed
- FPGAs may be able to support it, but the fixed interface limits what the device can do
- What about NICs other than FPGAs? Or old devices?

2. Declarative generic network driver

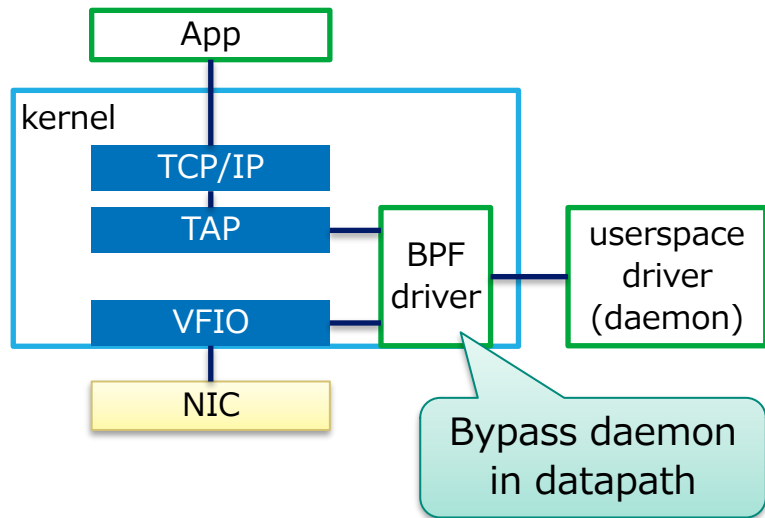
- A new driver which determines descriptor/register layout from userspace configuration
- Too limited to handle existing devices' descriptor/register layout variations

3. Programmable (BPF-based) network driver

- Allows more flexible layouts, eliminates the daemon's per-packet overhead
- Is this feasible? How? Try it and see what happens

BPF-based driver: building parts in BPF

- Idea: bypass the daemon in the datapath with BPF
 - TX: `ndo_start_xmit` as BPF prog
 - RX: IRQ handler and `napi->poll` as BPF prog
 - Descriptor ring: DMA map (use VFIO) in daemon, shared with BPF through mmapable BPF map
 - Packet buffer: DMA map (use VFIO) in daemon, used in TAP as buffer pool (copy remains)
 - MMIO: `kfunc` (`mmio_read/write`) and `kptr` MMIO address from VFIO (userspace daemon)
 - Memory barrier: `kfunc` (`dma_rmb/wmb`)



BPF-based driver: building parts in BPF

- Idea: bypass the daemon in the datapath with BPF

- TX: `ndo_start_xmit` as BPF prog
- RX: IRQ handler and `napi->poll` as BPF prog
- Descriptor ring: DMA map (use VFIO) in daemon, shared with BPF through mmapable BPF map
- Packet buffer: DMA map (use VFIO) in daemon, used in TAP as buffer pool (copy remains)
- MMIO: kfunc (`mmio_read/write`) and `kptr` MMIO address from VFIO (userspace daemon)
- Memory barrier: kfunc (`dma_rmb/wmb`)

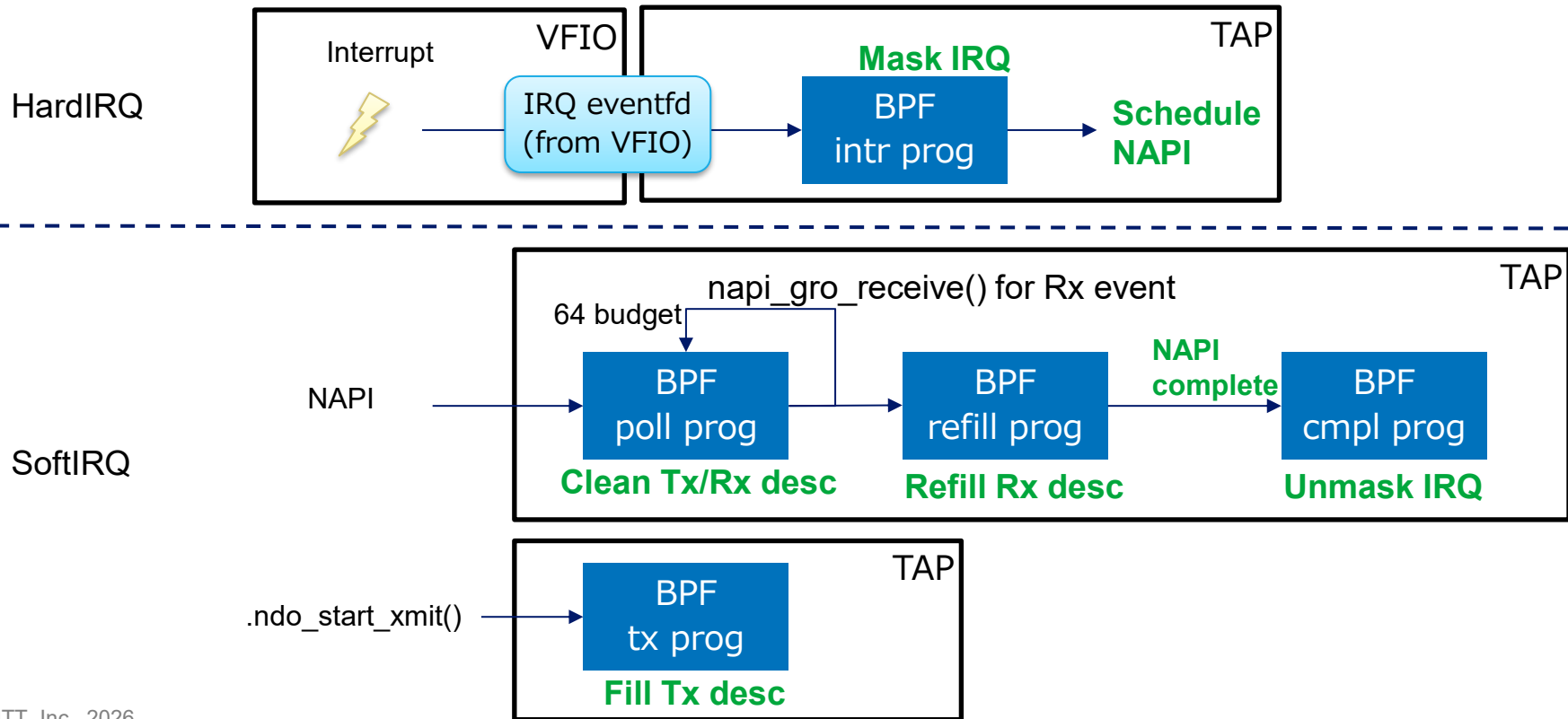
Attach point (or `struct_ops`)
Not in current kernel

Covered by existing BPF features

kfunc
Not in current kernel
MMIO is restricted to regions VFIO provides
(the same as what userspace can do)

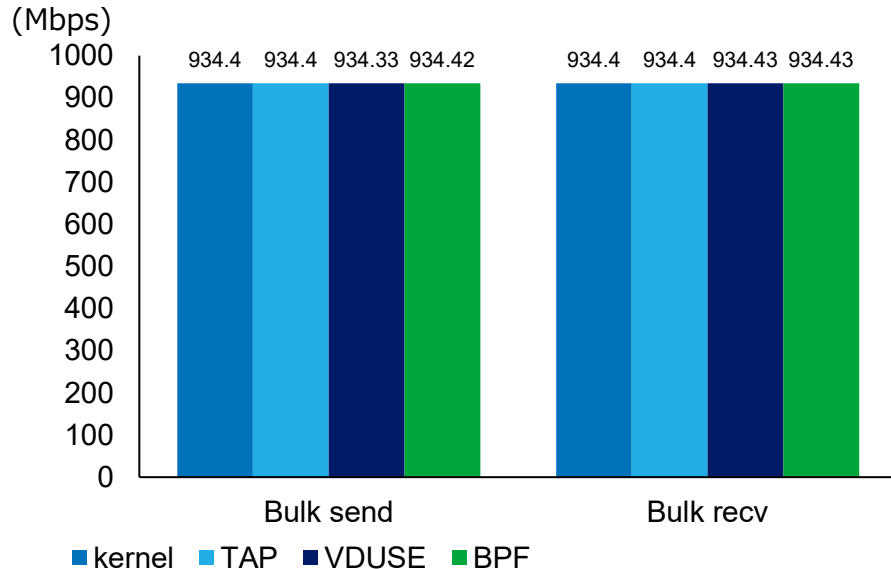
PoC BPF driver concept (in TAP, for combined queue)

- Most drivers share a **common pattern**, push only device-specific parts into BPF progs

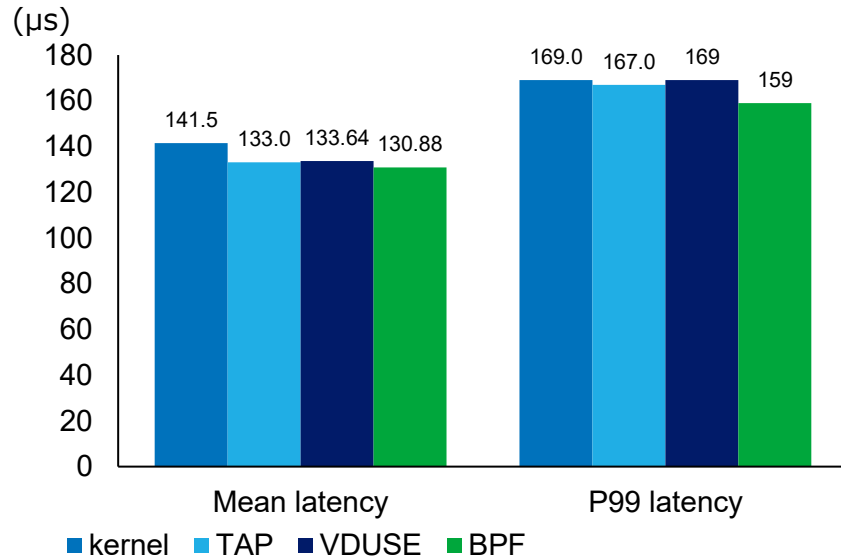


Performance comparison with BPF

TCP bulk transfer (bigger is better)



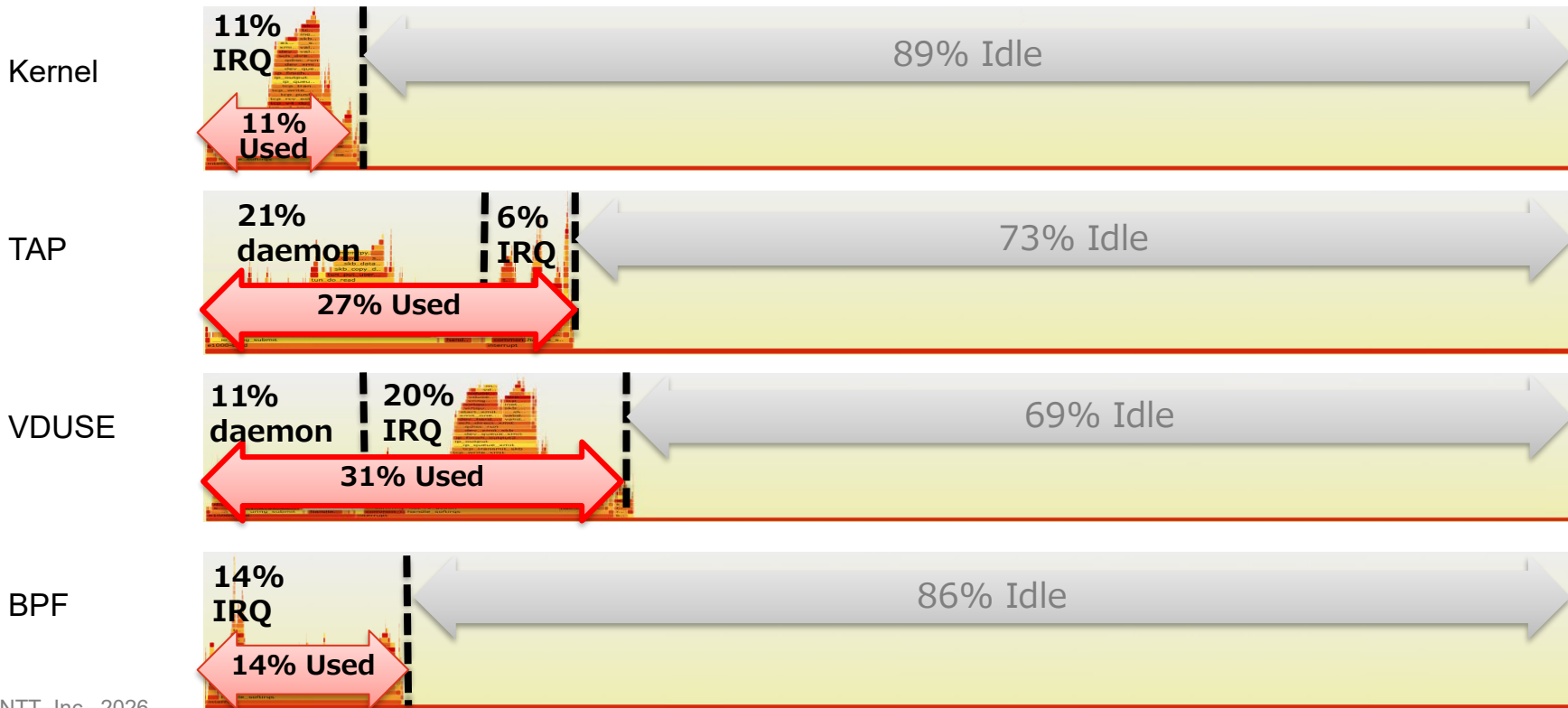
TCP request response (smaller is better)



BPF has slightly better latency than the other userspace drivers

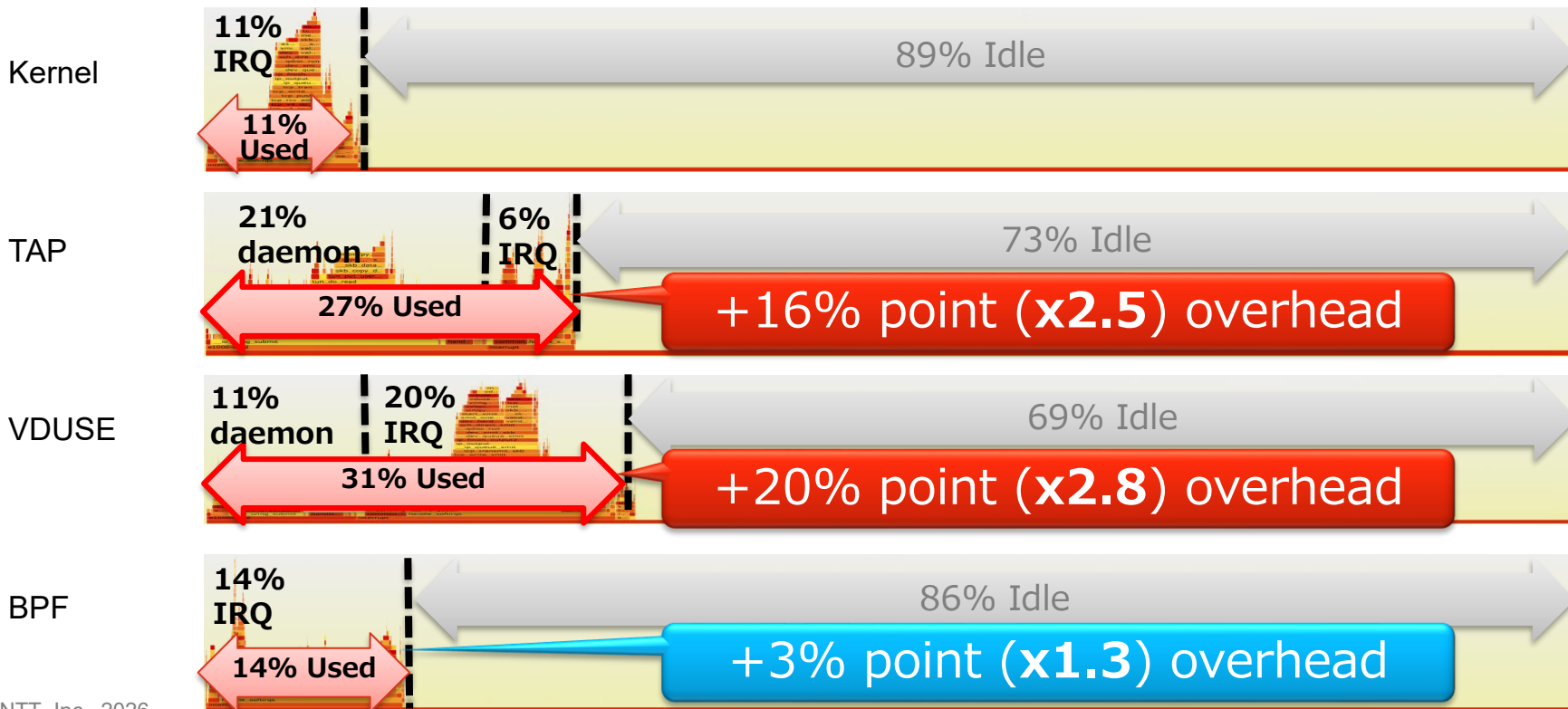
BPF CPU Usage Bulk Send

- CPU usage of core 2, where SoftIRQ and the userspace daemon are running



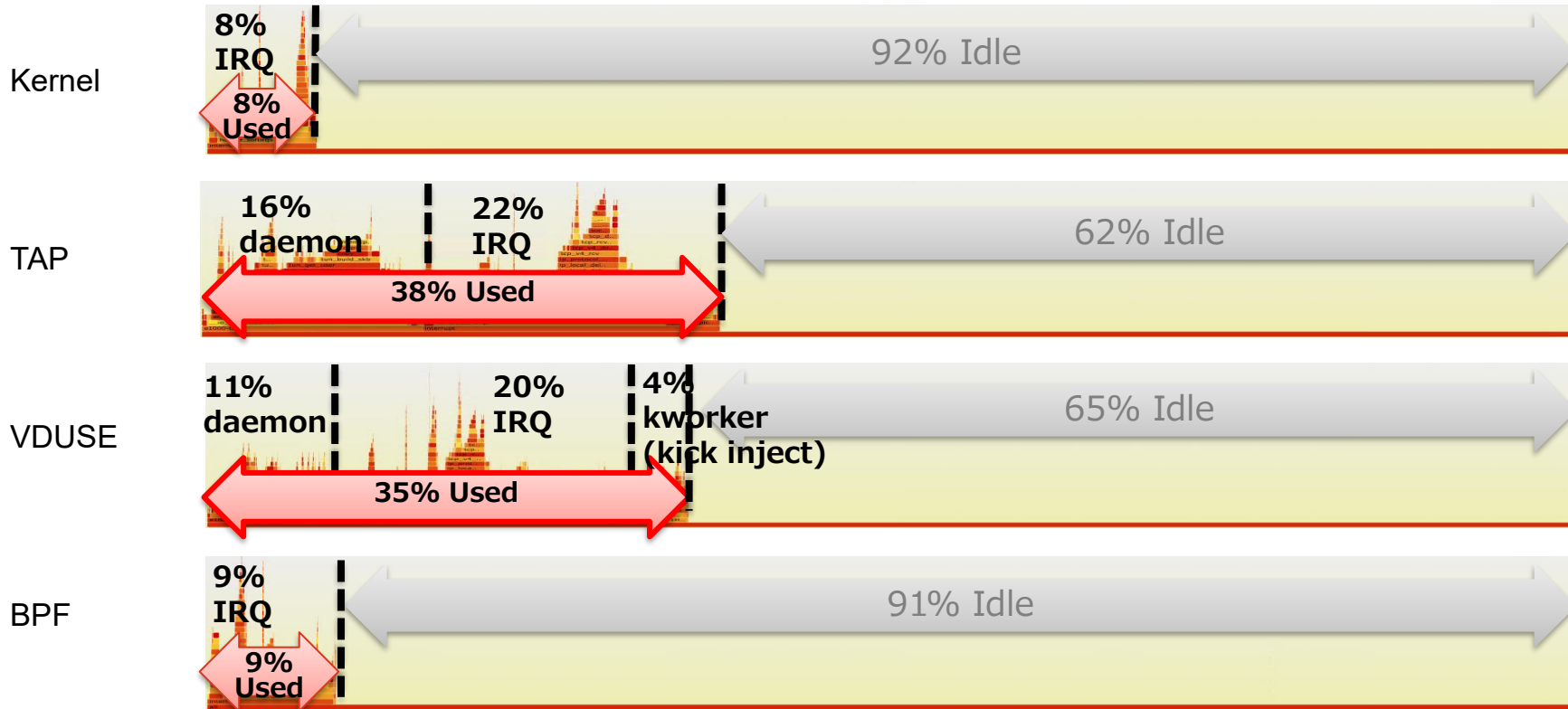
BPF CPU Usage Bulk Send

- CPU usage improved: +16-20% point (x2.5-2.8) → +3% point (x1.3)



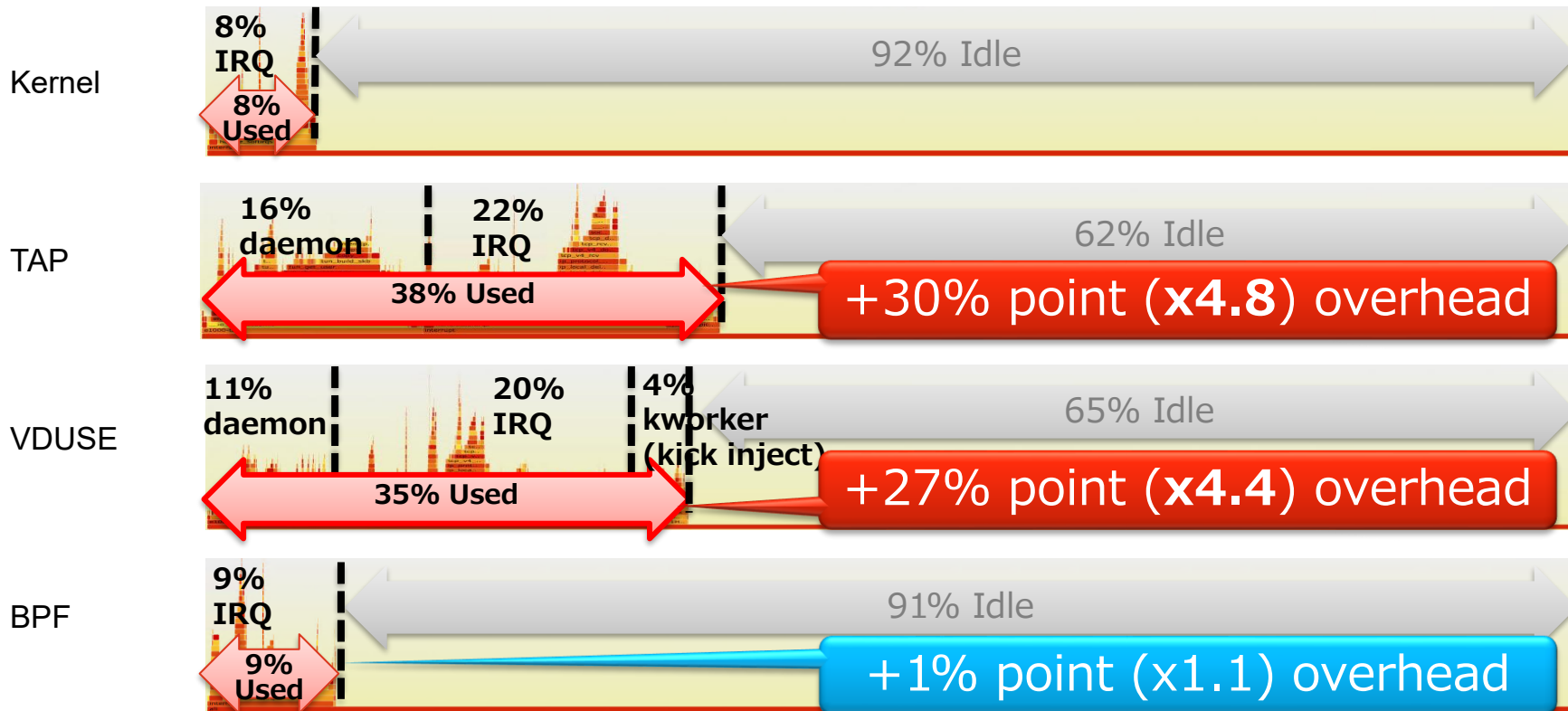
BPF CPU Usage Bulk Recv

- CPU usage of core 2, where SoftIRQ and the userspace daemon are running



BPF CPU Usage Bulk Recv

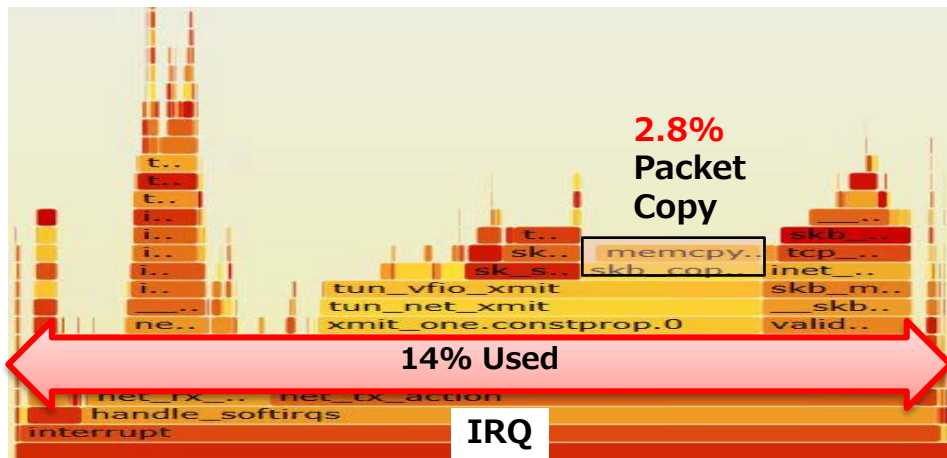
- CPU usage improved: +27-30% point (x4.4-4.8) → +1% point (x1.1)



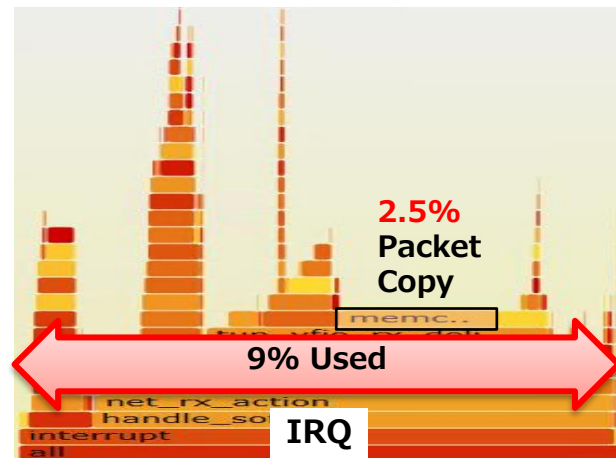
BPF CPU Usage Bulk Send/Recv: Zoom up

- Extra packet copy remains in current BPF PoC
 - TSO/HW-GRO would make it a bottleneck
 - Rx copy can be removed, Tx looks difficult (socket needs to allocate driver-supplied memory?)
- Excluding the copy, CPU usage is almost the same as the kernel e1000e driver
 - Need more tests: 10G/25G/100G, TSO/HW-GRO

Send



Recv



More Challenges

- Device-specific userspace interface: ethtool, ip-link, sysfs hierarchy, ...
 - Stats: no real hardware stats
 - Features: limited to TAP (or VirtIO with VDUSE) features; TAP has HW_CSUM but my e1000e only supports IPV4 CSUM...
 - Coalesce, Speed/duplex, Ring size, Desc size, ... none of these can be configured
- PCI-level problems: suspend/resume, PCI error
 - Cannot detect system suspend or stop device
 - Silent device reset on fatal error, cannot restore device state
- Daemon crash
 - With current implementation, the netdev disappears when the daemon crashes (restarting the daemon restores the NIC)
 - Need a way to recover without recreating the netdev
- BPF security / safety
 - MMIO from BPF must be no less safe than userspace VFIO; at least revoking access on device reset is needed
 - Other concerns?

Limitations / Pitfalls

- Limitation: IOMMU required
 - VFIO uses it to safely use devices in userspace
 - noiommu mode exists but is unsafe
- Pitfall: Data format inconsistency
 - Inline (VDUSE) vs outbound (e1000e) metadata
 - Buffer size limitation difference

Pitfall: Data format inconsistency

- “VDUSE: inline virtio_net_hdr” vs “e1000e: descriptor for metadata”
 - Rx requires headroom
 - HW-GRO/jumbo frame support is complicated with this model:
 - › VirtIO (mergeable rxbuf) requires headroom only for first buffer, second buffer must start with offset 0
 - › e1000e cannot know in advance where the first buffer of the next packet will be
 - › Reserve headroom for all buffers, copy last 12 bytes of each buffer to next buffer’s headroom...
 - TAP is OK as userspace driver can add an extra buffer for virtio_net_hdr to iovec
- Buffer size limitation difference
 - vIOMMU requires contiguous GPAs; otherwise, packets are silently dropped
 - NICs also have limitations; the userspace driver needs to split buffers properly anyway

And what's more?

- CPU usage
 - BPF has a big effect
 - Other options?
- Netdev
 - TAP/VDUSE or a dedicated new netdev for ethtool/BPF etc?
 - Netdev with concept similar to ublk / FUSE
 - › Most ethtool ops need information prefilled by userspace, or upcalls (VDUSE does similar)
- PCI-level problems (suspend/resume, PCI error)
 - Not sure, probably need to study VFIO more

- Introduced TAP+VFIO and VirtIO-VDPA+VDUSE+VFIO as ways to implement userspace drivers for real NICs
- CPU usage is much higher than with the kernel driver
 - BPF can significantly reduce it, but need more testing (e.g., 100G, TSO/HW-GRO)
- What's more?
 - BPF datapath (new attach points/struct_ops and kfuncs)?
 - Dedicated userspace netdev for BPF/ethtool etc.?

Innovating a Sustainable Future for People and Planet

