Introduction

Number of transistors doubles every 18-24 months

Confused w/ performance because as something get smaller. it can be modified quickly

Stopped improving clock speeds since 2005-2006

Access link bandwidth have 4-10x times in the same time

As a result problem of efficient network design is being extensively explored

- → kernel network stack optimizations
- → hardware offloads
- → clean slate userspace network stacks

Linux network stack

1 Setup

i) Driver first loads itself into kernel → Informs device ID which it can control → Install function pointers to setup device

ii) Then kernel needs to figure out which device driver to control each device

iii) kernels calls probe() function of the device driver which sets up the device

- setting up memory → Talk about Rx queue here
- init of struct net_device [representing network device]
  - sets function pointers for operations that network subsystem needs to control the device

[Eg: ndo_start_xmit : igb_xmit_frame]

1.1 IRQ and NAPI

Packet → IRQ → packet processing

Bottleneck on high packet rate

NAPI allows device drivers to register a poll() function

Register an interrupt handler

Enable interrupts

x —— Now device is up ——x

(2) Soft IRQs

Way to execute driver code outside the interrupt handler

Why? Cannot do much, work needs to be deferred to softirq context

Series of kernel thread (one per CPU)

↳ run handler functions registered for different softirq events

[Registration happens during setup]

(3) Data arrives

Packet arrives at NIC

It picks a descriptor off an RX ring

DMAs packet into RAM using descriptor

Raises an interrupt → Talk about tuning interrupt [IRQ affinities OR coalescing]

Interrupt handler disables further interrupts and trigger NET_RX_SOFTIRQ softirq

let's the kernel look at the softirq thread when its looking to schedule something to run on CPU

Ksoftirq get scheduled to run on CPU

↳ calls loop() function registered before

loop() is bounded ↓ harvests packets from NIC

weight and time → configurable

After NAPI loop finishes

disable NAPI + enable interrupts

↓ opposite of triggering

NET_RX_SOFTIRQ

Inside poll

Picks a deccriptor from Rx ring and builds skb around the packet

Frees the descriptor for the NIC to use

Tries to get the whole frame into skb

Passes this skb to napi_gro_recieve()

GRO tries to coalesce packets

↳ makes sense because less processing for kernel to do

You can choose to switch this off

Hand off packet to protocol layer

(4) Talk about RSS and RPS

Why is RSS important

How does it happen → via an indirection table

4-tuple hash + mask give which entry in indirection table

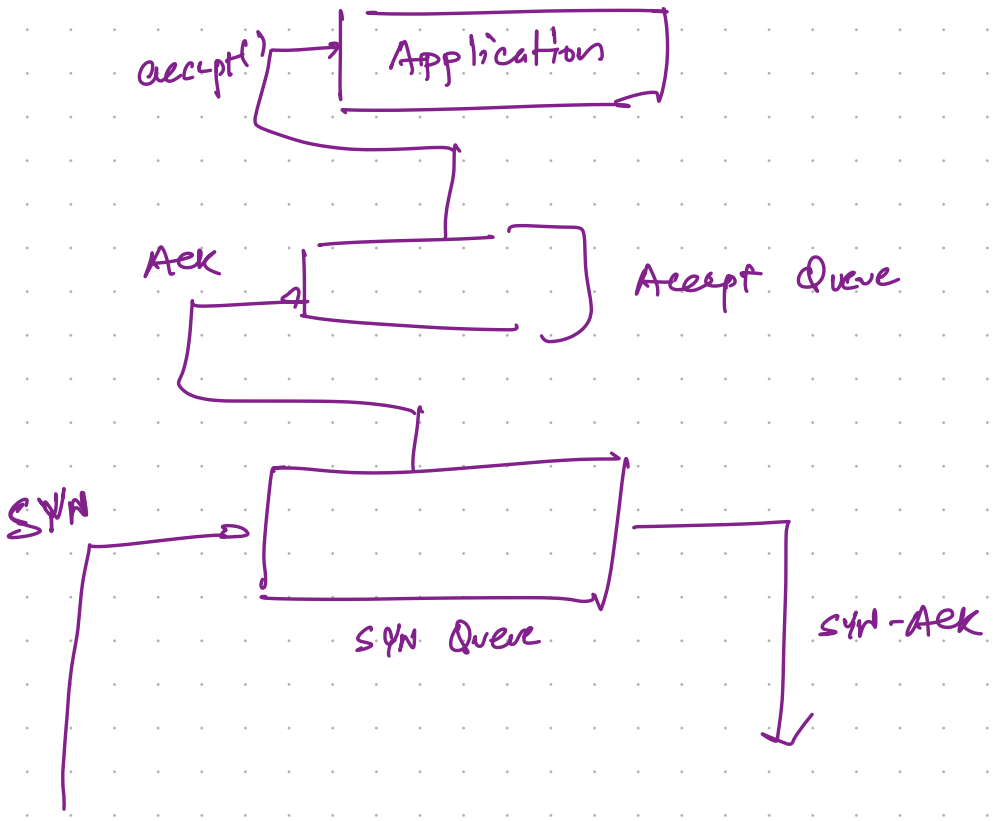
Advanced NICs can configure complicated rules to match packet with RX queue

RPS is a software implementation of RSS

Why needed ? Hardware support of multiple RX queues not needed + advance filters in software

(5) Listen socket and SO_REUSEPORT

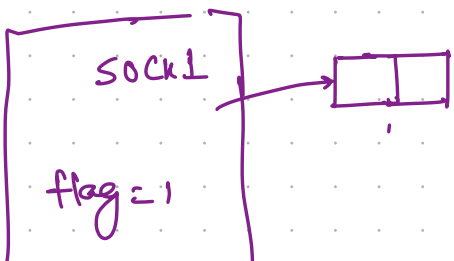
Listen socket → passive socket [Talk about how applications use]



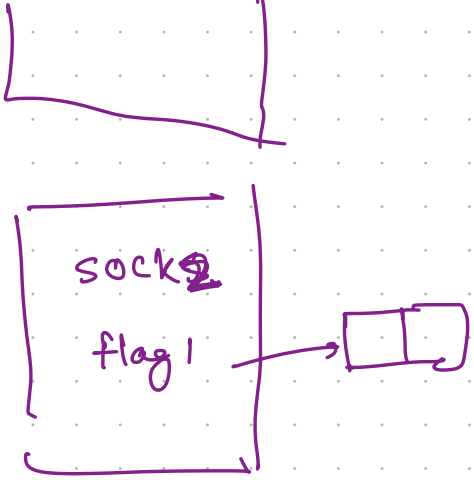
Talk about the scaling bottleneck with this approach

Introduce SO_REUSEPORT and how it solves this problem

Talk about how SO_REUSEPORT works



Emphasize why socket selected randomly

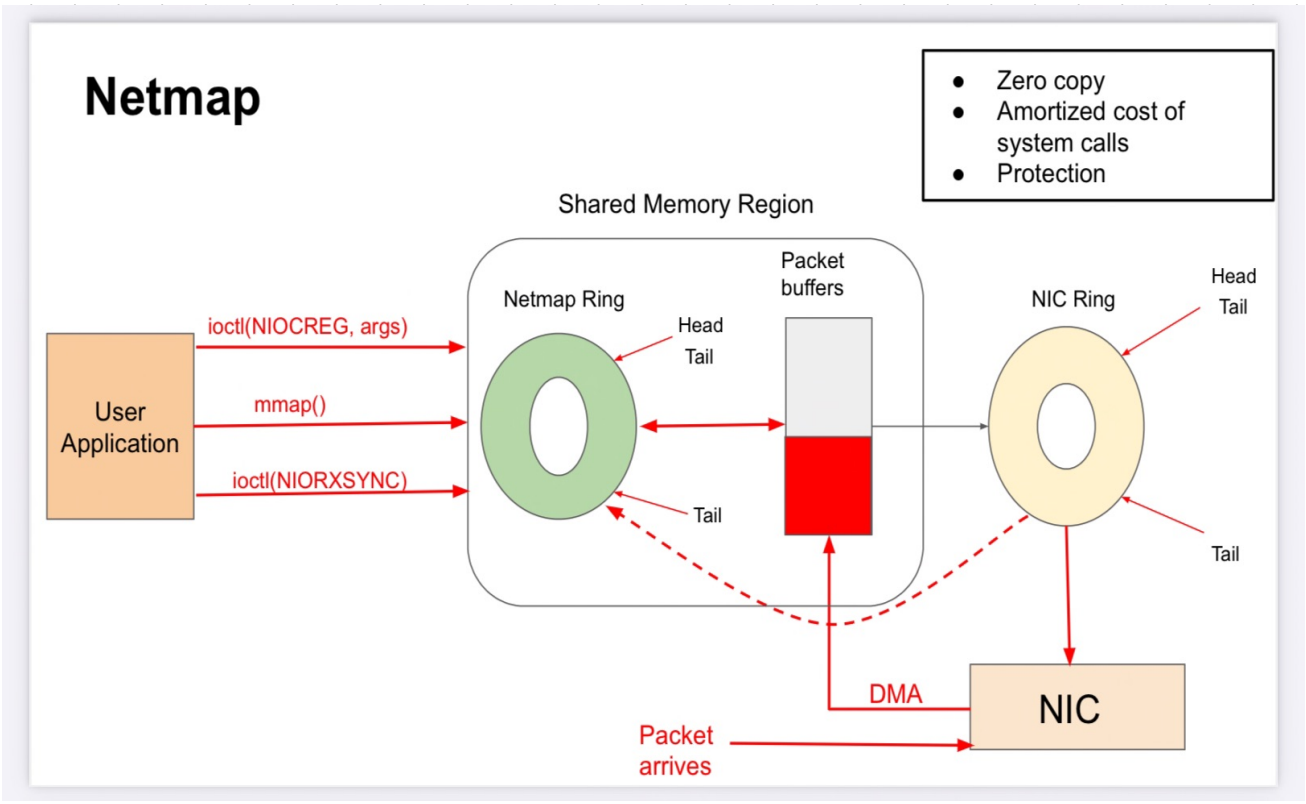


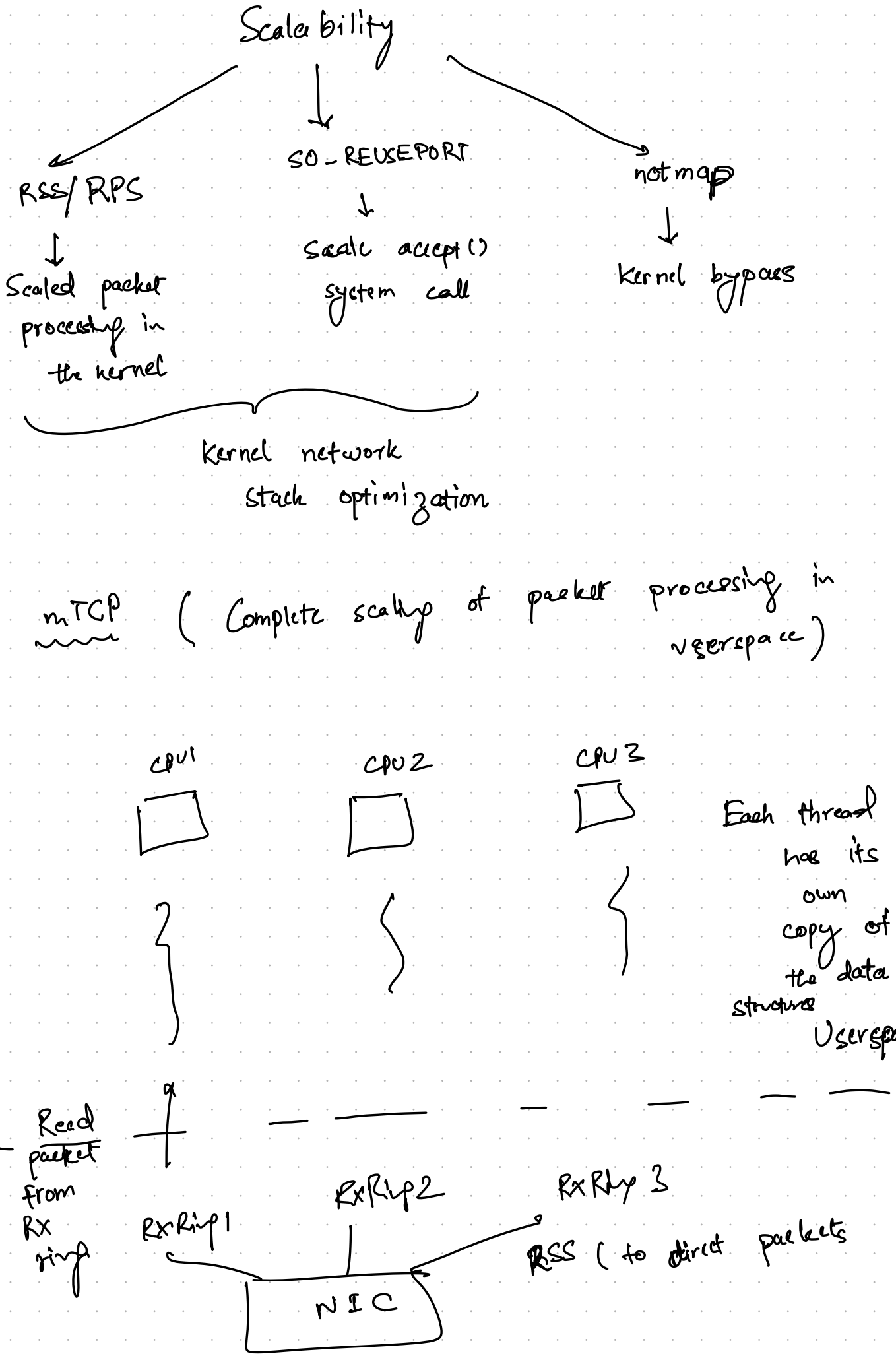
(6) Why kernel Bypass

Why Kernel Bypass?
Connection Locality
VFS overheads
System call overheads
- Memory allocation and deallocation
- and more .....

Linux kernel was not built for being on the data path of 40 Gbps and 100 Gbps network cards.
Difficult to fix the kernel, so bypass it.
The era of Kernel bypass ... DPDK and netmap

(7) Netmap




Scalability
RSS/RPS
SO_REUSEPORT
netmap
Scaled packet processing in the kernel
Scale accept() system call
Kernel bypass
Kernel network stack optimization
mTCP (Complete scalup of packet processing in userspace)
CPU1
CPU2
CPU3
Each thread has its own copy of the data structures
Userspace
Read packet from RX ring
RxRing1
RxRing2
RxRing 3
RSS (to direct packets
NIC