43/44 網路性能最佳化的幾個思路

SRE Round4 讀書會

HungWei Chiu

網路性能指標

- 先搞清楚你的應用類型，需要什麼指標
- Latency
- Throughput (Bit/Byte per second)
- PPS (Packet per second)
- CPS (Connection per second)
- TPS (Transactions per seconds)
- Maximum concurrent connections.

網路性能指標 - 流量工具

性能指標	工具
BPS/PPS	iperf3,pktgen
Latency	ping,hping3
Application	wrk,ab,jmeter, dnsperf

網路性能指標 - 觀測工具

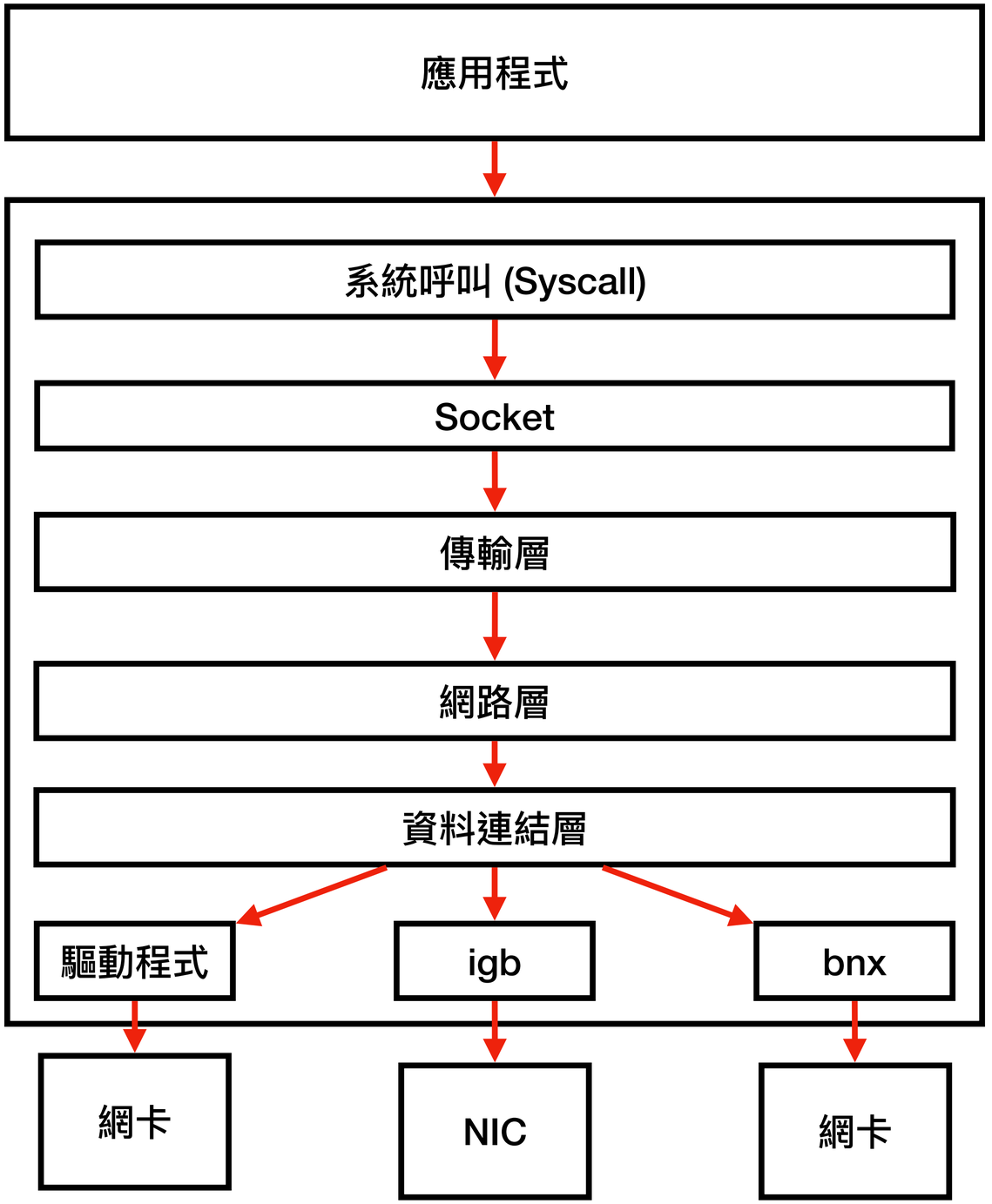
性能指標	工具
BPS	iftop/sar/nethogs
PPS	sar,/proc/net/dev
連接數	netstat,ss

網路性能指標 - 除錯工具

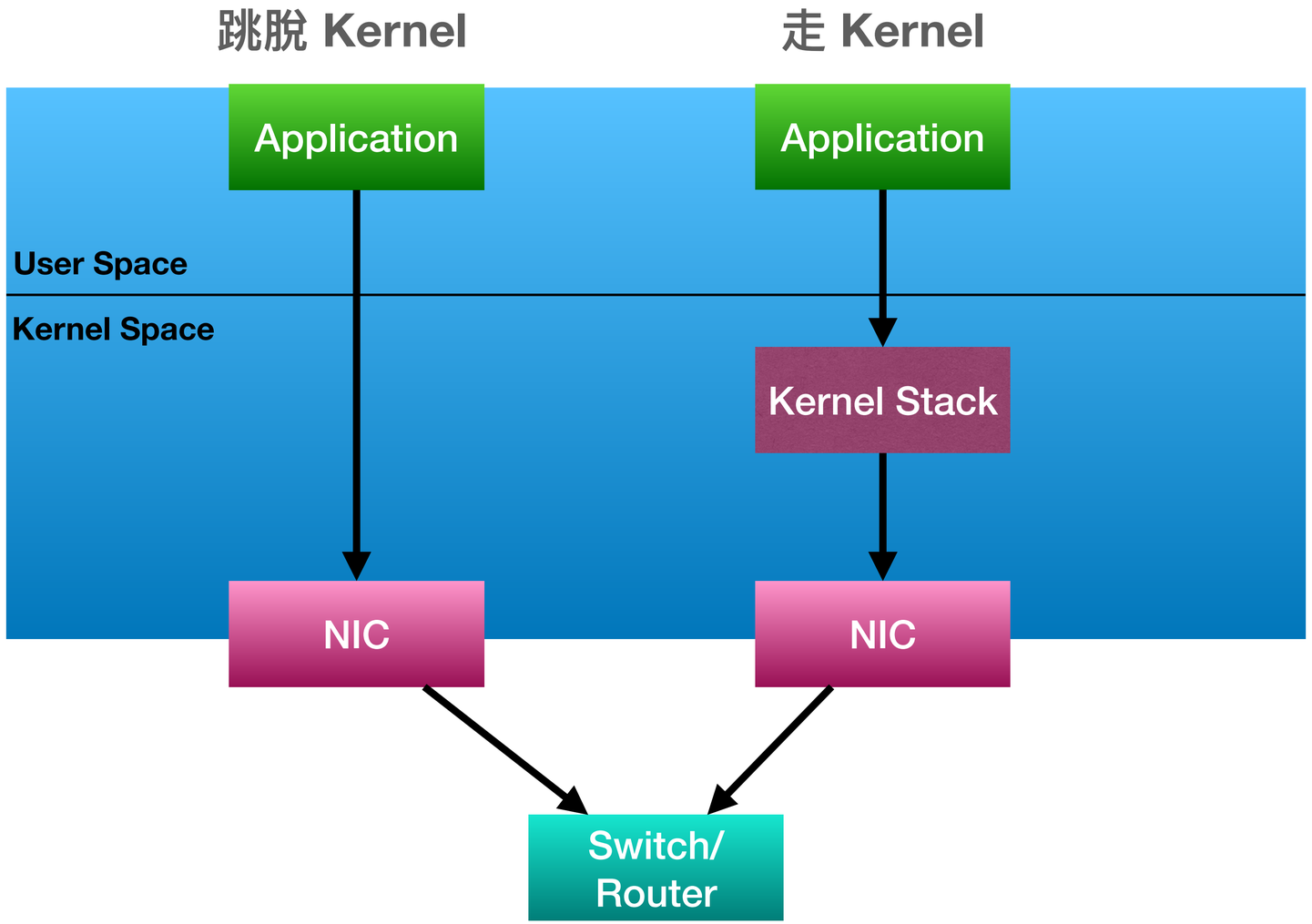
性能指標	工具
連接追蹤數	conntrack
DNS	dig, nslookup
連接數	netstat,ss
轉發規則	route, brctl, ip
擷取封包	tcpdump, wireshark
網卡控制	ethtool
防火牆	Iptables,ufw,ipvsadm
追蹤 kernel 行為	bcc, systemtap
觀察 MTU	tracepath
通暢與否	mtr, deadman,traceroute

最佳化層級

- 網路模型
- 應用程式
- Socket
- 傳輸層
- 網路層
- 資料連結層



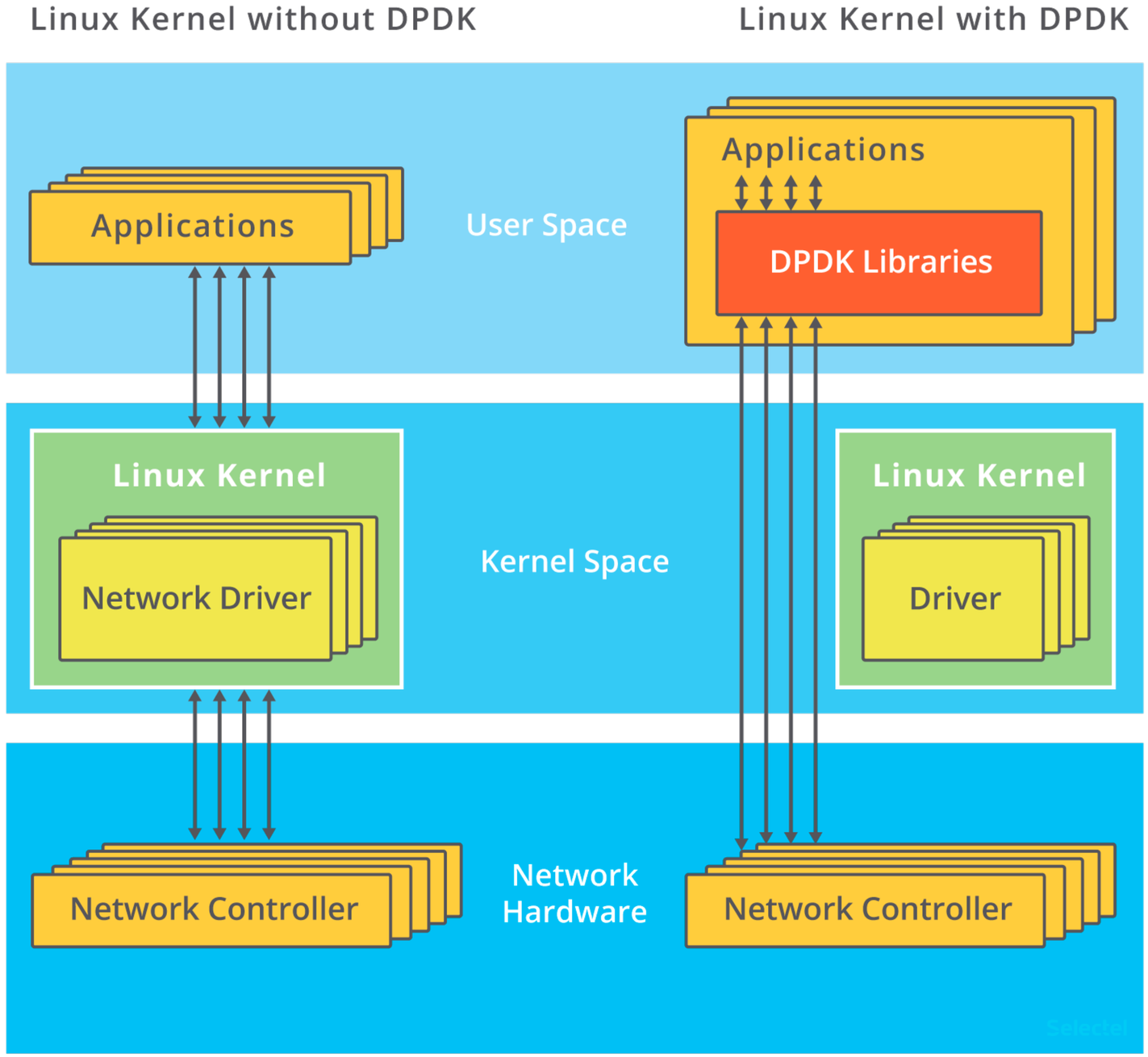
網路模型


跳脫 Kernel
走 Kernel
Application
Application
User Space
Kernel Space
Kernel Stack
NIC
NIC
Switch/
Router

網路模型 (跳脫 Kernel)

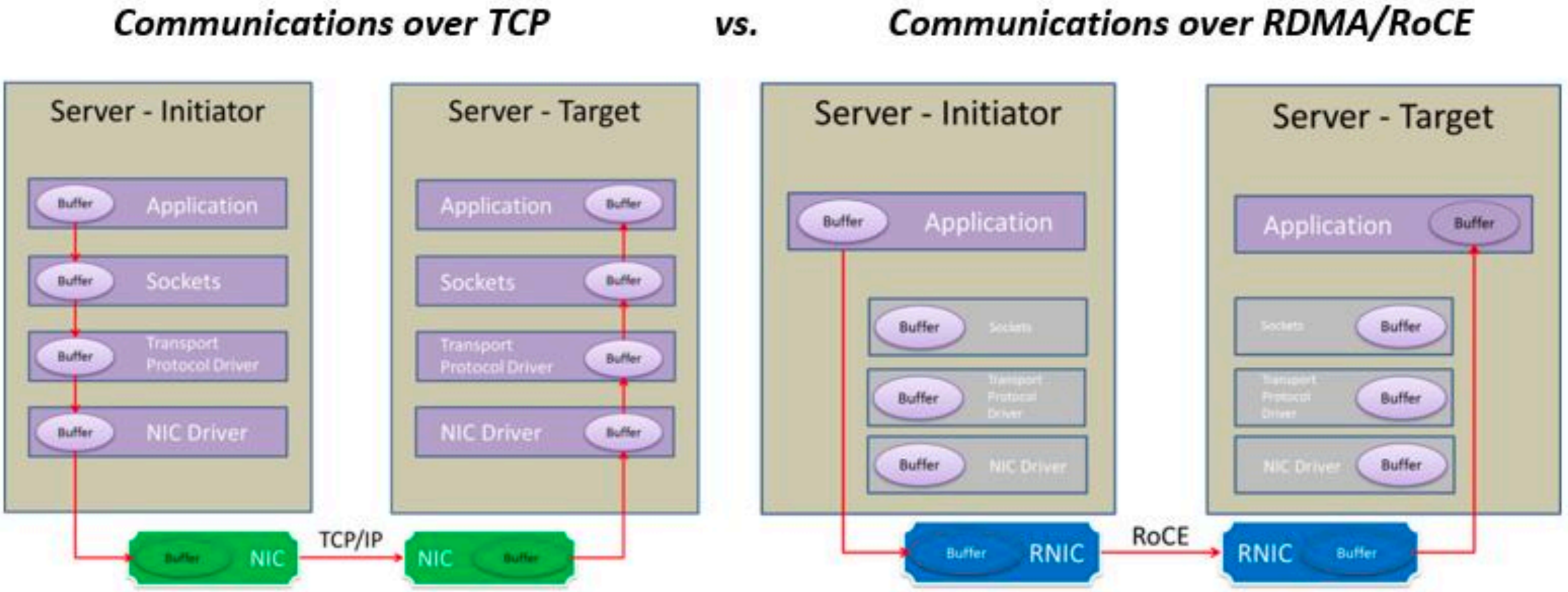
- DPDK
- RDMA
- XDP

DPDK



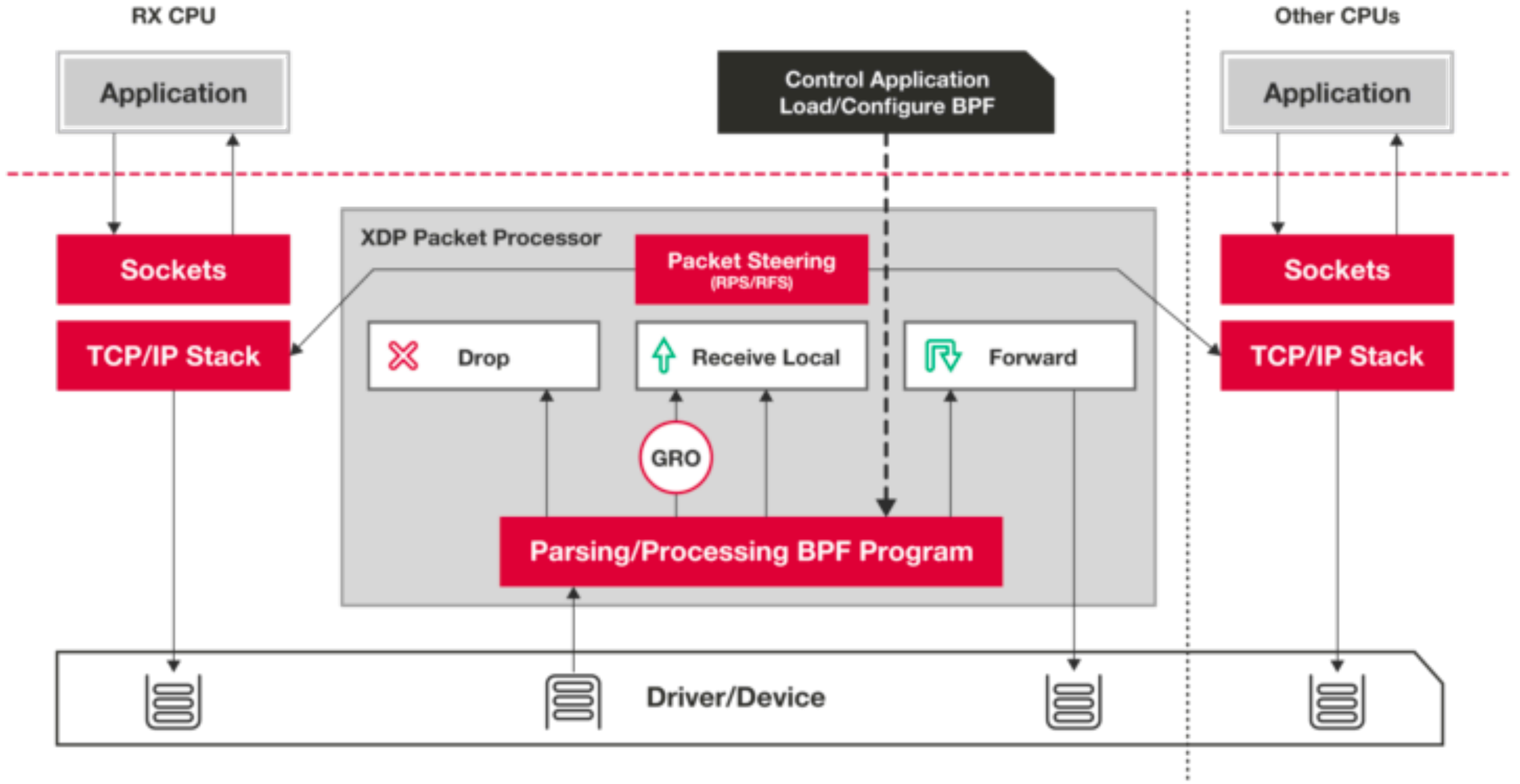
https://blog.selectel.com/introduction-dpdk-architecture-principles/

RDMA



https://blog.mellanox.com/2018/06/double-your-network-file-system-performance-rdma-networking/

XDP

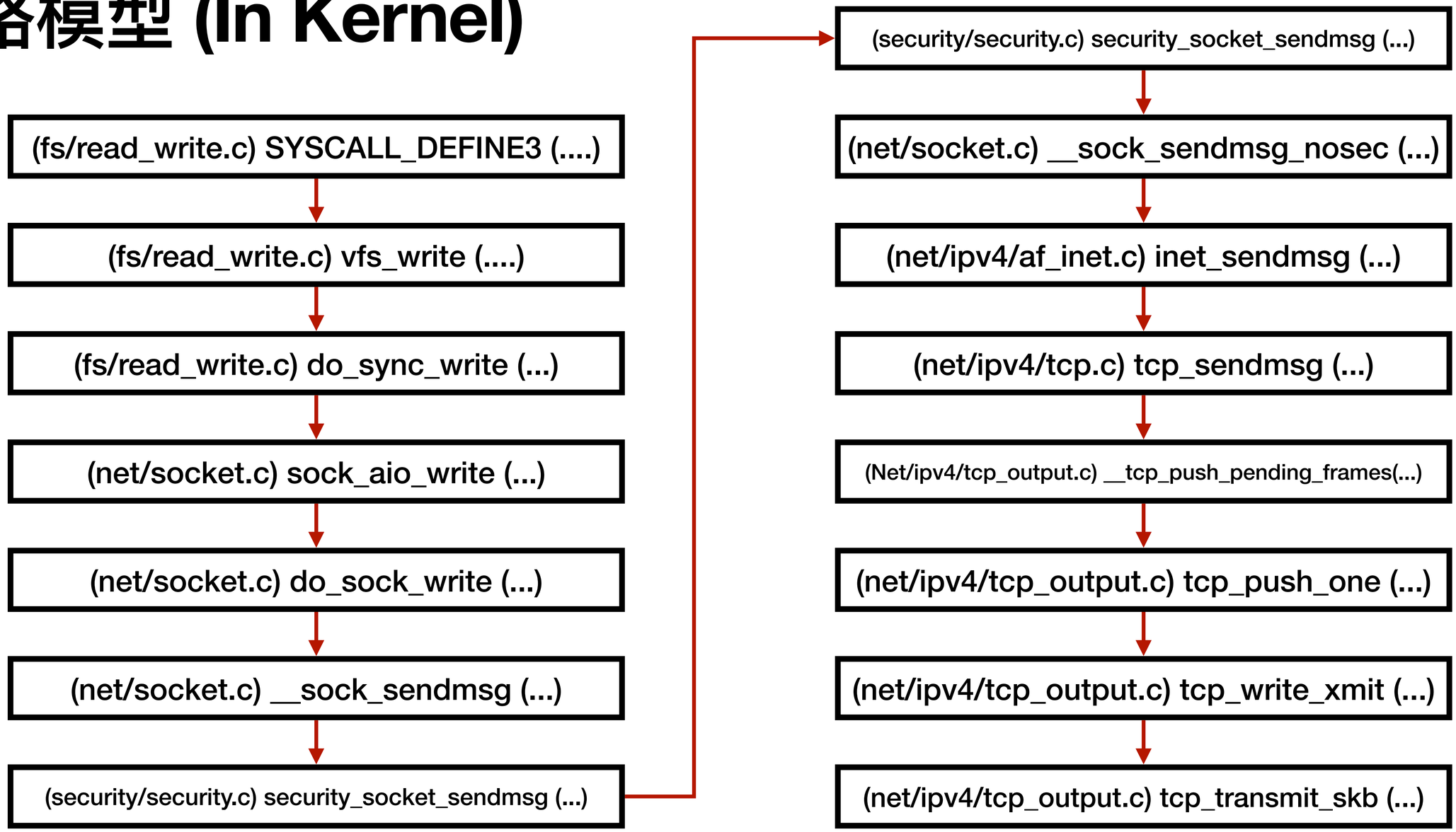


https://www.iovisor.org/technology/xdp

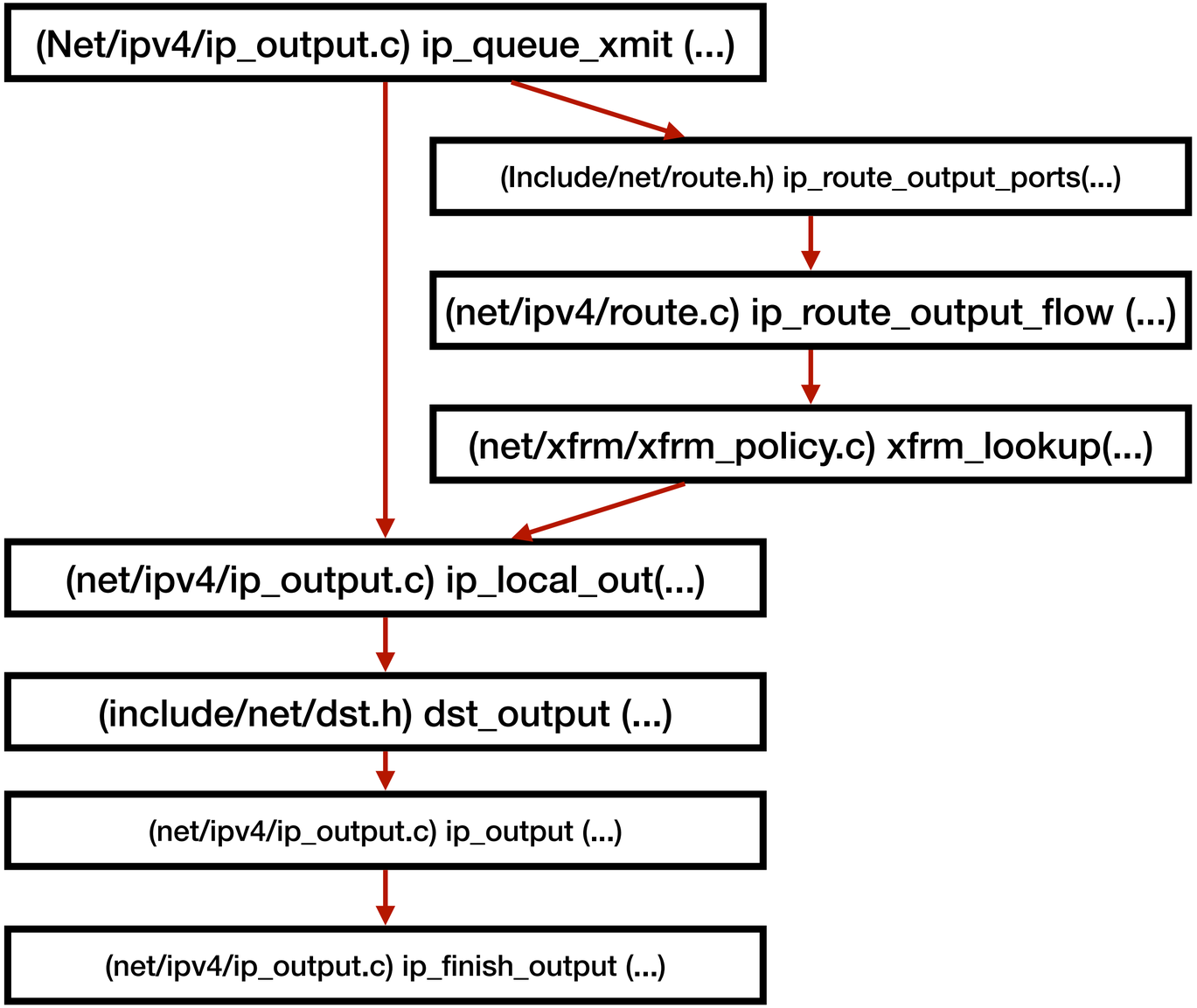
Ceph Example

- Ceph (Distributed Storage Solution)
- https://github.com/ceph/ceph/tree/master/src/msg/async
- DPDK/RDMA
- Epoll/Kqueue/Select

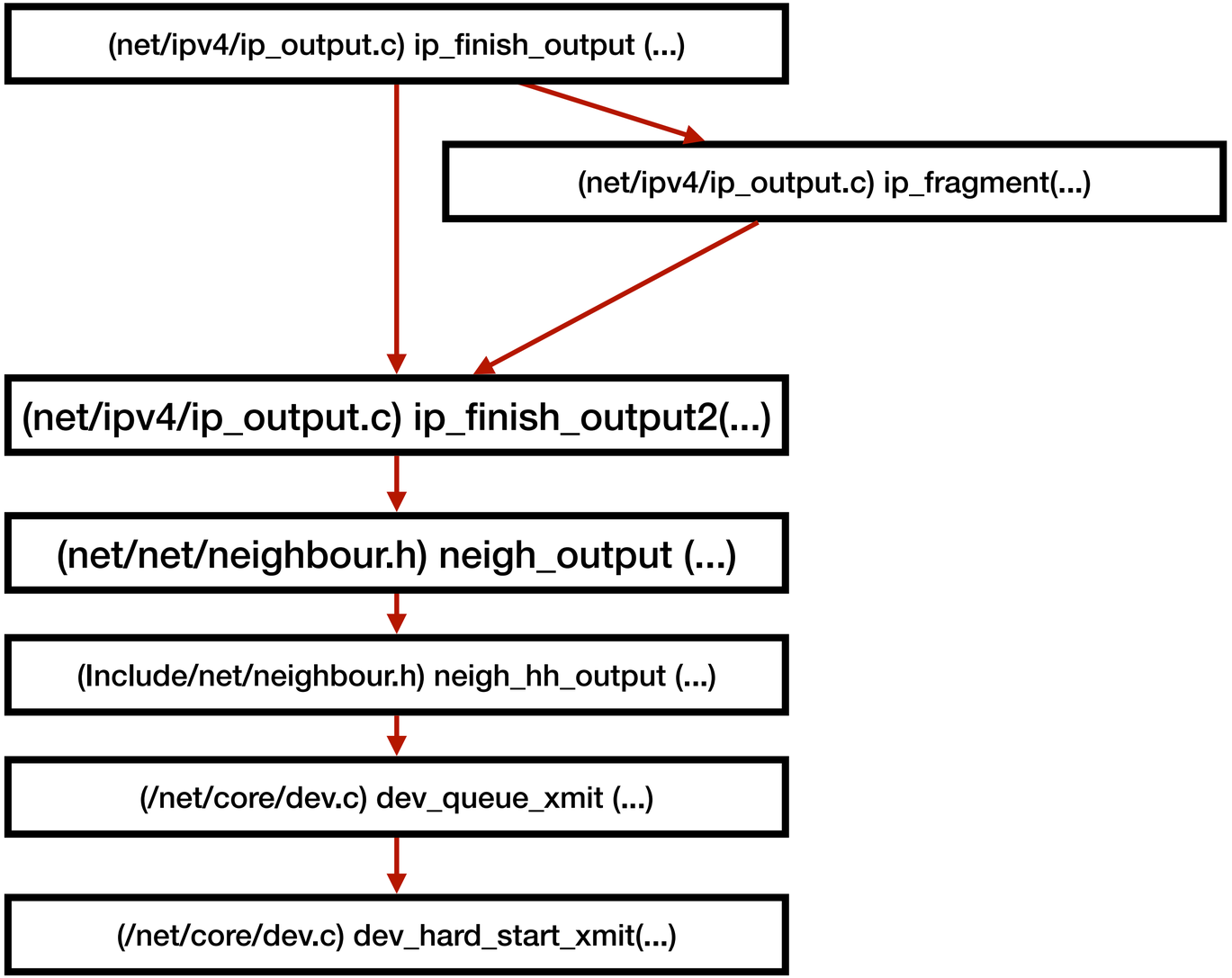
網路模型 (In Kernel)



網路模型 (In Kernel)

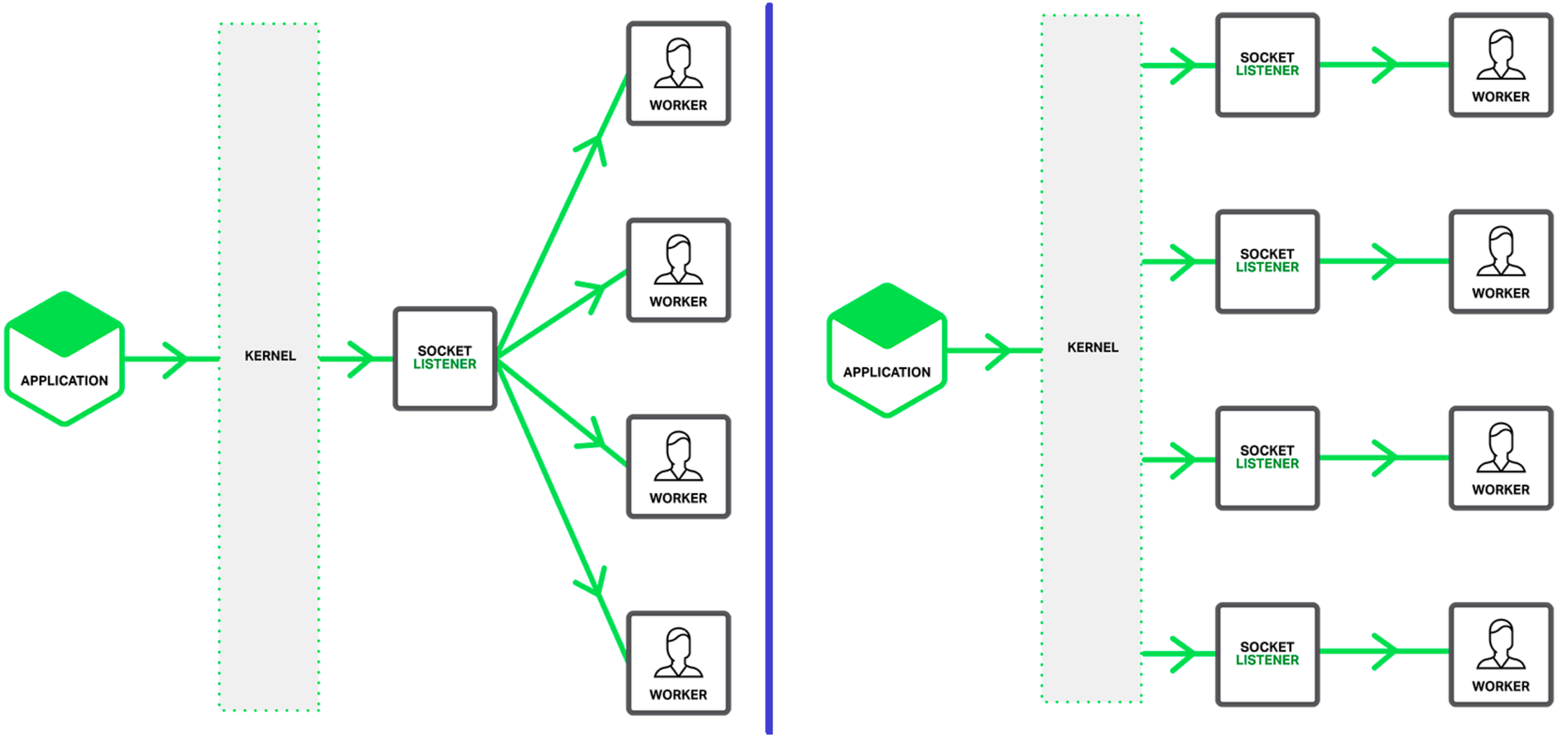


網路模型 (In Kernel)



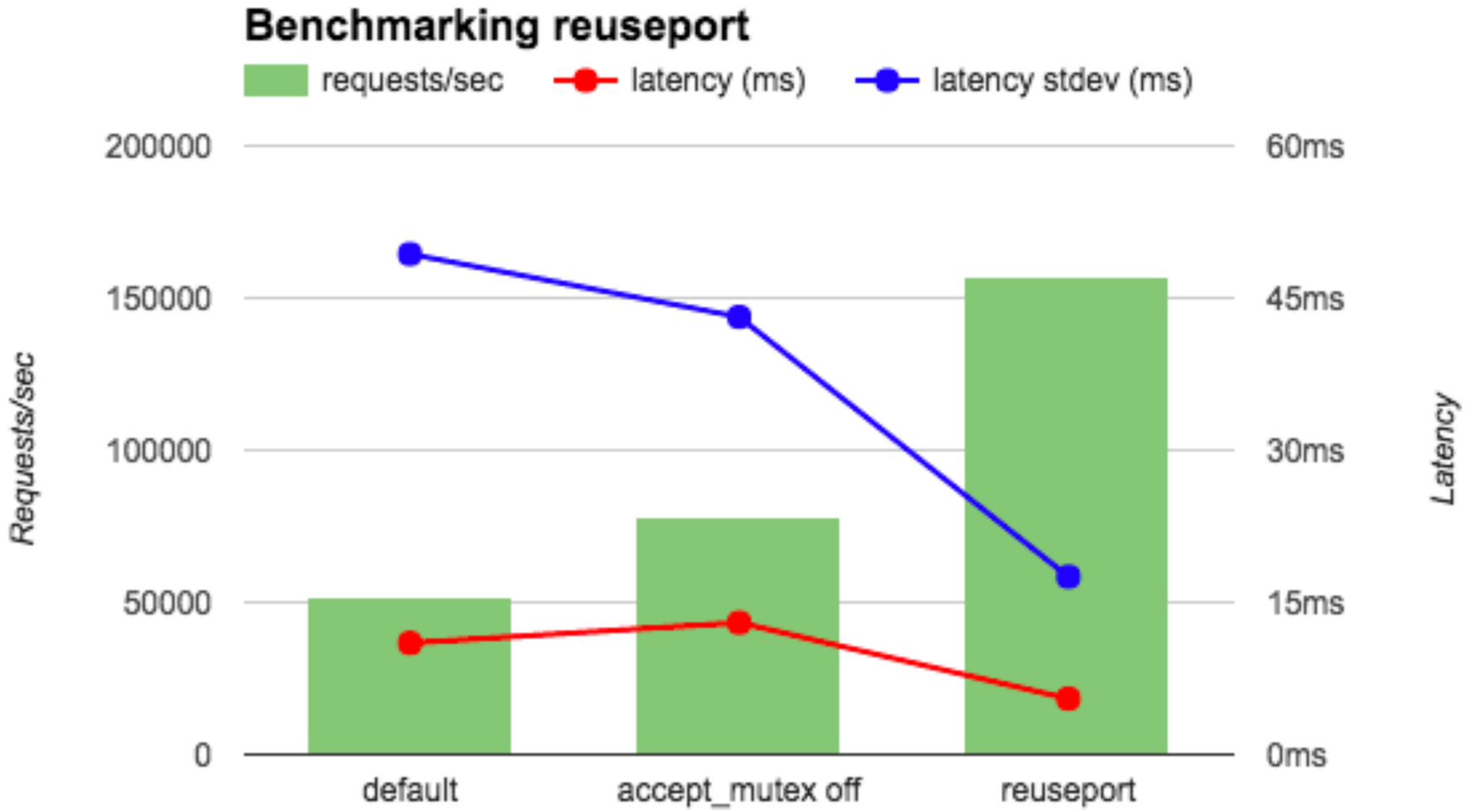
最佳化層級 - 應用程式

- I/O 模型
  - epoll, select ...etc
  - AIO
- Process 的工作模型
  - 主 Process 配上眾多 worker
  - 多 Process
    - SO_REQUESTPORT



https://www.nginx.com/blog/socket-sharding-nginx-release-1-9-1/

Wrk benchmark



https://www.nginx.com/blog/socket-sharding-nginx-release-1-9-1/

最佳化層級 - 應用程式

- Long-lived connection
  - 減少 TCP 建立成本
    - Load-Balancer 則是額外議題，譬如 gRPC，本身傳輸已經不是 L3/L4 的行為了
- Protocol Buffer
  - 序列化資料，壓縮傳輸量
- 各種 Cache

最佳化層級 - Socket

- 直接看 Kernel 內文件，由 Intel 針對 ixgb 系列網卡 driver 所介紹的效能提昇設定
- https://lxr.missinglinkelectronics.com/linux/Documentation/networking/device_drivers/intel/ixgb.rst#L272
- With the 10 Gigabit server adapters, the default Linux configuration will very likely limit the total available throughput artificially.

最佳化層級 - BBR

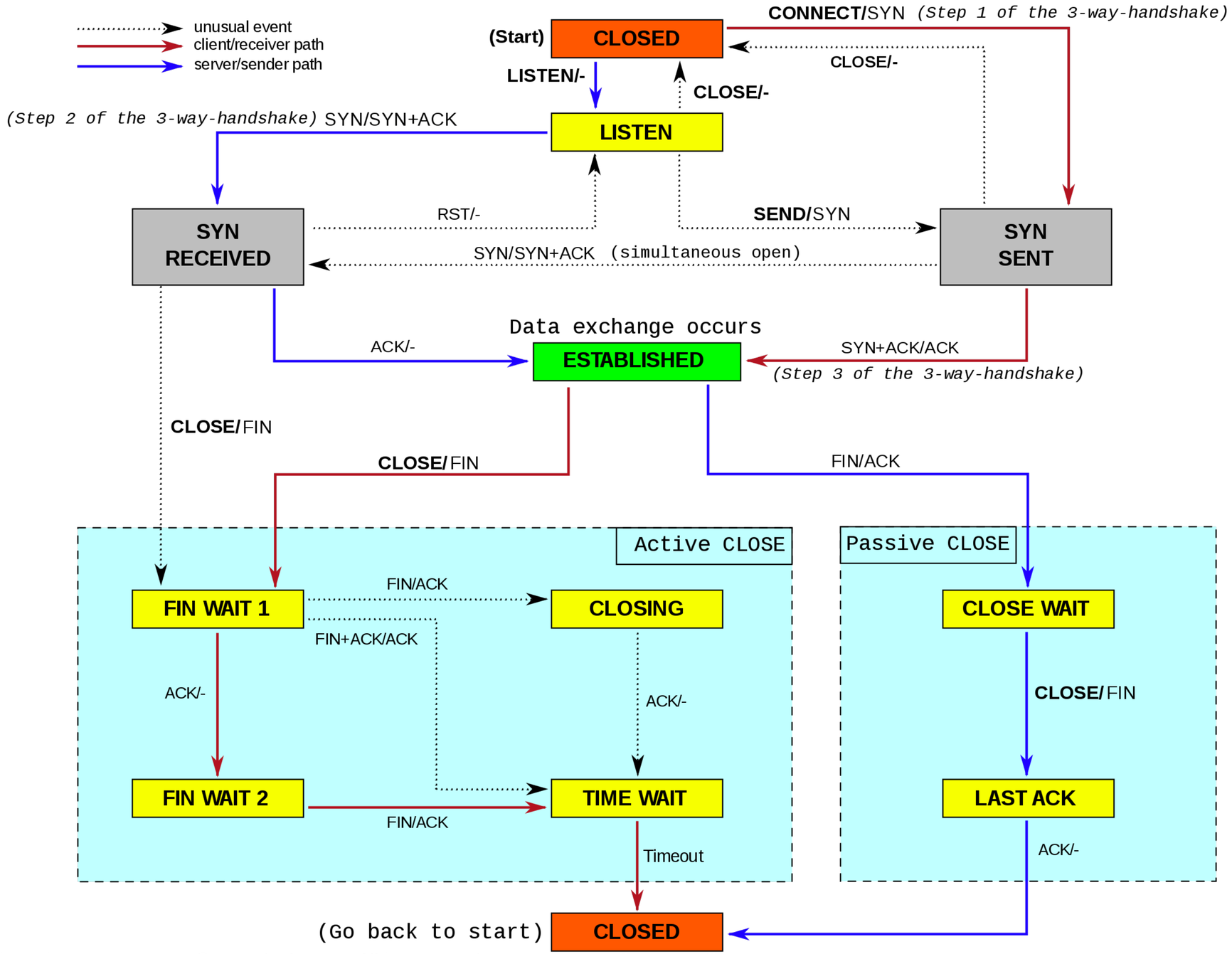
- Congestion-Based Congestion Control
- Released by Google (2016)
- Better throughput and lower latency
  - Compare to CUBIC (default algorithm before 4.19)
- 參考: https://shorturl.at/jxV35

最佳化層級 - 傳輸層

- TCP, UDP 針對協議最佳化
- 除了這些之外還有不同的協議，譬如 MPTCP (Multipath TCP)
  - 提昇整體頻寬使用量以及加強冗於性 (多路徑可用)
  - Wiki: https://en.wikipedia.org/wiki/Multipath_TCP#Use_cases
  - Apple uses MPTCP to support Siri on iPhone.
    - 5x reduction of network failures

最佳化層級 - TCP

- 三方交握建立連線
- TCP 的所有連線都會發送請求，並且等待回應
- Client 的目的是連接對方，會有相關的狀態變化
- Server 的目的是等待他人連接，也會有相關變化
- 所有的連線都是基於 Socket 為概念去看



https://en.wikipedia.org/wiki/Transmission_Control_Protocol#/media/File:Tcp_state_diagram_fixed_new.svg

最佳化層級 - TCP

- 減少 TIME_WAIT 狀態霸佔資源
  - 減低 TIMOUET，趕緊釋放
  - 增大容量，讓系統可以同時支持更多 TIME_WAIT 狀態
- TIME_WAIT 的連接埠可重複使用 TCP_TW_REUSE
- 修改 ip_local_port_range 來增加更多 port 給 server 使用 (source port)
  - 大部分預設是 32768 ~ 60999
  - Kubernetes Node Port (30000 ~ 32767)
  - 要是衝突可能會讓你的 NodePort 開不起來，因為 port 被人佔用走，要設定請小心

最佳化層級 - TCP

- KeepAlive, 確認當前連線還是否存在
  - 定期發送封包給對方並且等待回應，因為 TCP 是 stream(串流)連線，所以發送一個沒有資料的 probe 封包其實對應用程式沒有影響
    - 額外的封包流量
- 發送(keepalive packet), 告知對方我活，探測 (keepalive probe), 確認對方還活
- 縮短發送間隔時間 (keepalive_time)
- 縮短探測時間 (keepalive_intvl)
  - Default: 9 probe
- 減少通知標準 (keepalive_probe)
  - 多少個沒有收到回應的 probe
- Example: 9(probe), 30sec(intvl) --> 9*30 = 270 = 4 mins.

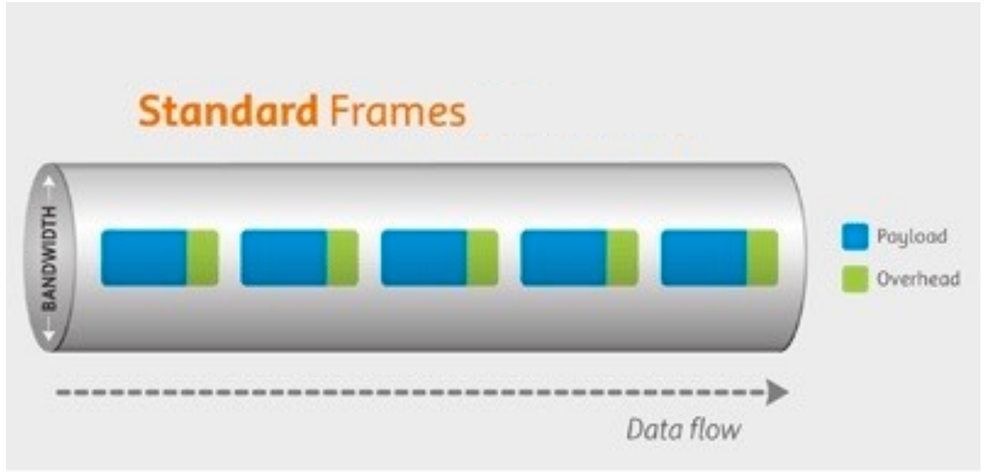
最佳化層級 - TCP

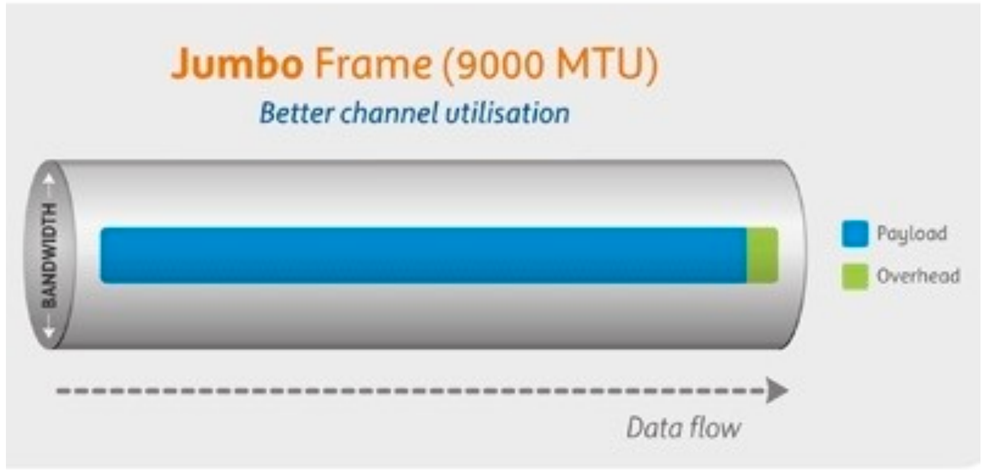
- SYN Flood
  - 一種攻擊模式，利用三方交握
  - Client ----> Server (SYN) (攻擊者送大量的SYN 並且偽造 source IP)
  - Server ---> Client (SYN-ACK) (Server 期望連線，因此在等對方的 ACK，遲遲等不到，就浪費很多資源)
- max_syn_backlog
    - 最大半連接的最大數量(有想要建立連線，但是還沒收到ACK)
- synack_retries
    - Kernel 嘗試多少次後就會放棄這條連線

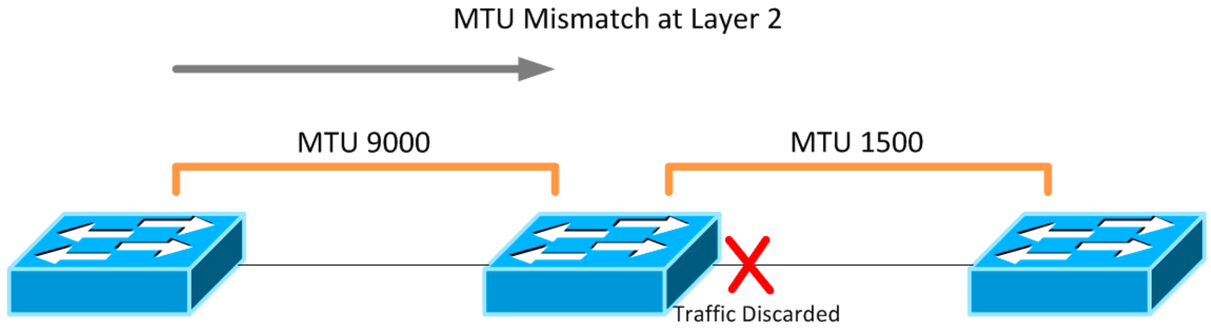
最佳化層級 - UDP

- 無狀態連線，沒有對方回應，相對於 TCP 的架構簡單許多
- 緩衝區大小
- 調整 MTU
  - 注意！路徑上所有節點都要有對應的 MTU
  - MTU 是 Layer 2 的東西，沒有切片(fragmentation)功能, 超過就丟棄
  - PMTUD (Path MTU Discovery) 算法，幫你根據路徑自動調整
  - Cloudflare: https://blog.cloudflare.com/path-mtu-discovery-in-practice/

MTU







https://www.packetflow.co.uk/mtu-jumbo-mss/

https://oswalt.dev/2013/03/jumbo-frames-beyond-the-broadcast-domain/

最佳化層級 - 網路層

- 功能:
  - 網路封裝，轉發，
- 最佳化思路
  - 路由加速，IP 分段
  - ip_forward =1 (單純開啟，沒啥最佳化，完全是應用導向)
  - ip_default_ttl (決定預設 TimeToLive 的值，決定封包被丟棄前可以走多少節點)
  - rp_filter (反向檢查封包來源，不合理的就先丟棄)

最佳化層級 - 網路層

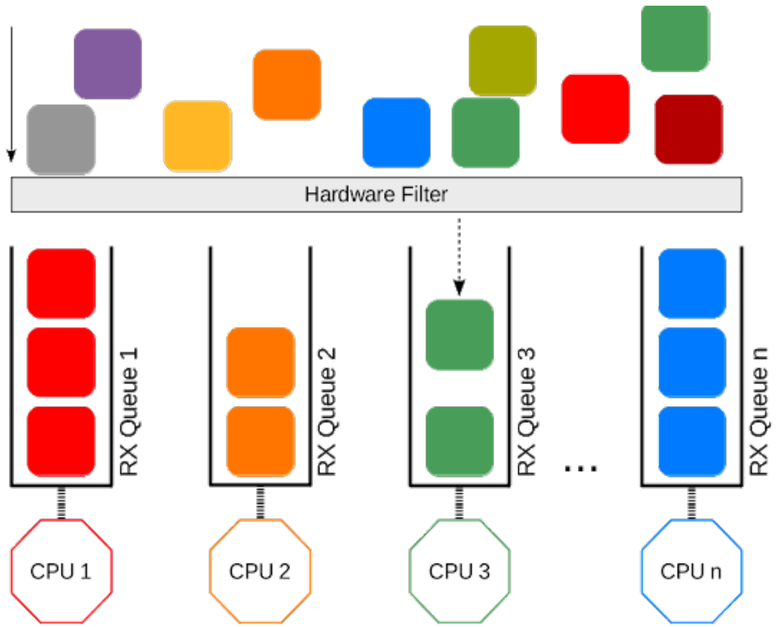
- MTU
  - 請注意任何封裝技術加入的額外標頭檔案
  - VXLAN/GRE/NVGRE/GENEVE

最佳化層級 - 資料連結層

- 網卡收到封包厚，會透過軟中斷去觸發接收封包
  - 中斷調度到不同 CPU
    - smp_affinity, irqbalance
    - 減少 CPU Cache Missed 的次數，讓 CPU 與應用程式用同一顆 CPU 處理
  - RSS/RPS/RFS
    - RSS (硬體處理)
    - RPS/RFS (軟體處理)

最佳化層級 - 資料連結層

- RSS (Receive Side Scaling)
  - 硬體實作，根據封包標頭內容來進行雜湊換算，送到相同的 RX Queue
  - 關閉 irqbalance，自行將每個 CPU 與 Queue 給對應
    - /proc/irq<IRQ_NUMBER>/smp_affinity



https://garycplin.blogspot.com/2017/06/linux-network-scaling-receives-packets.html

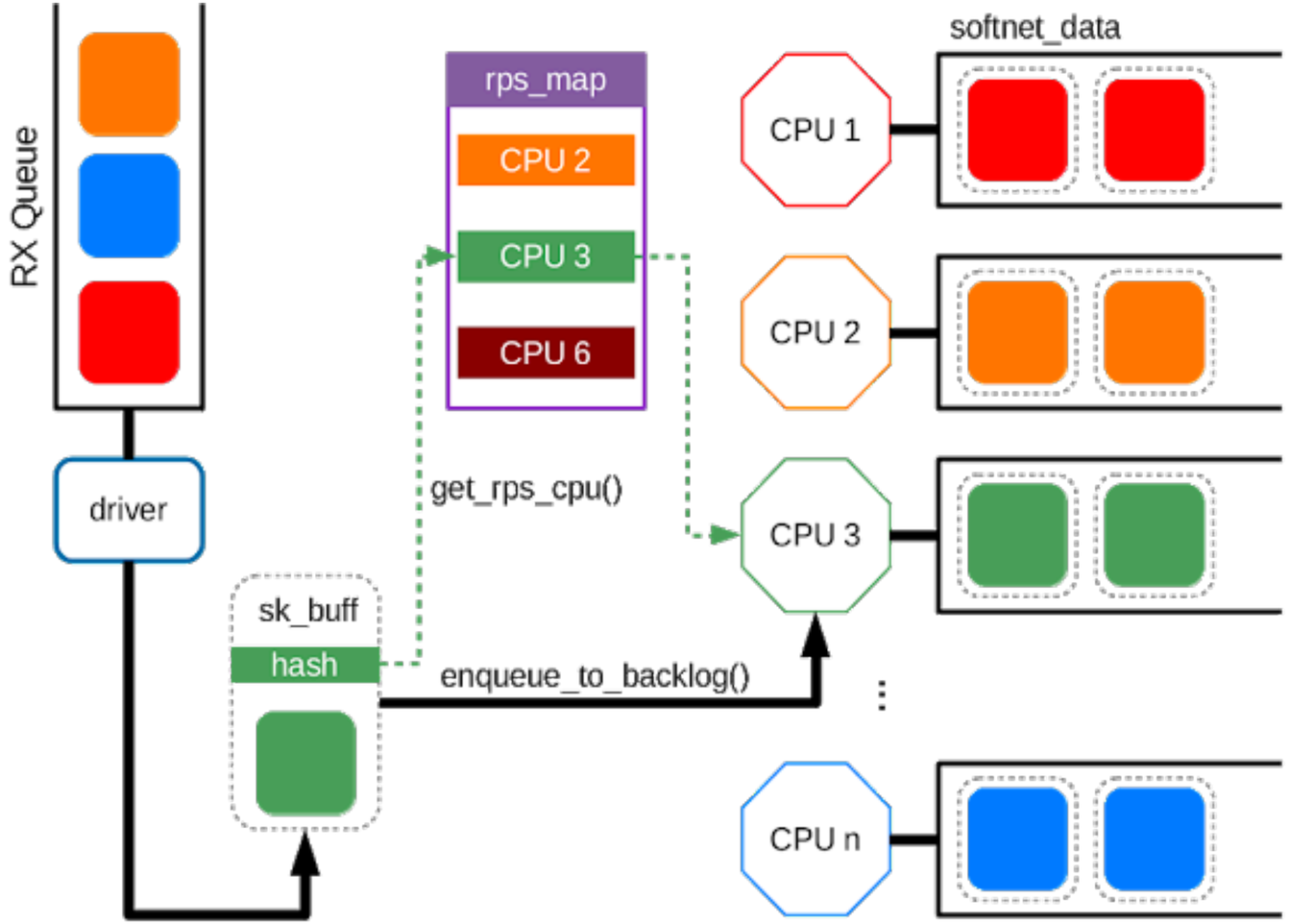
最佳化層級 - 資料連結層

- RPS (Receive Packet Steering)
  - Kernel 內實現的軟體功能
    - Layer4 Hash (source IP, source Port, dest IP, dest Port)
    - 同連線中的所有封包會有相同的 Hash, 應該要同個 CPU 處理效能會更好
    - /sys/class/net/eth0/queues/rx-xx/rps-cpus (bitmask)

https://garycplin.blogspot.com/2017/06/linux-network-scaling-receives-packets.html

最佳化層級 - 資料連結層

- RPS (Receive Packet Steering)



https://garycplin.blogspot.com/2017/06/linux-network-scaling-receives-packets.html

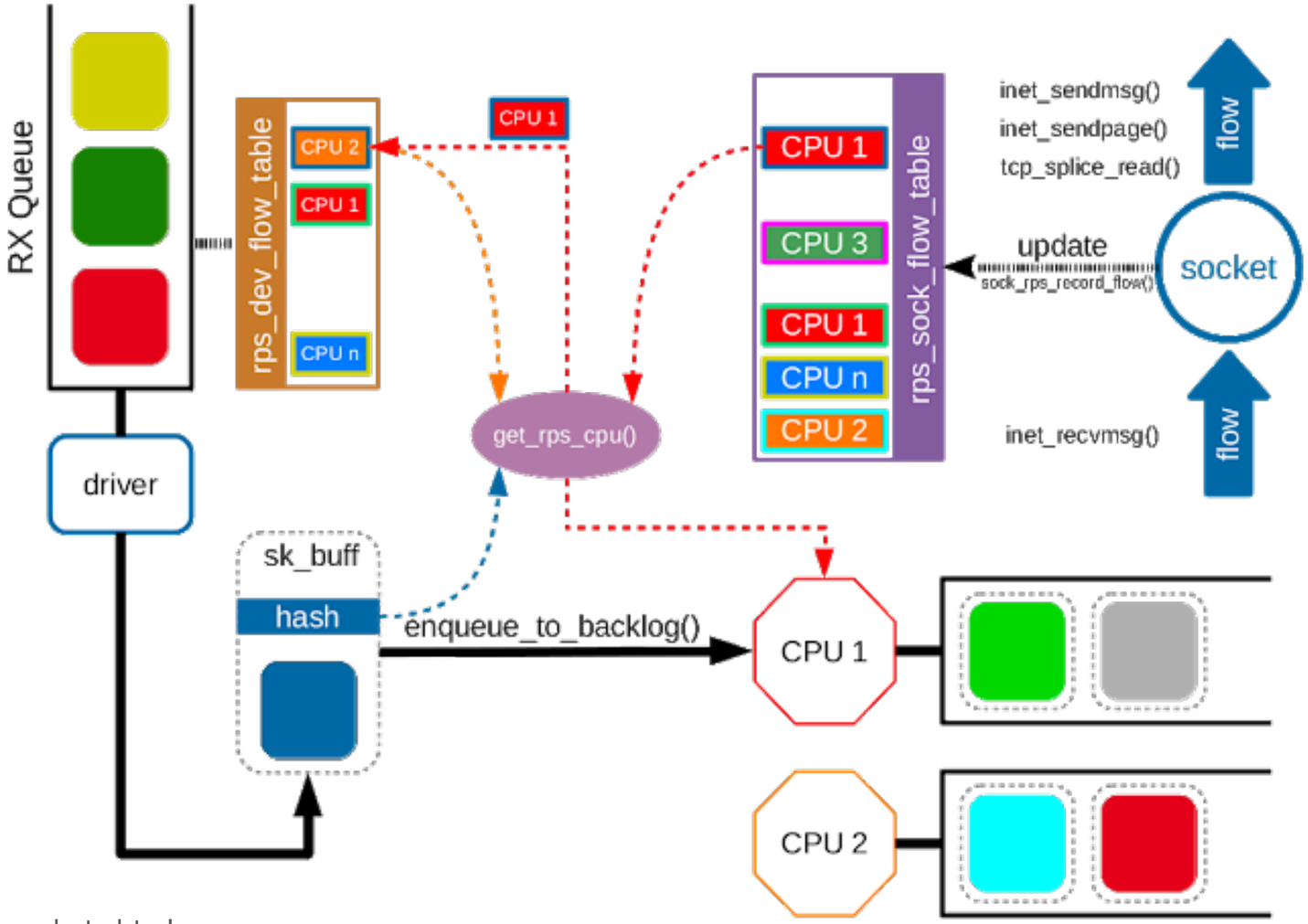
最佳化層級 - 資料連結層

- RFS (Receive Flow Steering)
  - RPS 加上應用程式考慮，避免應用程式跟Queue屬於不同CPU，導致 Cache Missed
    - 提昇 Locality,

https://garycplin.blogspot.com/2017/06/linux-network-scaling-receives-packets.html

最佳化層級 - 資料連結層

- RFS (Receive Flow Steering)



https://garycplin.blogspot.com/2017/06/linux-network-scaling-receives-packets.html

最佳化層級 - 資料連結層

- Hardware Offloading (ethtook -k)
  - 送封包
    - TSO/UFO/GSO
    - 讓你軟體方面可以創造很大的封包，kernel 只要處理一次，剩下讓硬體幫你拆解
  - 收封包
    - LRO: 聚合 TCP 封包，將 DATA 合併後往上送
    - GRO: 強力 LRO，支援 TCP/UDP
  - Tunneling

Q&A