IX

What is the trade-off between throughput and latency in a TCP/IP stack?
- Interrupts, handling one packet at a time, leads to low latency
  - Under load, uses more CPU time/packet, lower throughput (+ higher latency)
- Polling+batching packets is much better under load
  - Save on interrupts, protection-domain crossings (in/out of kernel)
  - Possibly better icache behavior
  - But waiting for more packets in batch adds latency
  - Dedicating a hyperthread to TCP/IP (mTCP) also increases latency
    - Need to wait for application threads to notice and pick up data

What's kernel bypass networking?
- Give user-level code direct access to NIC (e.g., DPDK)
- What about other users of NIC?
  - SR-IOV allows hardware virtualization
- What if user code asks NIC to DMA over important kernel data structures?
  - Can use VT-d (intel) or IOMMU (AMD) to virtualize DMA addresses
- So is kernel bypass + SR-IOV + VT-d the ideal solution?
  - Still need a TCP stack (or more likely stuck with UDP for simplicity)
  - User code can still do bad things to the network
    - E.g., violate TCP flow control, forge IP address ,send bogus ARPs
- What about RDMA? Provides one host access to memory on another
  - Fast, but requires specialized NIC. Protection issues even worse.

What's the issue with resource efficiency?
- Can partly address network performance by over-allocating resources
- Much better would be to use extra resources for other applications
  - Or at least put cores into low-power states to save energy
- What are examples of wasted resources?
  - Dedicated core/hyperthread for TCP/IP processing not fully utilized
  - Gratuitous data copies/cache misses waste CPU time, memory bandwidth

What's a "middlebox"?
- Sits on the network path but isn't an endpoint for communication
- Examples: NAT, firewall, intrusion detecters

Enterprise middleboxes keep up with fast networks--how?
- Process each packet to completion (no batching, extra queuing latency)
- Use *flow-consistent hashing*--what's this?
  - Hardware has multiple receive queues
  - Chooses packet by hashing flow identifiers (e.g., source+destination IP)
  - Means all packets from a given flow are processed by a single core
    - Minimizes synchronization between cores
    - E.g., NAT could keep state in per-core hash table

What's the promise of IX? "Resolve 4-way trade-off"
- Low latency, high throughput, protection, resource efficiency

What is high-level IX architecture (Figure 1a, p. 54)?
- Dataplane OS runs in Dune CPL0, while app runs in Dune CPL3
  - One dataplane OS & address space per application
  - Elastic threads process network packets on dedicated hyperthreads/cores
    - TCP/IP processing at CPL0, then hand off to application at CPL3
  - Background threads can timeslice other hyperthreads/cores
  - Passes non-network system calls through to Linux
- Control plane IXCP is ordinary linux process (outside Dune)
  - Allocates cores to different processes using linux mechanisms (cgroups)
  - Reallocation can happen at very coarse granularity
- Linux used for everything else
  - Bootstrap, hardware initialization, management (just ssh in)

What is a packet flow (Figure 1b)?
- Poll NIC, get small adaptively-sized batch of packets
  - Don't wait, just get packets accumulated since last poll

```
    Don't exceed CPU cache size, either
  Do TCP/IP processing (e.g., generate ACKs, etc.)
  Send batch of packets up to application in CPL3
  Application responds, potentially generates packets
  Transition back to CPL0 to send packets & run timer wheel

What is system call API (Table 1, p. 55)?
  Are the dataplane synchronous system calls cross-CPL calls like in linux?
    Technically only one networking system call, namely "run_io"
    "calls" in table are really messages in memory shared by CPL0/CPL3
    No return value, instead "event conditions" written to mem by kernel
  What is lifecycle of a TCP connection, say for simple HTTP GET?
    Event: knock(handle, src_IP, src_port) - notification of SYN received
    Call: accept(handle, cookie)
    Event: recv(cookie, mbuf_ptr, mbuf_len)
    Calls: recv_done(handle, bytes_acked)
           sendv(handle, scatter_gather_array)
    Event: sent(cookie, bytes_sent, window_size) - app can free memory
    Call: close(handle)
  What's an mbuf?  Memory for received packets

What are cookies and handles?
  Handle: sort of like a file descriptor, but non-contiguous
  Cookie: value for handle chosen by application--why?
    Kernel may have to look up handle in (protected) hash table
    Don't want app also to have to maintain a hash table
    Cookie is chosen by app, so can be pointer to per-connection struct
      Note: okay for application to trust cookies from kernel
      Conversely, kernel still needs to sanity-check handles

Does IX obey the scalable commutativity rule?
  Yes because handles can be anything kernel wants
    (e.g., partition handle space by hyperthread/core)
  But what if packet comes in on wrong CPU for handle?
    That's what flow-consistent hashing is for
  What about outgoing connections? How does response come back on same CPU?
    Can't reverse Toplitz hash (p. 56)
    Just try multiple port numbers to find one that hashes to local CPU
  Are there no global data structures requiring synchronization?
    ARP table protected by RCU

What is incast (p. 54)?
  Large number of machines all send to same machine, overflowing switch buffer
  Dropped packets lead to disastrously slow retransmissions (e.g., 200 msec)
  Problem can be exacerbated by synchronized TCP timers
    E.g., retransmit batch of requests every 200 msec
  Retransmission potentially solved by very fine granularity timers
  Ideally future IX could also support paced_sendv for transmission, too

Does IX protect against ill-behaved application code?
  Yes, because all TCP/IP processing happens in protected dataplane kernel
  So app cannot forge IP address, bypass congestion control, etc.
Does IX protect against ill-behaved dataplane kernel code?
  Not at time paper was published--why not?
  Dataplane kernel interacts with NIC, could DMA over linux kernel memory
  Solution?  Make IX use IOMMU/VT-D (possibly with SR-IOV to share NIC)

Why is mbuf memory accessible only in read-only mappings at CPL3?
  mbufs get recycled for more incoming packets after recv_done
  Don't want to have to knock it out of TLB
    One INVLTB instruction per received packet would be expensive
    Plus IX only uses 2MiB pages, so need multiple mbufs/page
  If page not read-only, app could keep modifying mbuf after recv_done
    Means it could overwrite new packets not yet processed
  So?  Since one app per dataplane OS, would app just shoot itself in foot?
```

No, because might overwrite incoming packet before kernel sees it!
E.g., could corrupt ACK values to bypass congestion control

What is Intel's Data Direct I/O (also DCA) technology? (p. 59)
Delivers data directly into CPU cache instead of just DRAM

What evaluation questions should we ask?
Does IX deliver
1. Low latency?
2. high throughput?
3. protection?
4. resource efficiency?
How compatible is it with existing applications? How easy to deploy?