Networking Operating System from Scratch towards High-Performance COTS Network Facilities

Hirochika Asai <panda@jar.jp>
The University of Tokyo
IIJ-II Seminar, Tokyo, Japan
April 9, 2015

Biography

- Hirochika Asai (panda)
  - Professional history
    - 2013: Received Ph.D in Information Science and Technology from the Univestiy of Tokyo
      - "Analysis and Management of the Internet based on Data Flow Profiling"
    - 2013-now: Project Assistant Professor at the University of Tokyo
    - 2014-now: Board member of WIDE Project
  - Research interests
    - Operating system (networking)
      - had been my hobby…
    - Distributed system (especially Internet-wide system)
    - Internet traffic and topology analysis

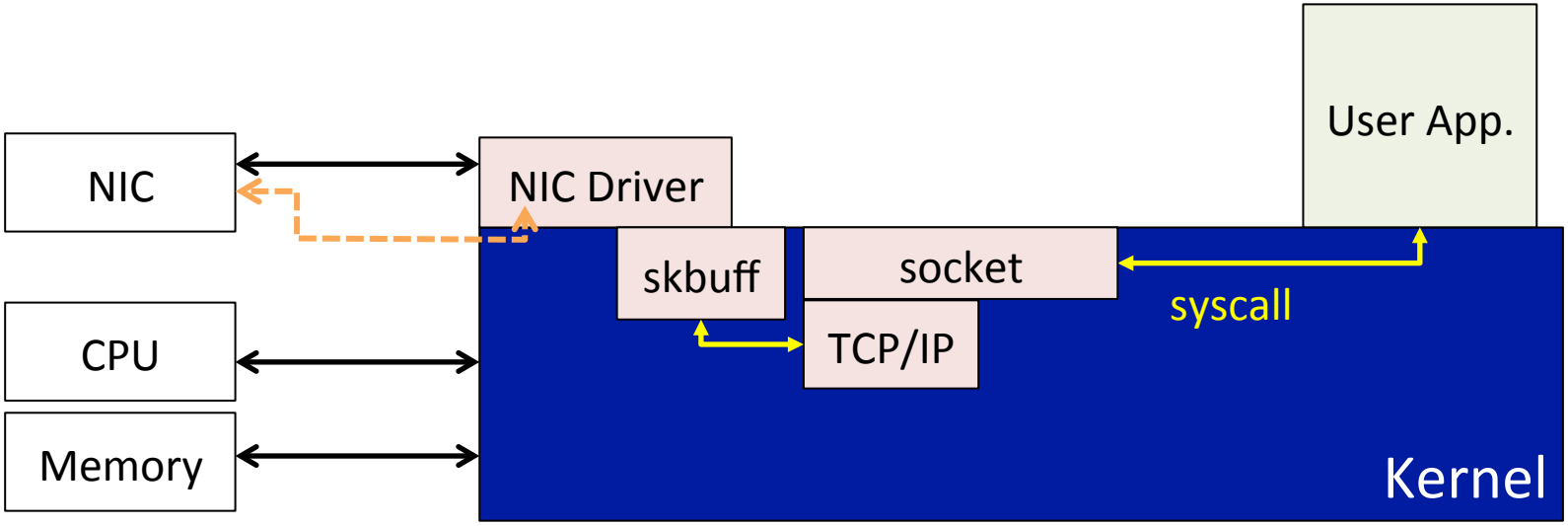
Trends on Network Functionalities by Software on COTS hardware

- SDN: Software Defined Network
  - Separation of
    - Forwarding Plane; by hardware
    - Control Plane; by software
- NFV: Network Function Virtualization
  - Network function by software with virtualization technologies (e.g., virtual machine, container, process)

Networking Operating System

- "Operating System" (OS)
  - Fundamental system software in charge of
    - Resource management (hardware/software)
    - Protection, Filesystem, multitasking etc.
- COTS Network Facilities (using generic CPU) = *Networking Operating System*
  - Inexpensive
  - Flexible
  - Extensible

Departing from Generic OS



Generic OS: Not designed for networking facilities

Fat kernel, many overhead, dirty-slate

⇨ Clean-slate approach

Networking Operating System from Scratch

- *"from scratch"*
  1. Evaluate the *best* performance of COTS hardware
     - Bottleneck analysis
  2. Design new algorithm/architecture
     - Scheduler
     - Memory management
     - Protection
     - Protocol stack
     - Routing table lookup algorithm

Towards High-Performance Network Facilities with COTS hardware

Network Facilities with COTS hardware

- Background
  - Generic CPU (IA) for packet processing
  - PCIe NIC for packet forwarding
- Goal: High-performance network facilities w/ software
  - Router: 40GbE/100GbE line-rate routing (1M RiB entries)
  - Middlebox: Firewall, Load-balancer, etc.
  - Server apps: HTTP, Authentication, Accounting, etc.

Network Facilities with COTS hardware

- Background
  - Generic CPU (IA) for packet processing
  - PCIe NIC for packet forwarding
- Goal: High-performance network facilities w/ software
  - Router: 40GbE/100GbE line-rate routing (1M RiB entries)
  - Middlebox: Firewall, Load-balancer, etc.
  - Server apps: HTTP, Authentication, Accounting, etc.

VFSR: Very Fast Software Router

- Essential Components
  1. Fast packet forwarding
     - High-rate per core/port for in-order processing
  2. Fast IP routing table lookup
     - # of routes: >512k (envisioning >800k)
     - High-rate per core for in-order processing

Key Numerical Values of "fast": Packet Rate for 10/40/100GbE

- Ethernet
  - Minimum frame length: 64-Byte (=Maximum frame rate)
    - 1GbE: 1.488Mpps
      = 672 ns/packet
    - 10GbE: 14.88Mpps
      = 67.2 ns/packet
    - 40GbE: 59.52Mpps
      = 16.8 ns/packet
    - 100GbE: 148.8Mpps
      = 6.72 ns/packet

VFSR: Very Fast Software Router

- Essential Components
  1. Fast packet forwarding
     - High-rate per core/port for in-order processing
  2. Fast IP routing table lookup
     - # of routes: >512k (envisioning >800k)
     - High-rate per core for in-order processing

Myths on Packet Forwarding

- Bottlenecks in packet forwarding
  - CPU is slow.
  - Memory copy is so heavy.
  - Interrupts incur excessive overheads.

Myths on Packet Forwarding

- Bottlenecks in packet forwarding
  - CPU is slow.
    - Yes, for packet processing, but forwarding requires only a set of simple instructions
      - e.g., 0.3 ns / CPU cycle @ 3.3GHz CPU
  - Memory copy is so heavy.
  - Interrupts incur excessive overheads.

Myths on Packet Forwarding

- Bottlenecks in packet forwarding
  - CPU is slow.
    - Yes, for packet processing, but forwarding requires only a set of simple instructions
      - e.g., 0.3 ns / CPU cycle @ 3.3GHz CPU
  - Memory copy is so heavy.
    - At least, throughput is enough.
      - e.g., DDR3-1866 Dual Channel: 29.867GB/s (238.933Gbps)
  - Interrupts incur excessive overheads.

Myths on Packet Forwarding

- Bottlenecks in packet forwarding
  - CPU is slow.
    - Yes, for packet processing, but forwarding requires only a set of simple instructions
      - e.g., 0.3 ns / CPU cycle @ 3.3GHz CPU
  - Memory copy is so heavy.
    - At least, throughput is enough.
      - e.g., DDR3-1866 Dual Channel: 29.867GB/s (238.933Gbps)
  - Interrupts incur excessive overheads.
    - Not excessive, but non-negligible for 100 GbE
      - Discuss this later

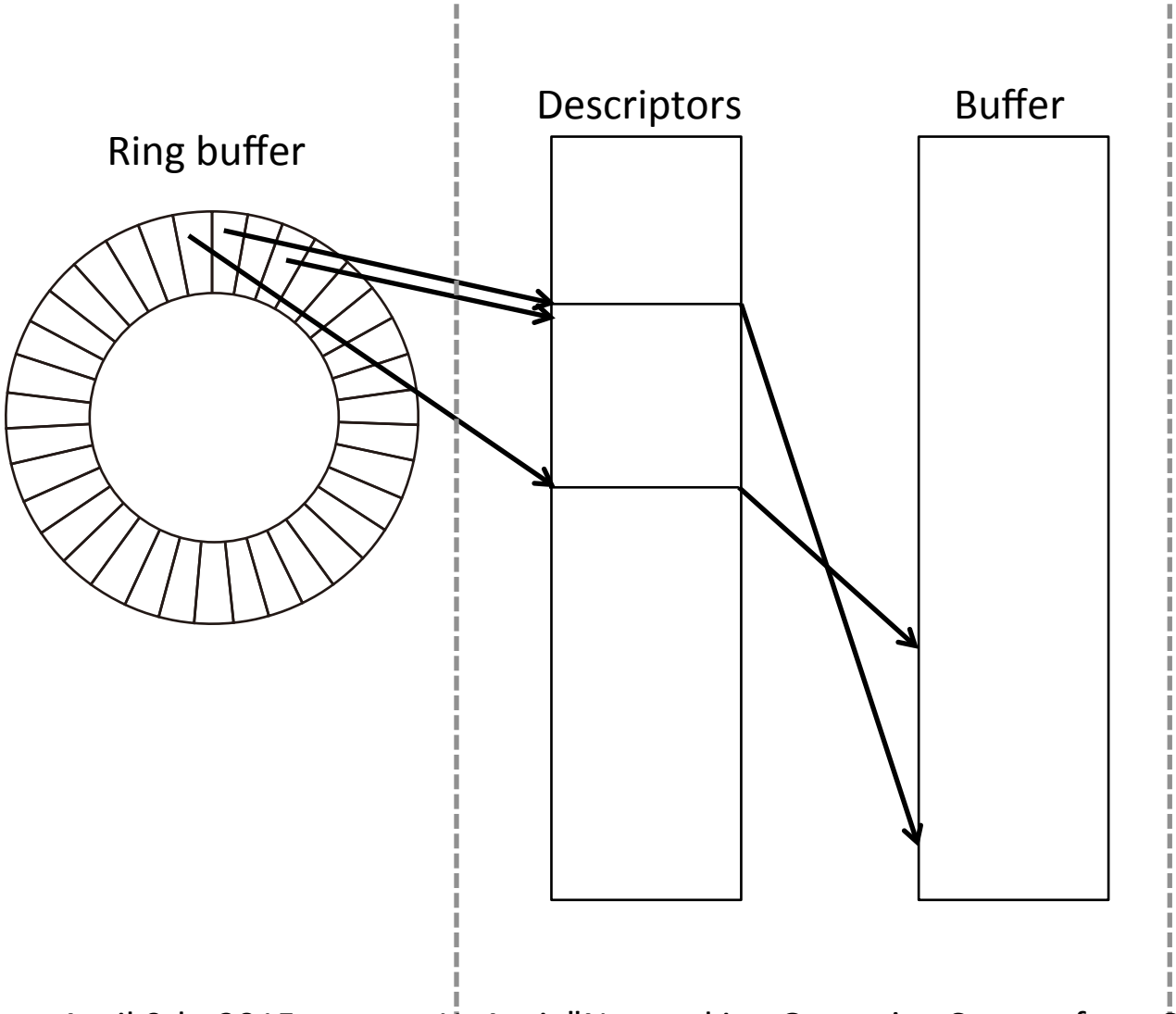
Real Bottleneck on Packet Forwarding

- PCIe device register access
  = Memory Mapped I/O (MMIO)
  - No cache
    - ~250ns/access [Miller et al. ACM ANCS '09]
  - **Read**
    - 1529.17 cycles / read
    - 392.1 ns / read
  - **Write**
    - 282.621 cycles / write
    - 72.47 ns / write

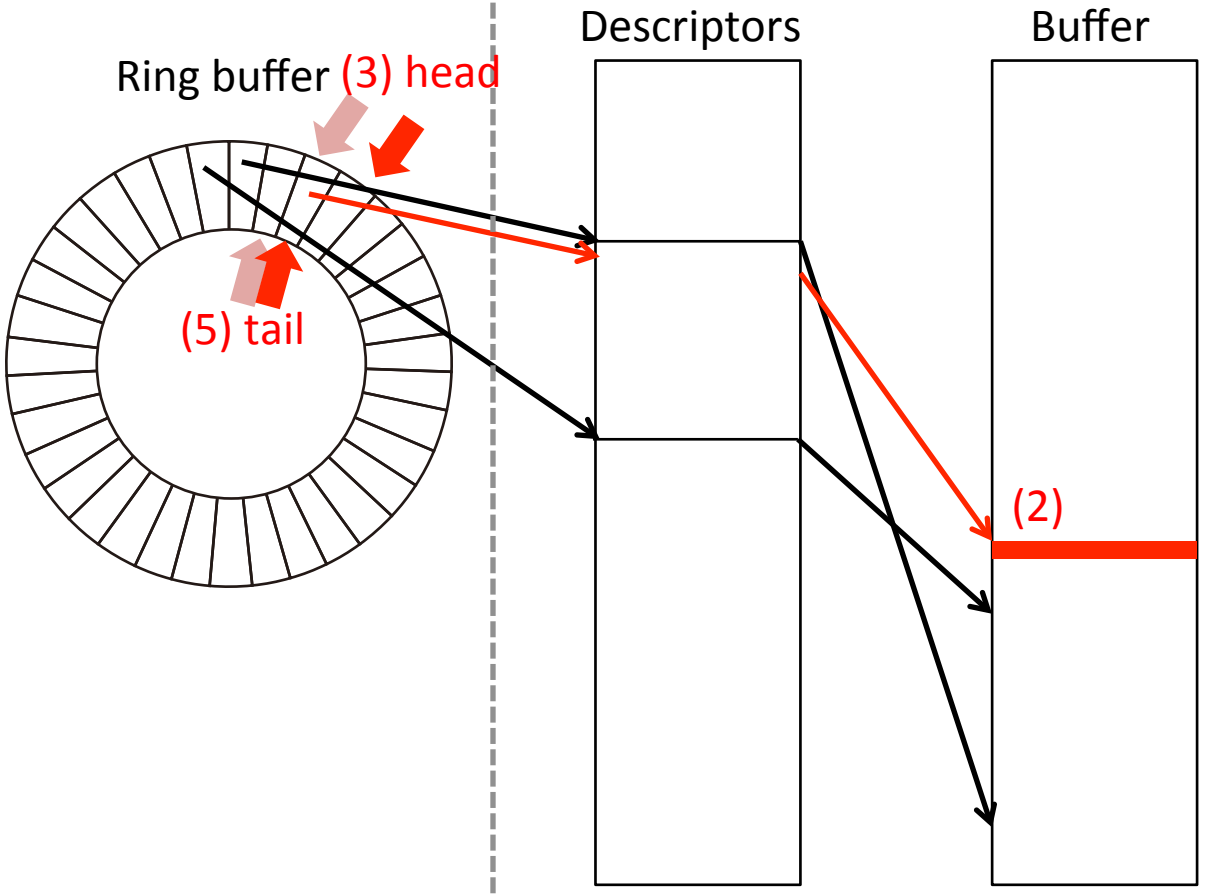
※Measure CPU cycles to access to the same register
1 million times by Performance Monitoring Counter (PMC)

CPU: Intel Core i7 4770K
Memory: Corsair DDR3-1866 8GB x4
NIC: Intel X520-DA2

Review: Generic NIC Architecture



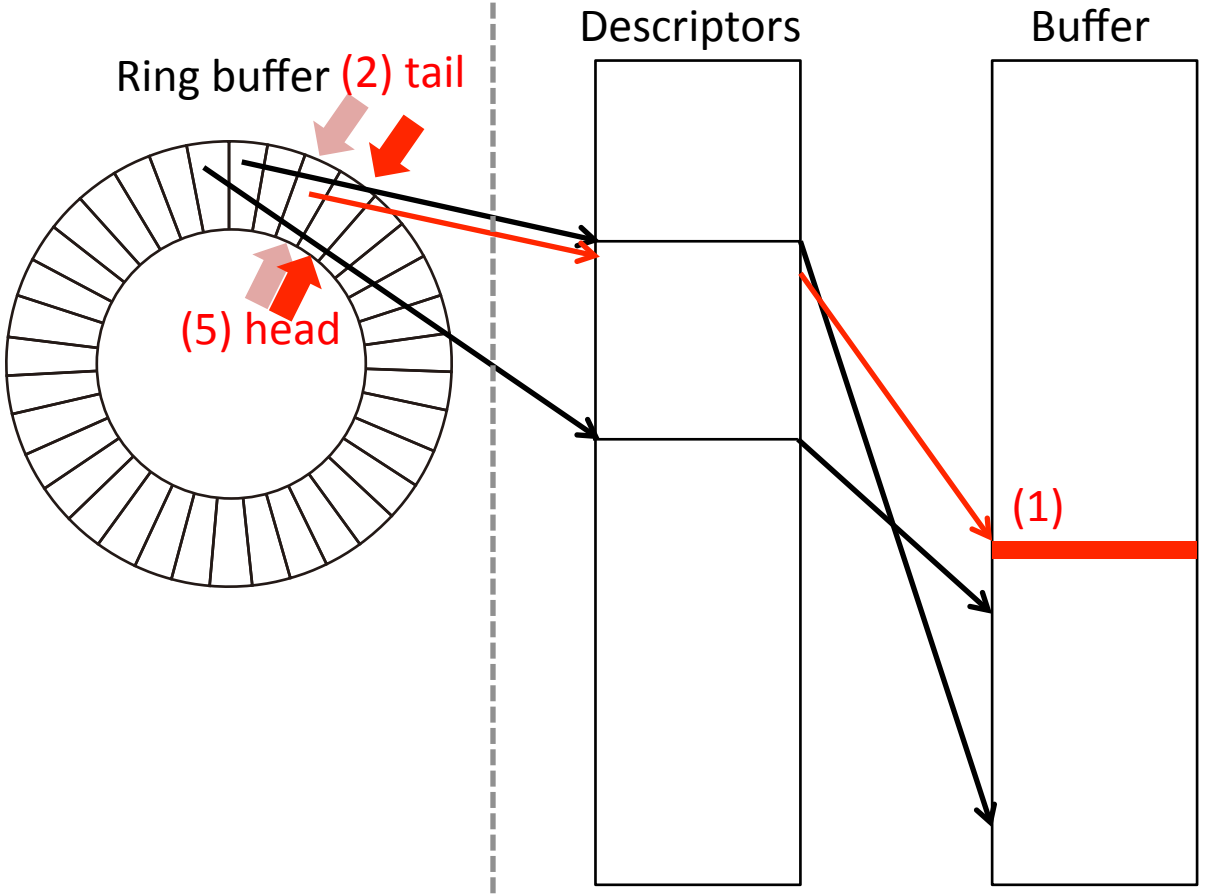
Review: Generic NIC Architecture



Packet reception

1. NIC receives a packet
2. NIC transfer the packet data to a buffer in RAM via DMA
3. NIC proceeds the head pointer
4. Software processes the packet
5. Software proceeds the tail pointer to release the packet

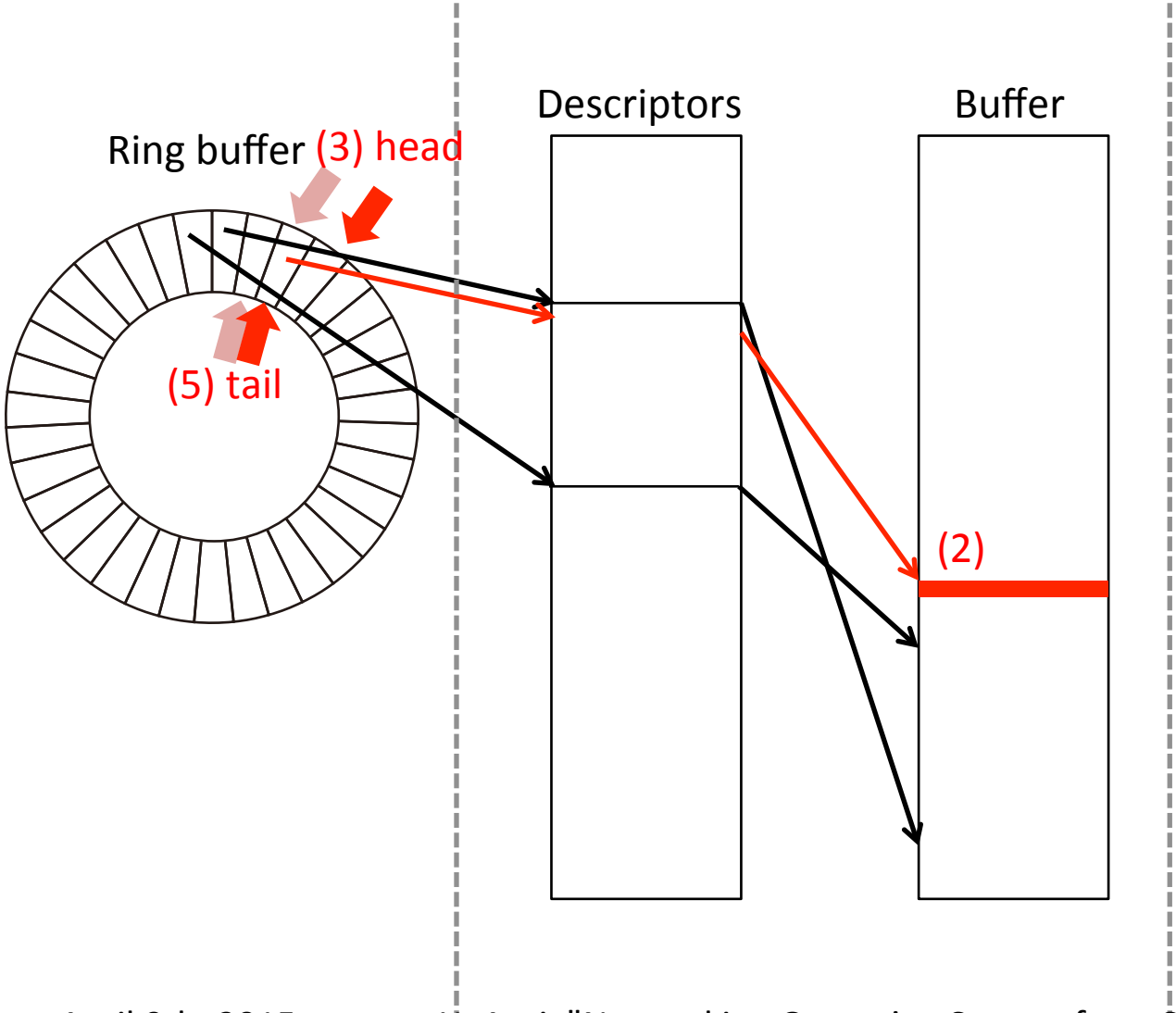
Review: Generic NIC Architecture



Packet transmission

1. Software writes a packet to a buffer in RAM
2. Software proceeds the tail pointer to commit the packet
3. NIC transfer the packet data from the buffer in RAM via DMA
4. NIC transmit the packet
5. NIC proceeds the head pointer to notify the packet is transmitted

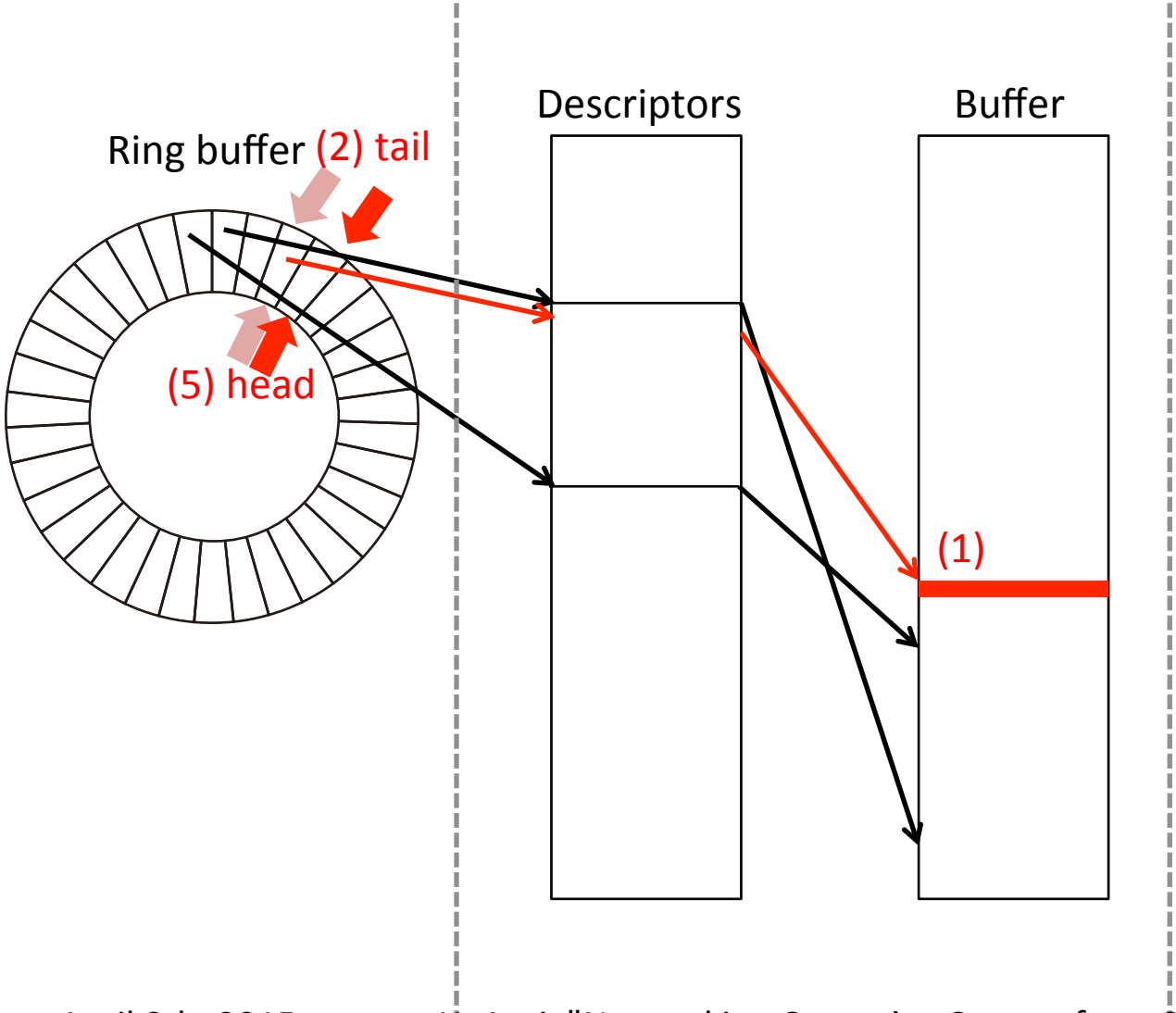
Review: Generic NIC Architecture



Packet reception

1. NIC receives a packet
2. NIC transfer the packet data to a buffer in RAM via DMA
3. NIC proceeds the head pointer
4. Software processes the packet
5. Software proceeds the tail pointer to release the packet

Review: Generic NIC Architecture



Packet transmission

1. Software writes a packet to a buffer in RAM
2. Software proceeds the tail pointer to commit the packet
3. NIC transfer the packet data from the buffer in RAM via DMA
4. NIC transmit the packet
5. NIC proceeds the head pointer to notify the packet is transmitted

Polling & Bulk Processing (Transmission, Intel® X520)

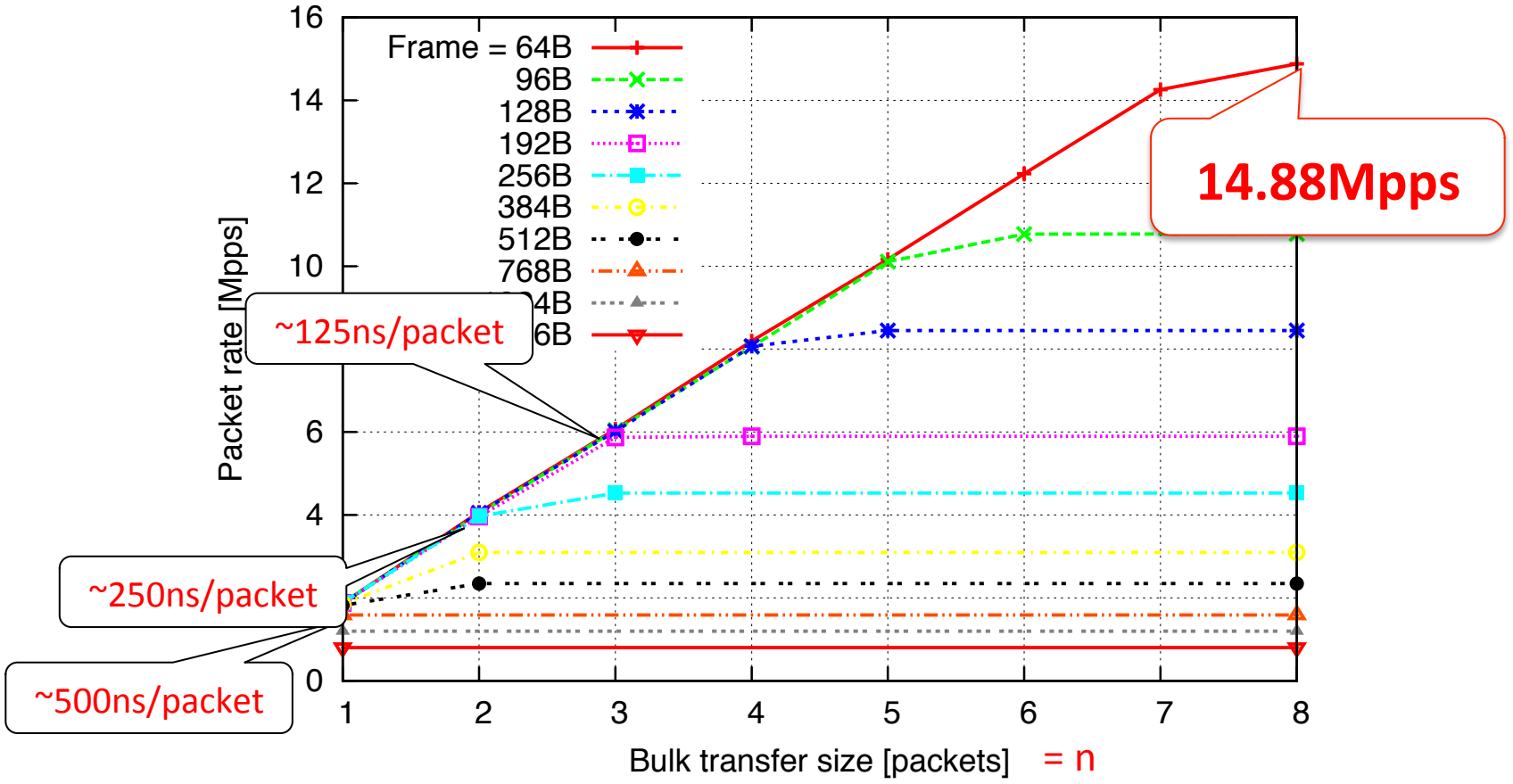
```
txq_tail = 0;
for ( ;; ) {
    txq_head = read_txq_head();
    /* Available Tx queue length */
    txq_len = txq_sz
        - (txq_sz - txq_head + txq_tail) % txq_sz;
    /* Check the available Tx queue length */
    if ( txq_len < n ) continue;
    for ( i = 0; i < n; i++ ) {
        // Set packet to the ring buffer to txq_tail
        txq_ring[txq_tail].pkt = pkt_to_transmit;
        txq_tail = (txq_tail + 1)  % txq_sz
    }
    /* Commit */
    write_txq_tail(txq_tail);
}
```

~392.1ns

Note: Can be optimized…

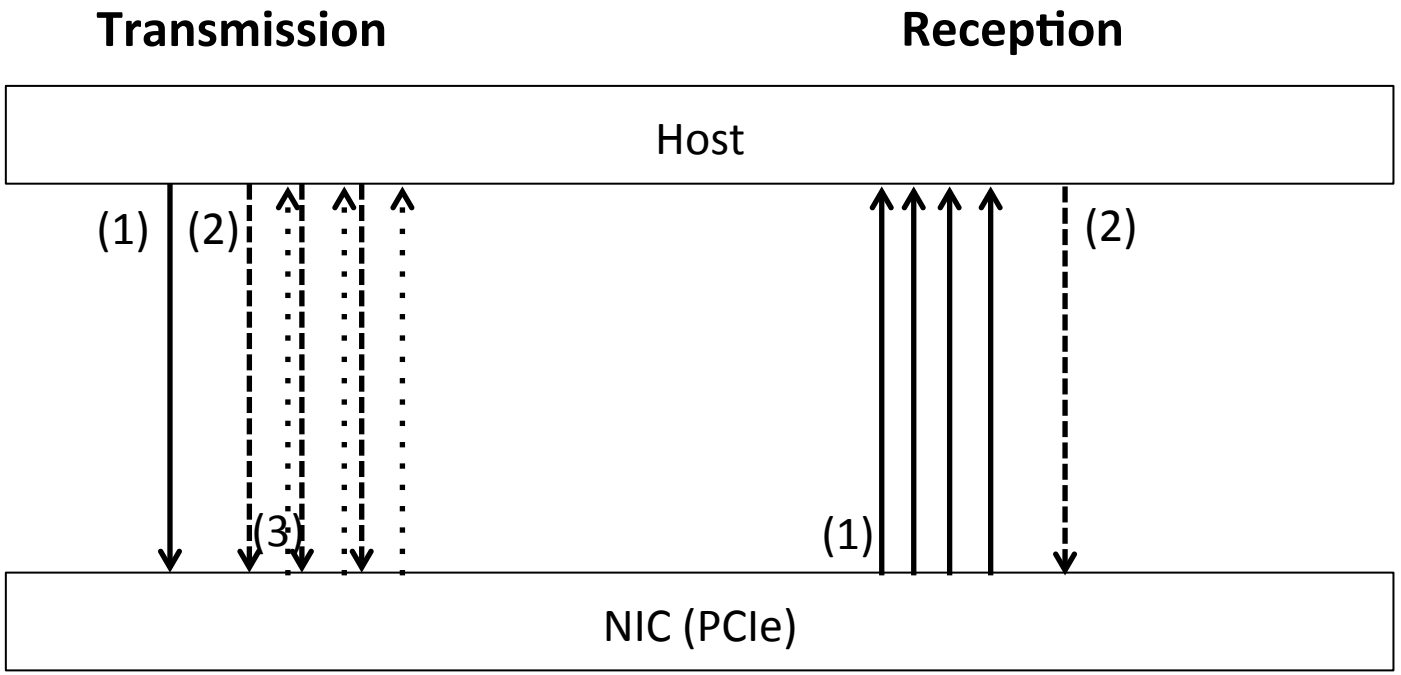
~72.47ns

Tx Performance by bulk size (Intel® X520)



Note: Also confirmed 59.52 Mpps Tx (2 Intel® X520-DA2) @ 1 core from Intel® Core i7-4770K

Intel® XL710's Operation



(1) Write the tail pointer (MMIO write)
(2) Transfer the packets via DMA
(3) Write-back the transfer status

(1) Transfer the packets via DMA with status
(2) Write the tail pointer (MMIO write)

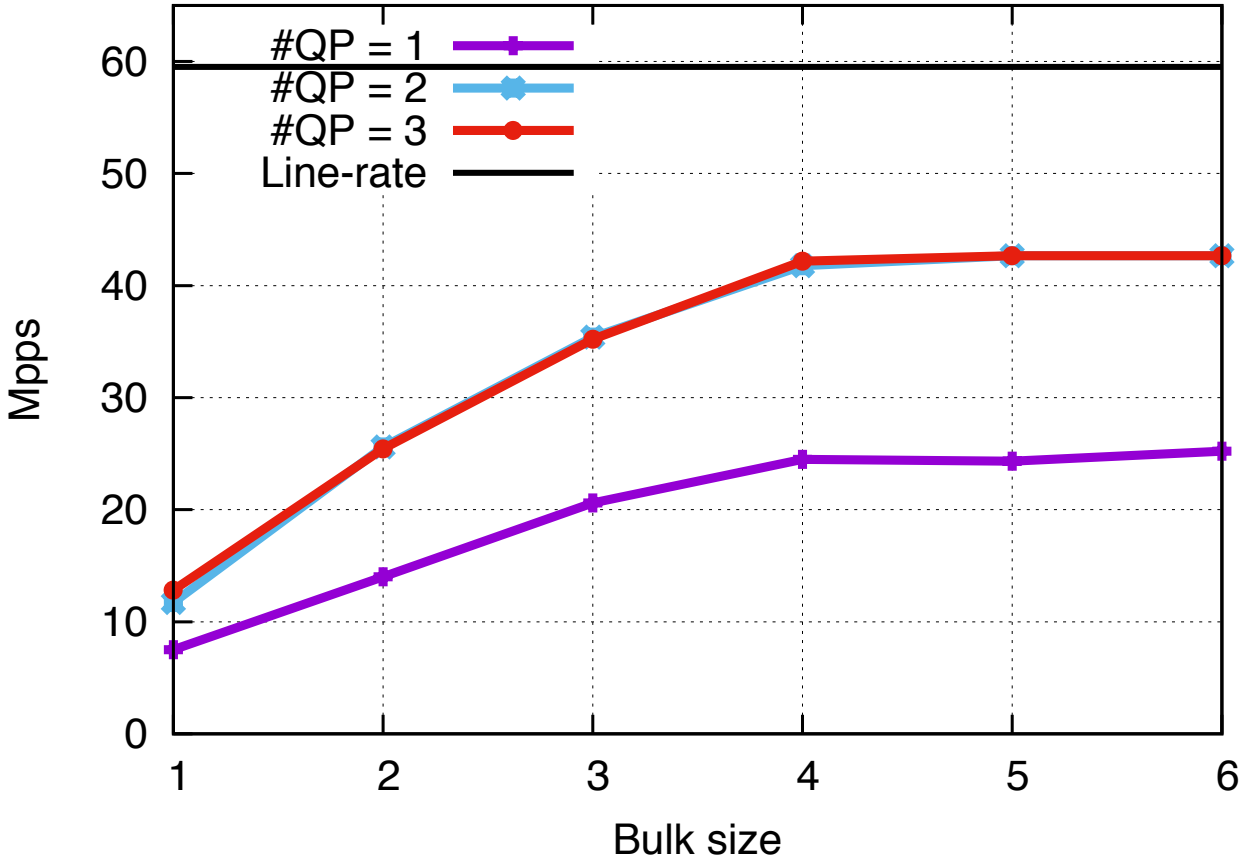
Polling & Bulk Processing (Transmission, Intel® XL710)

```
txq_tail = 0;
for ( ;; ) {
    completed = check_wb_status(txq_tail, n);
    /* Check the transmission is completed */
    if ( !completed ) continue;
    for ( i = 0; i < n; i++ ) {
        // Set packet to the ring buffer to
        txq_ring[txq_tail].pkt = pkt_to_transm
        txq_tail = (txq_tail + 1)  % txq_sz
    }
    /* Commit */
    write_txq_tail(txq_tail);
}
```

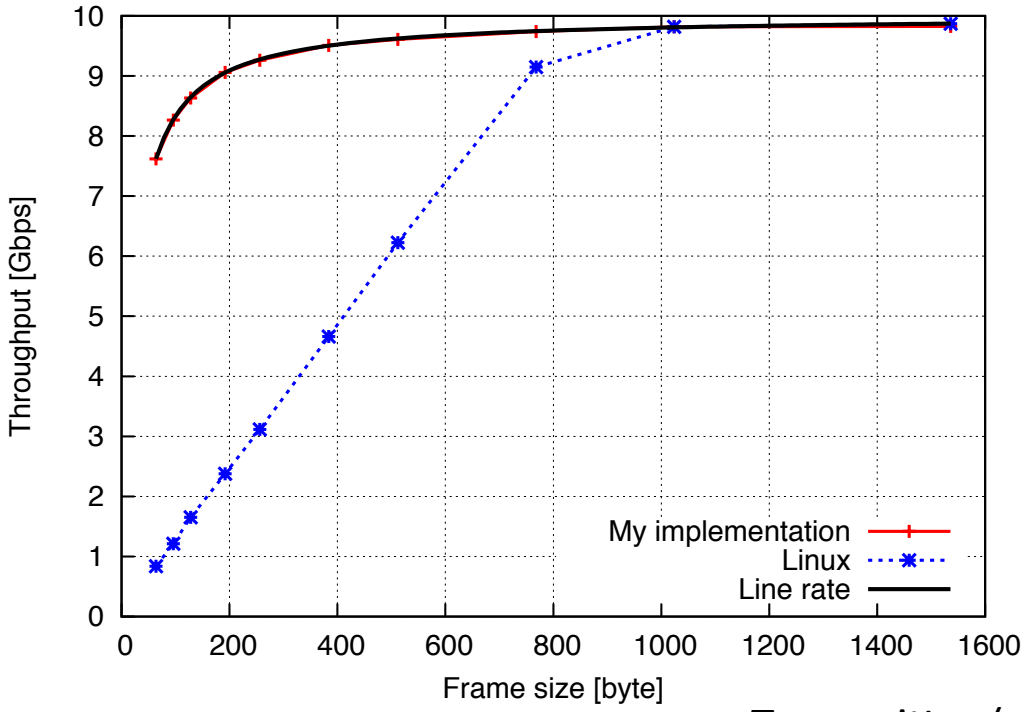
Note: Can be optimized…

PCIe MMIO

Intel® XL710's performance

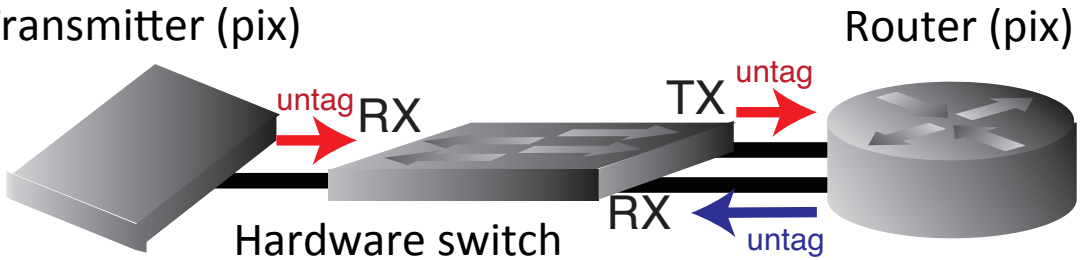


Same Strategy for Forwarding (Routing for 1 route)



Bulk polling & transmission
(dynamic bulk size = received queue length)

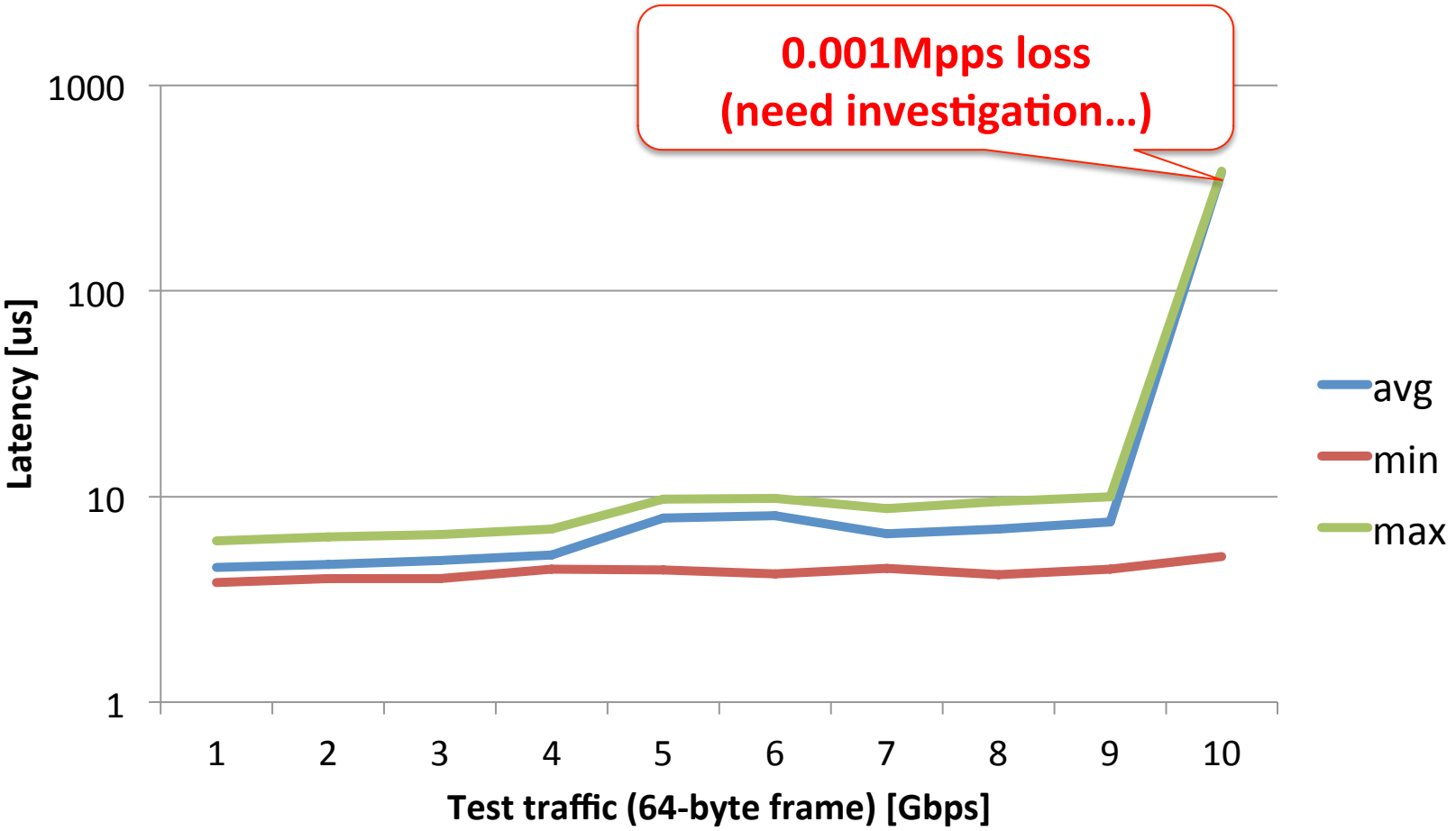
※1 route
TTL and checksum calculation
are done by CPU



Latency measurement

- Experimental setup
  - Tester
    - Spirent Communications Spirent TestCenter
      - Chassis: SPT-N4U-110
      - Module: CV-10G-S8
    - Supported by 株式会社東陽テクニカ様 during Interop Tokyo 2014

Low Latency



Low latency（~10us）for 90% of line-rate traffic

Revisiting the Overhead of Interrupts for Faster Packet Processing & I/O

```
pushq   %rax
pushq   %rbx
pushq   %rcx
pushq   %rdx
pushq   %rdi
pushq   %rsi
pushq   %rbp
pushq   %r8
pushq   %r9
pushq   %r10
pushq   %r11
pushq   %r12
pushq   %r13
pushq   %r14
pushq   %r15
call    _kintr
```

```
popq    %r15
popq    %r14
popq    %r13
popq    %r12
popq    %r11
popq    %r10
popq    %r9
popq    %r8
popq    %rbp
popq    %rsi
popq    %rdi
popq    %rdx
popq    %rcx
popq    %rbx
popq    %rax
iretq
```

Push 15 general purpose registers onto the stack, pop 15 general purpose registers from the stack, and then return to the restore point while popping the original stack pointer etc.

	Latency	Throughput
PUSH (@0F_2H)	1.5	1
POP (@0F_2H)	1.5	1
CLI (@06_2A/2D)	5	2

Referred from Intel® 64 and IA-32 Architectures Optimization Reference Manual

30 CPU cycles for push/pop instructions

➔ **10 ns** @3GHz CPU

Interim Summary

- Faster packet forwarding
  - Reduce slow PCIe MMIO
    - ➔ Key: Bulk processing
      - **Read**
        - 392.1 ns / read
      - **Write**
        - 72.47 ns / write
  - Avoid using interrupt handlers for 40GbE/100GbE
    - ➔ Key: Polling, Tickless
      - 10 ns to save and restore CPU's registers

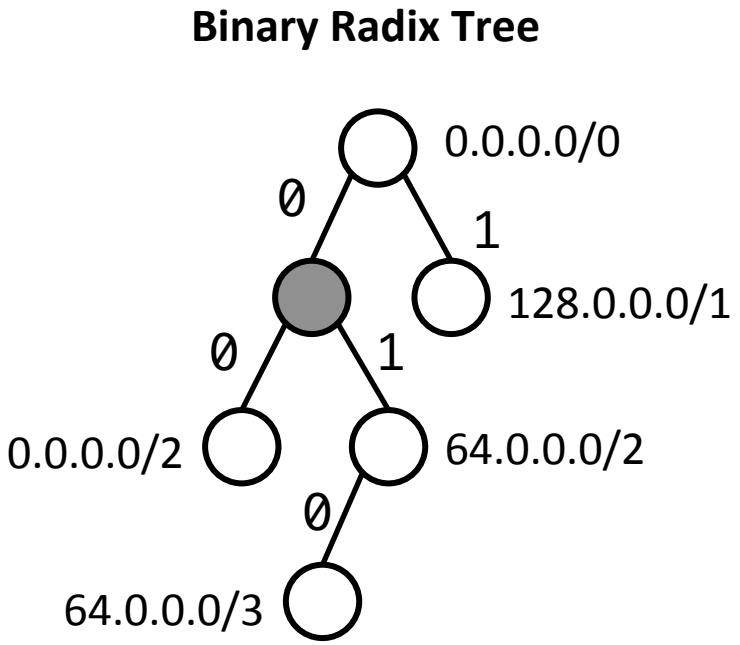
VFSR: Very Fast Software Router

- Essential Components
  1. Fast packet forwarding
     - High-rate per core/port for in-order processing
  2. Fast IP routing table lookup
     - # of routes: >512k (envisioning >800k)
     - High-rate per core for in-order processing

Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup

Hirochika Asai (Univ. of Tokyo)
Yasuhiro Ohara (NTT Communications)

Fundamental Algorithm for Longest Prefix Match



Problem with binary radix tree

- Depth up to 32 (for IPv4)
- Too many pointers

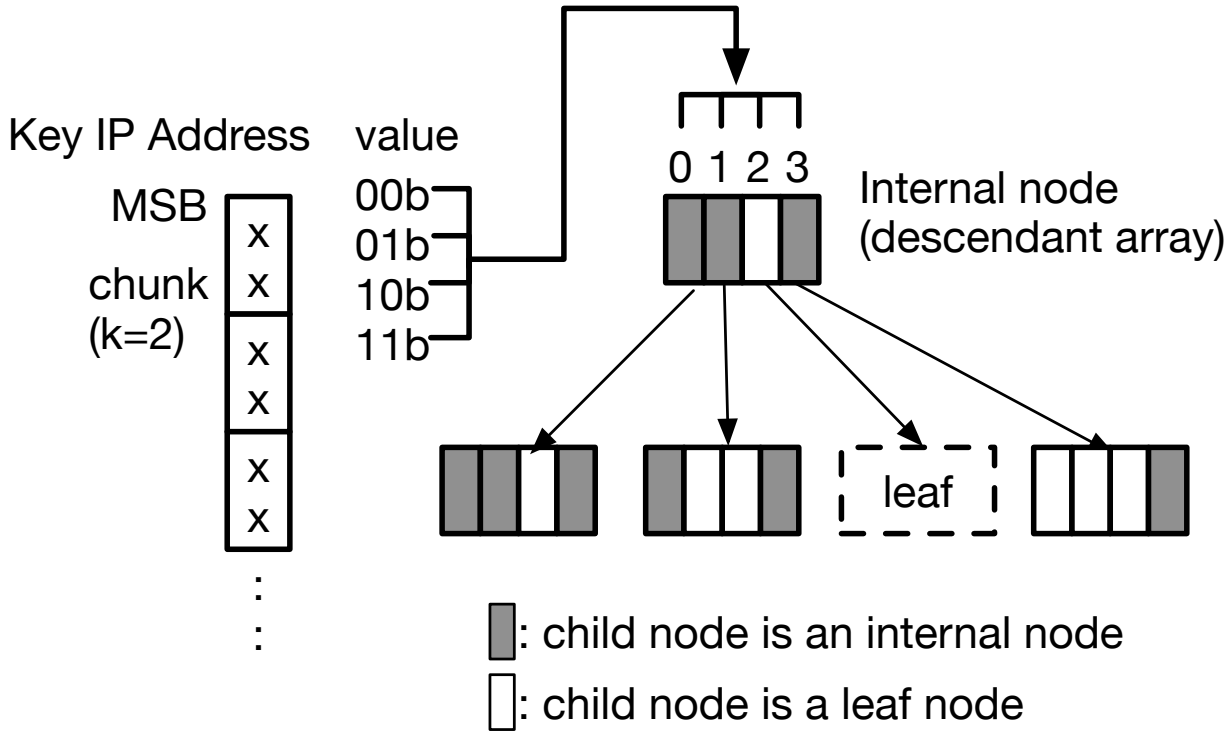
➔ **Slow**

Principle Ideas towards Faster IP Routing Table Lookup Algorithm

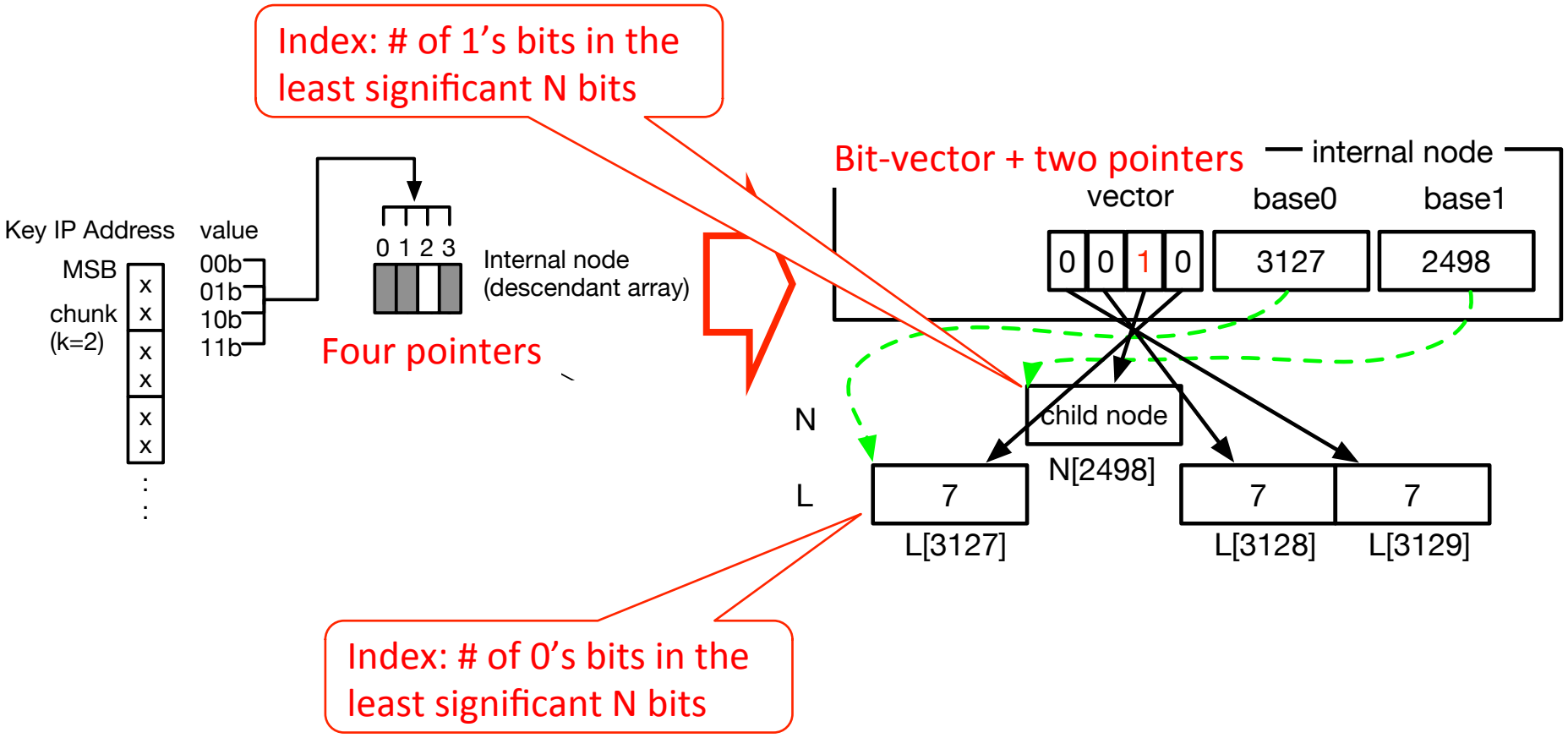
- Reduce the number of instruction, especially memory access
  - 1 or a few cycles for most of bitwise instructions
  - Memory access latency (in Intel Core i7-4770K)
    - L1 cache: 4-5 cycles
    - L2 cache: 12 cycles
    - L3 cache: 27.85 cycles
    - DRAM: ~65 ns
- Reduce memory footprint
  - Maximize CPU cache efficiency
    - L1/L2/L3 cache size in Intel Core i7-4770K
      - 64 KiB, 256 KiB, 8 MiB

Reduce the number of instructions: Starting from 2^k-ary radix tree

To reduce the depth of the tree (i.e., # of memory access)



Reduce memory footprint: Pointer Compression w/ Population Count

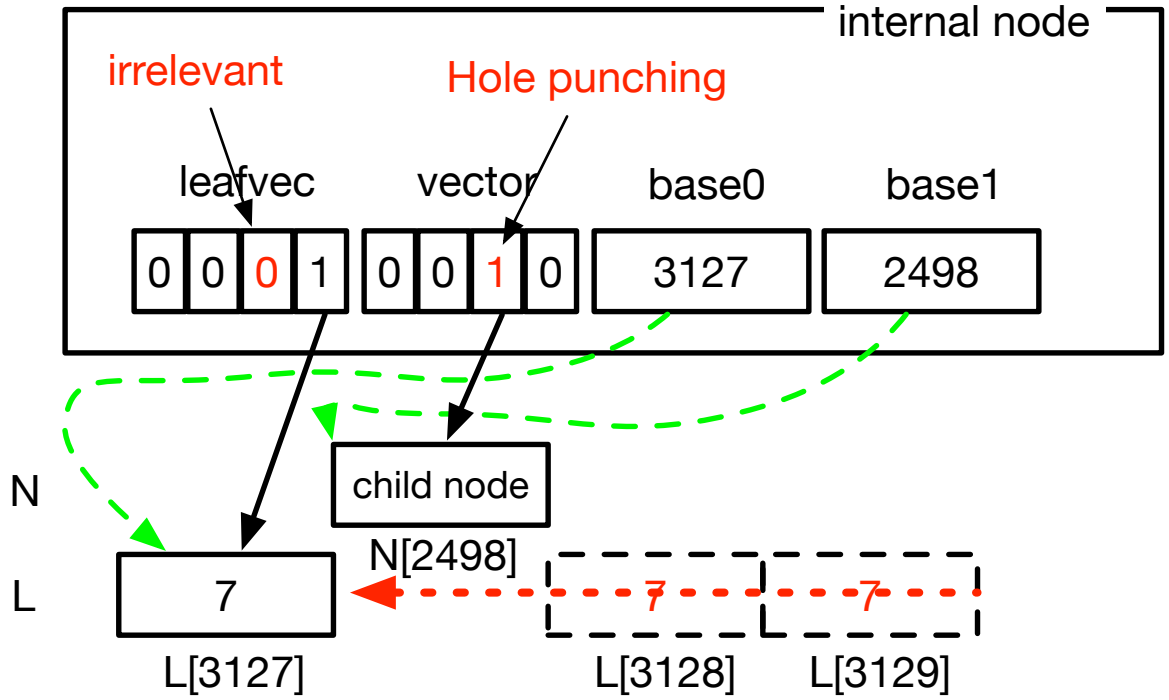


Which k? 64-bit CPU ➔ k=6 (so that vector is in 2^6 = 64 bits)

Further Compression: Leaf Vector to Remove Redundant Leaf Nodes

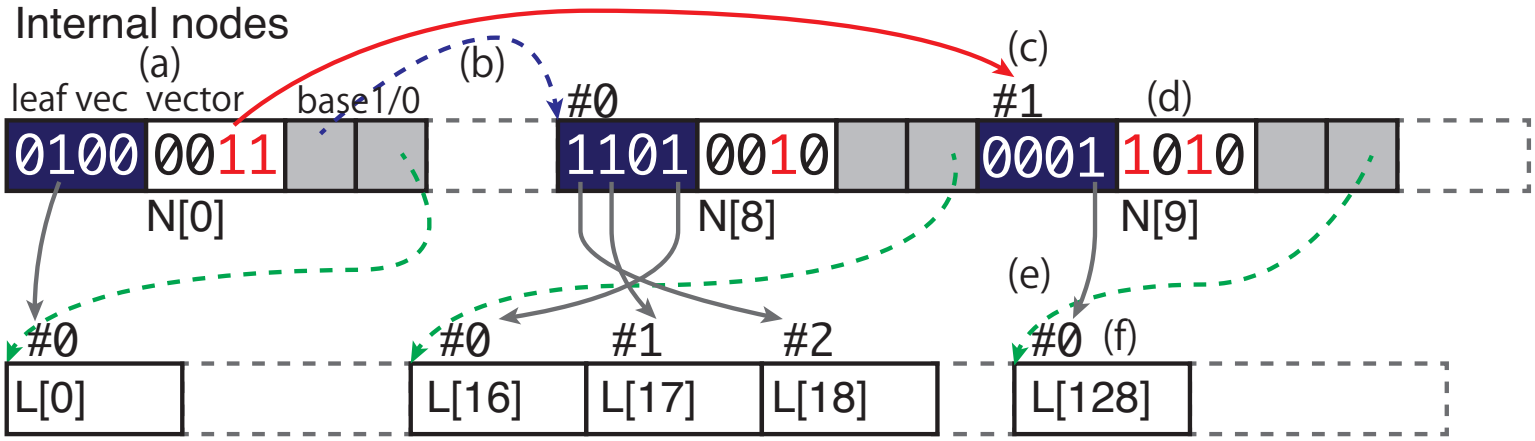
One of the problem with the basic data structure

- Redundant leaf nodes for prefixes that do not match k-bit boundary
- e.g., /1 (/7, etc. as well) may create 32 redundant leaf nodes when k=6

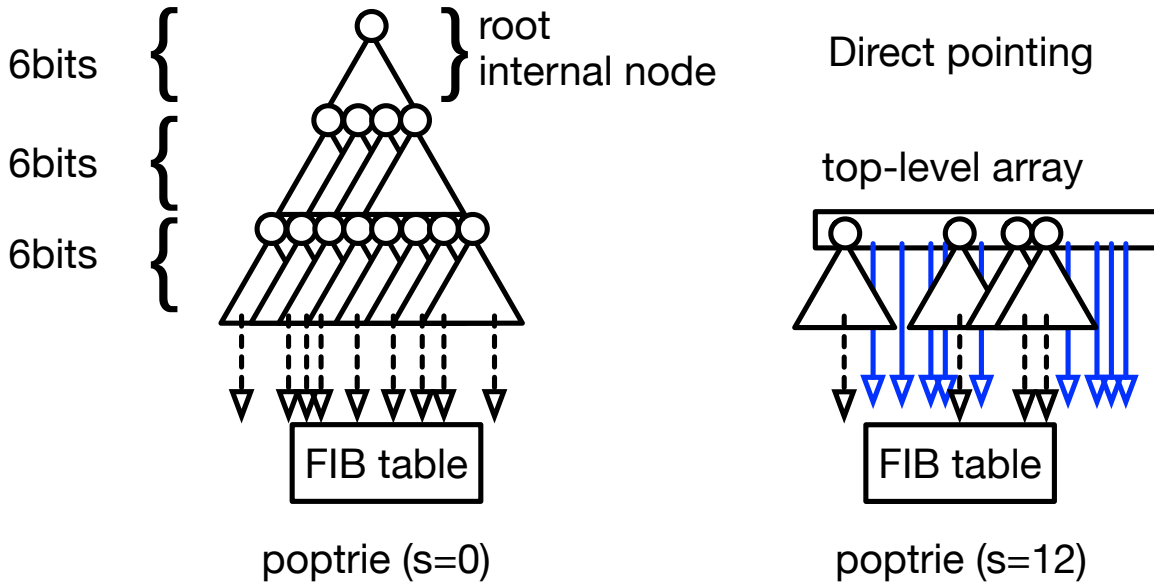


Visualized Lookup Algorithm Example

For the destination address 0110b



Direct Pointing

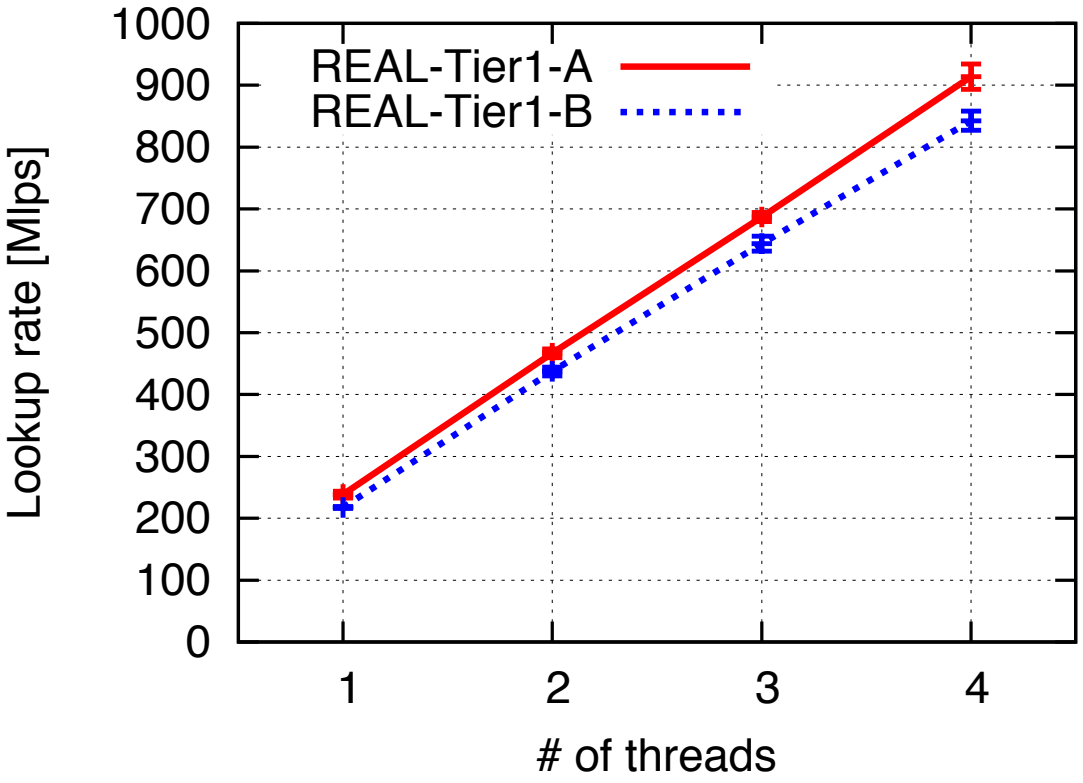


(a) Without Direct Pointing. (b) Direct Pointing.

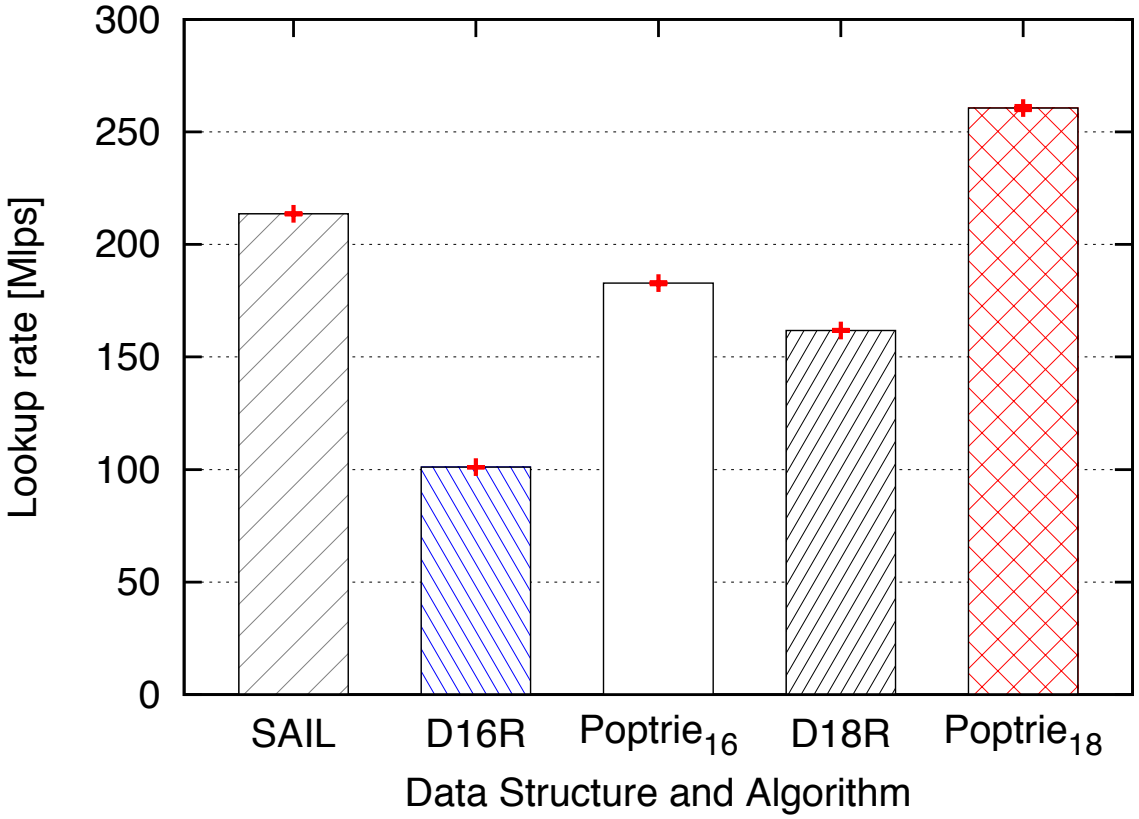
Lookup s bits at the first stage (like other algorithms)

Evaluation for Random Traffic

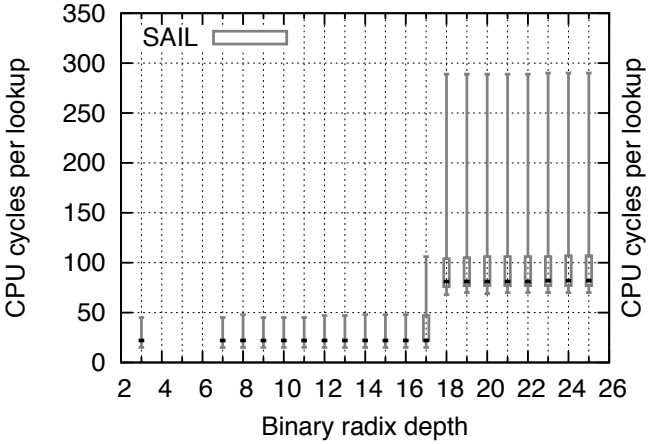
REAL-Tier1-A: Global Tier-1's BGP Router
REAL-Tier1-B: Domestic ISP's BGP Router

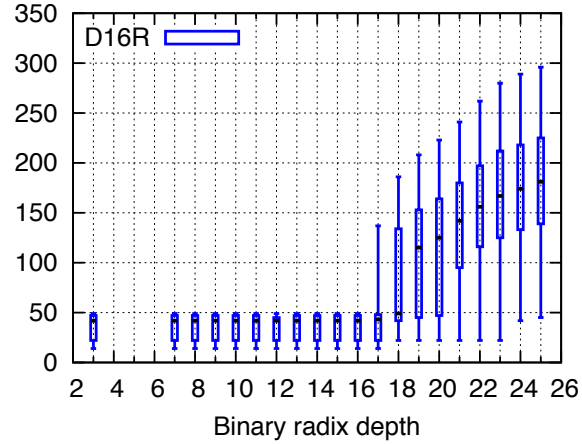


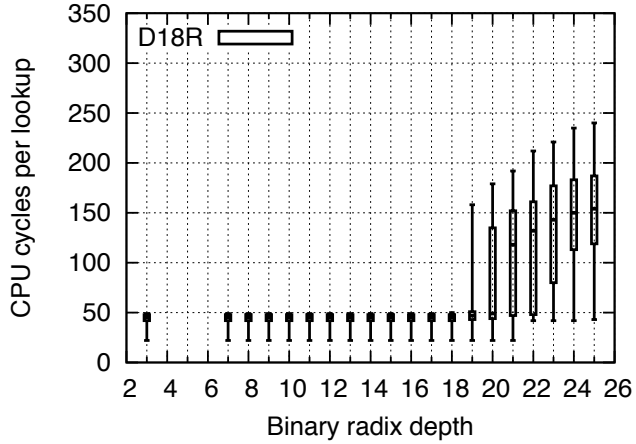
Performance Evaluation for Real Traffic (WIDE Transit)

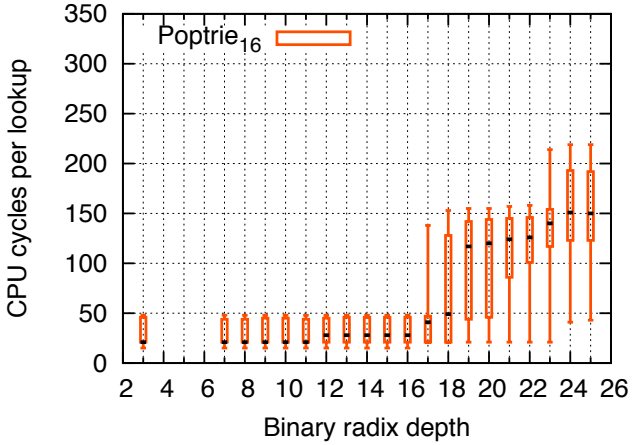


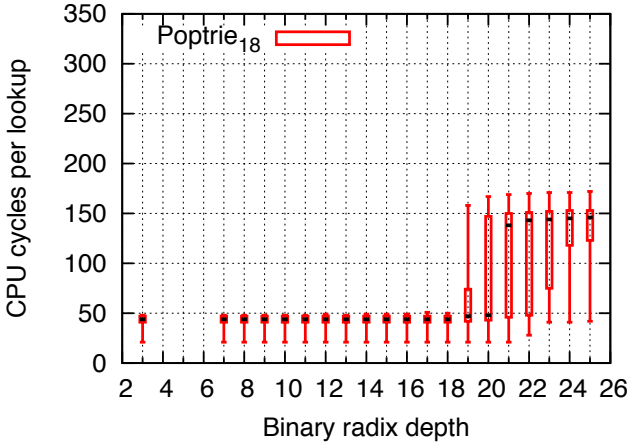
Detailed Analysis on CPU Cycles per Lookup for Random Traffic











Structural Scalability

Lookup performance for random traffic [Mlps]

Algorithm	SYN1 -Tier1-A	SYN1 -Tier1-B	SYN2 -Tier1-A	SYN2 -Tier1-B
SAIL	102.86	99.98	N/A	N/A
D18R (modified)	115.45	117.48	102.59	104.22
Poptrie_{18}	188.02	187.69	174.42	175.04

RIB dataset (synthetic RIB dataset)

Name	# of prefixes	# of nhops	Name	# of prefixes	# of nhops
SYN1-Tier1-A	764,847	45	SYN2-Tier1-A	885,645	87
SYN1-Tier1-B	756,406	19	SYN2-Tier1-B	876,944	33

Interim Summary

- Fast IP routing table lookup
  - 914 Mlps w/ 4 core
    - Global tier-1 ISP's full route (531k routes)
    - Random traffic
  - 175 Mlps per core
    - Synthetic 800k routes
    - Random traffic

Ongoing Project

- Socket API extension for middleboxes/VNF (Virtualized Network Function)
  - **Virtual machine** ← ETSI's approach
    - Network abstraction: Virtual NIC
      - Pros: Any kinds of OS works
      - Cons: Overhead of virtualization (incl. VMEntry/VMExit)
  - **Container**
    - Network abstraction: Virtual NIC
      - Pros: Linux works (when we use Linux Container)
      - Cons: Overhead of virtualized NIC driver
  - **Process** ← My focus
    - Network abstraction: Socket API
      - Pros: No overhead (a few scheduler overhead) in my design
      - Cons: Socket API is not good for packet processing

Non-TCP/UDP Socket

- Existing socket / IPPROTO
  - SOCK_RAW
    - Privileged socket…
  - SOCK_DGRAM (IPPROTO_UDP) / SOCK_STREAM (IPPROTO_TCP)
    - Basically UDP/TCP (Cannot handle Ethernet, IP)
- Socket
  - SOCK_DGRAM + IPPROTO_ETHERNET (IPPROTO?)
    - Bind a MAC address
  - SOCK_DGRAM + IPPROTO_IP (IPPROTO?)
    - Bind an IP address

Conclusion

- VFSR: Very Fast Software Router
  - Essential Components
    1. Fast packet forwarding
       - High-rate per core/port for in-order processing
    2. Fast IP routing table lookup
       - # of routes: >512k (envisioning >800k)
       - High-rate per core for in-order processing
- Socket API extension for process-based NFV
  - as ongoing work