Optimizing Packet Accesses for a Domain Specific Language on Network Processors

Tao Liu[1], Xiao Feng Li[2], Lixia Liu[2], Chengyong Wu[1], Roy Ju[3]

1. Institute of Computing Technology, China Academy of Sciences
2. Intel China Research Center
3. Microprocessor Technology Labs, Intel Corporation

Outline

- Motivation
- System Overview
- Packet accesses optimizations
- Experimental results

Network Processors

- Advantages
  - More flexible than ASICs/custom design
  - Higher performance on packet processing
  - Lower development cost
- Unfortunately, difficult to program
  - Complicated hardware
  - Limited resources
  - Low level programming languages

Domain specific language: Baker

- Handle programming challenges in compiler
  - Eliminate need for assembly programming
  - Automate resource management
  - Perform domain-specific optimizations in compiler
- Assist portable application development
  - Protocol stack component modularity
  - Abstracted programming model hiding underlying hardware details
  - Build-in language types and libraries for network applications
    - Packet type
- Big headache: still achieve high performance

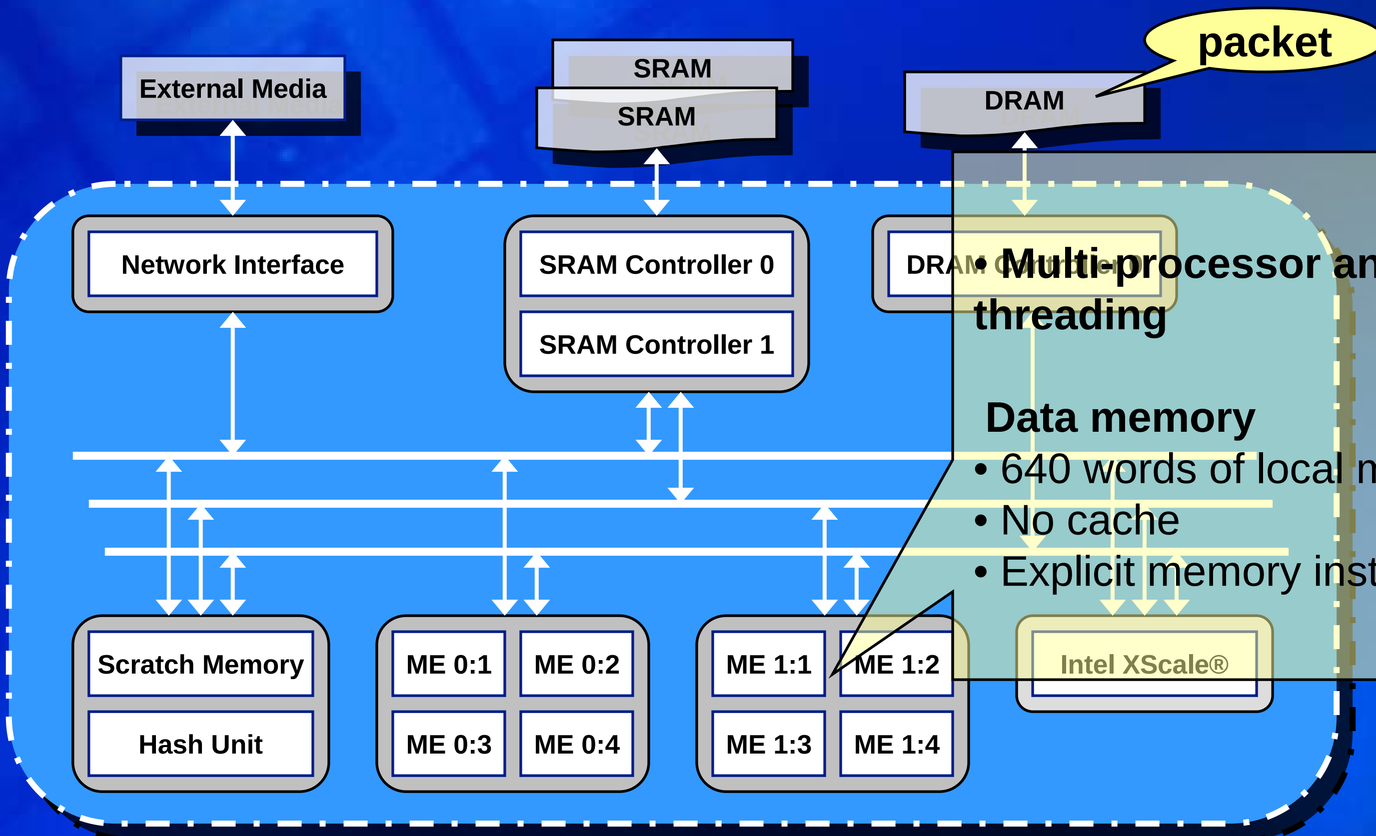
Packet accesses critical to performance

- Key factors of performance
  - Strict instructions budget per packet
    - 700 cycles on IXP2400
  - Constrained memory bandwidth
    - 2 DRAM accesses on IXP2400
- Characteristics of packet accesses
  - Consist of dozens of instructions
  - Need memory reference per access
  - Occur frequently in applications

Outline

- Motivation
- System overview
- Packet accesses optimizations
- Experimental results

Intel IXP2400 network processor

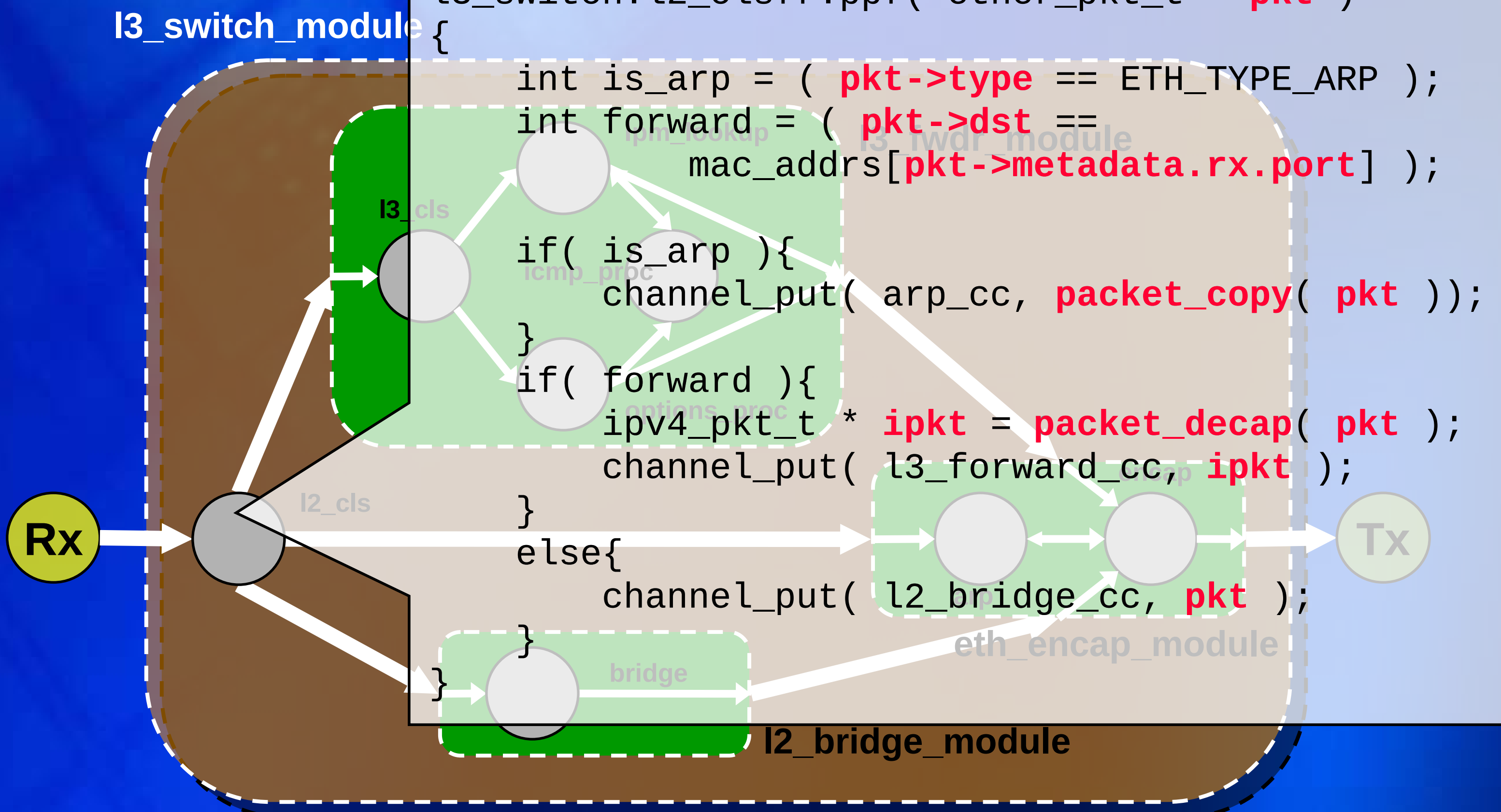


- **Multi-processor and Multi-threading**

Data memory

- 640 words of local memory
- No cache
- Explicit memory instructions

L3-Switch written in the Baker



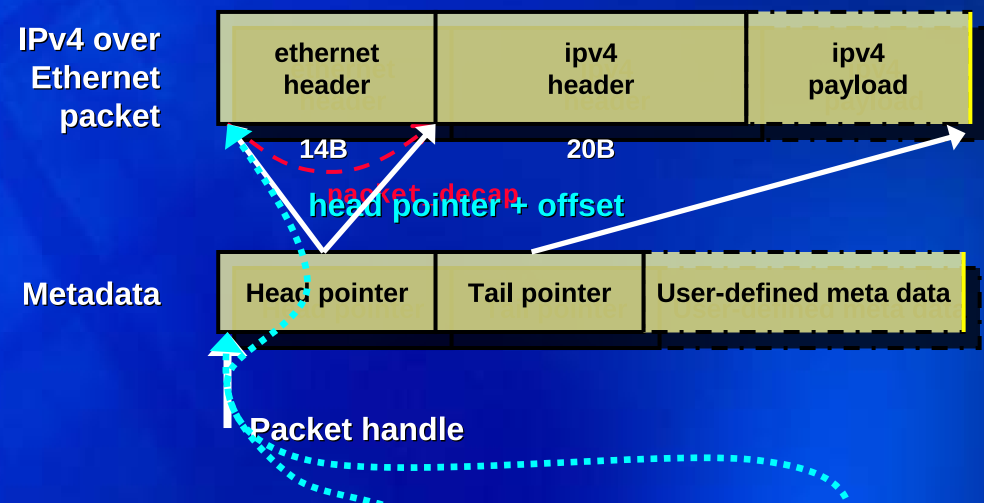

```
l3_switch.l2_clsfr.ppf( ether_pkt_t * pkt )
{
    int is_arp = ( pkt->type == ETH_TYPE_ARP );
    int forward = ( pkt->dst ==
            mac_addrs[pkt->metadata.rx.port] );

    if( is_arp ){
        channel_put( arp_cc, packet_copy( pkt ));
    }
    if( forward ){
        ipv4_pkt_t * ipkt = packet_decap( pkt );
        channel_put( l3_forward_cc, ipkt );
    }
    else{
        channel_put( l2_bridge_cc, pkt );
    }
}
```

Outline

- Motivation
- System overview
- Packet accesses optimizations
- Experimental results

Packet primitives in Baker and implementation

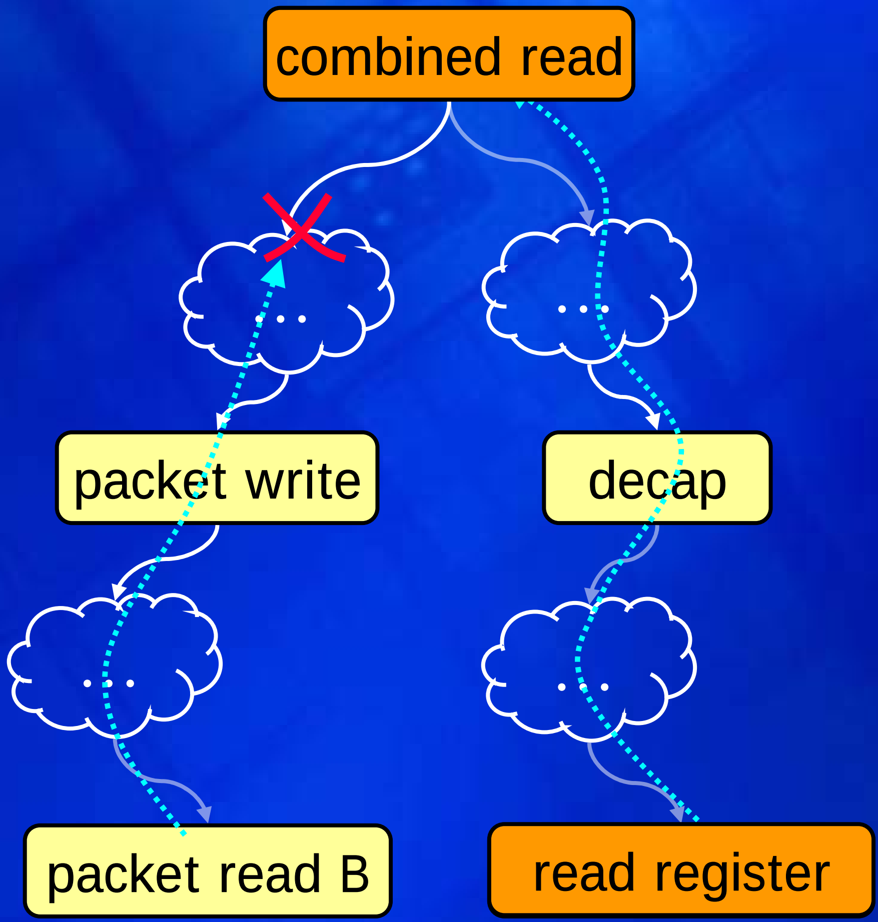
- Protocol construct
- Packet handle
- Packet access
- Decap/Encap




```
protocol ether {
  dst : 48;
  src : 48;
  type : 16;
  demux{ 14 };
};
```

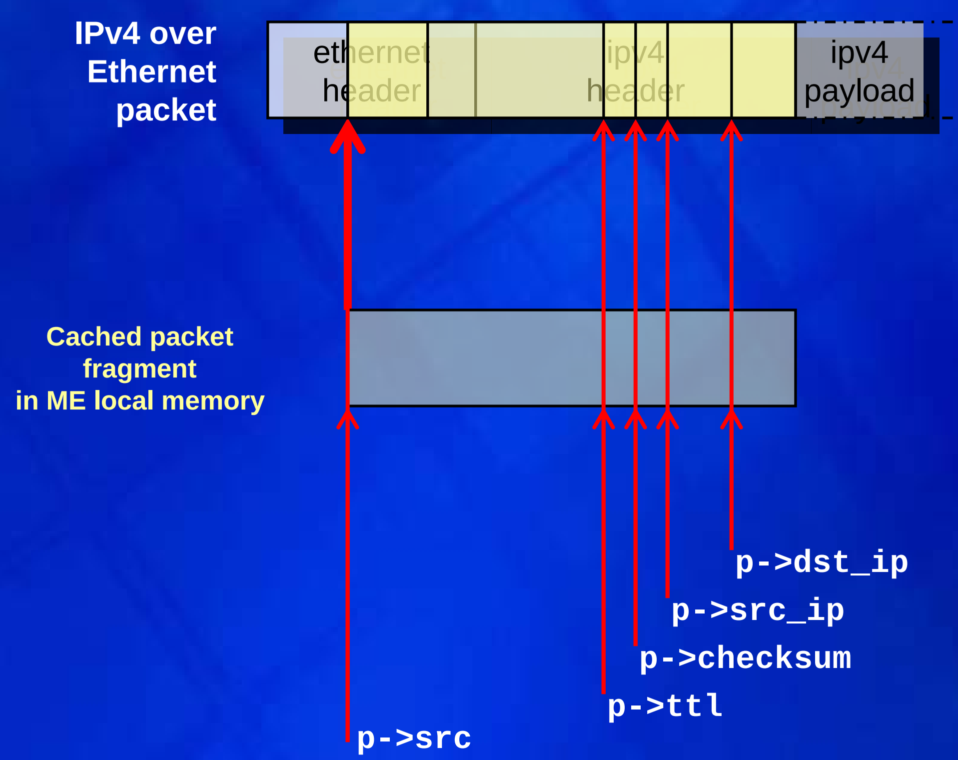
```
void A.process(ether_packet_t* in_pkt){
  ipv4_packet_t* p;
  mac_addr_t mac;
  mac = in_pkt->dst;
  if(fwd){
    p = packet_decap(in_pkt);
    channel_put(l3_fwdr_chnl,p);
  }}
```

Packet access combining



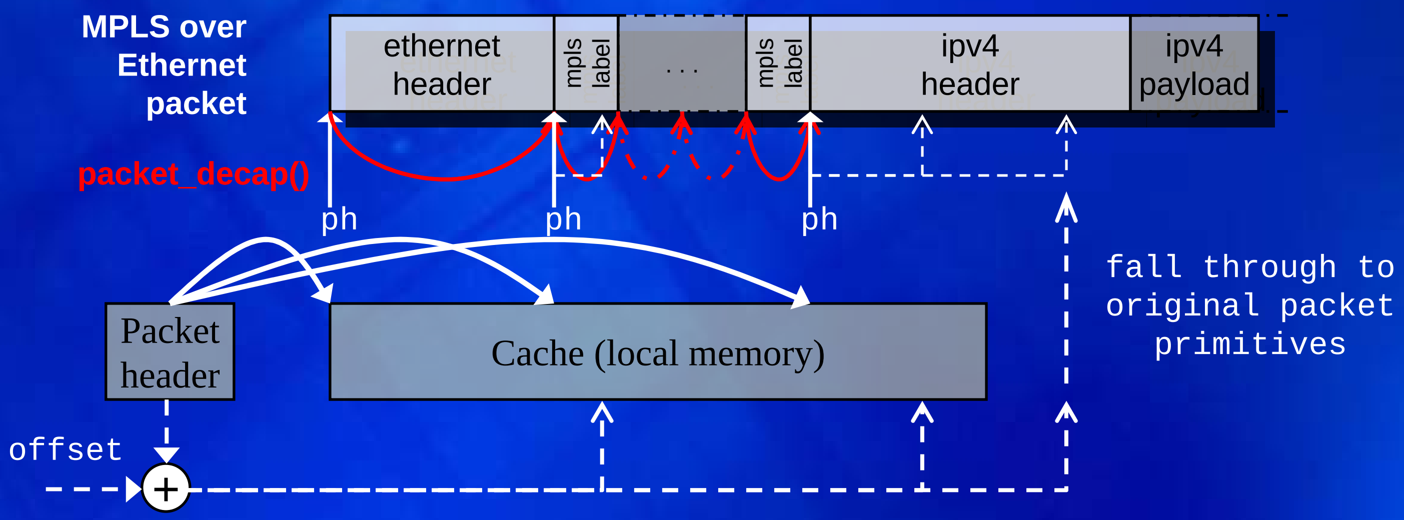
- **Assumptions**
  - HW can perform very wide memory accesses
  - Packet pointers are unique
- **Algorithm**
  - Select the best candidate to combine
  - Keep cached data in registers
  - Ensure data dependence by data flow analysis

Compiler-generated packet caching (static)



- Packet flow analysis
  - Inter-procedural and Inter-tasks analysis
  - Estimate the cache range
  - Annotate info onto packet primitives
- Code generator
  - Preload & write back cache
  - Generate efficient packet access code according to annotations
  - Remove unneeded packet primitives

Compiler-generated packet caching (dynamic)

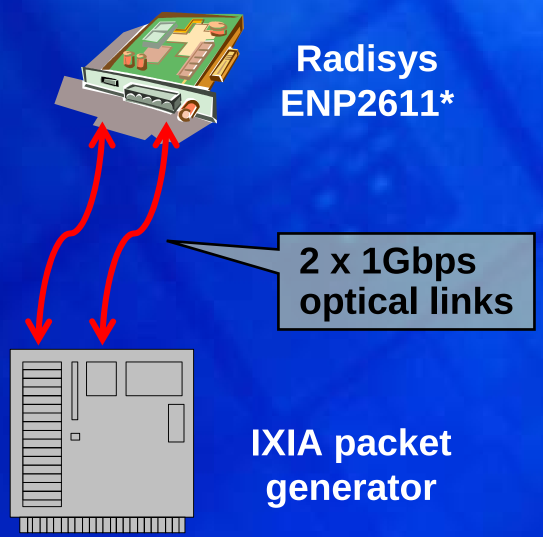


- **Packet primitives dynamically resolve field offsets and alignments**
- **Packet flow analysis**
  - **Estimates the cache range with profiling**
- **Code generator**
  - **Variable and run-time instructions to resolve offset and alignments dynamically**
  - **Run-time offset check to guarantee correctness**

Outline

- Motivation
- System overview
- Packet accesses optimizations
- Experimental results

Experimental setup

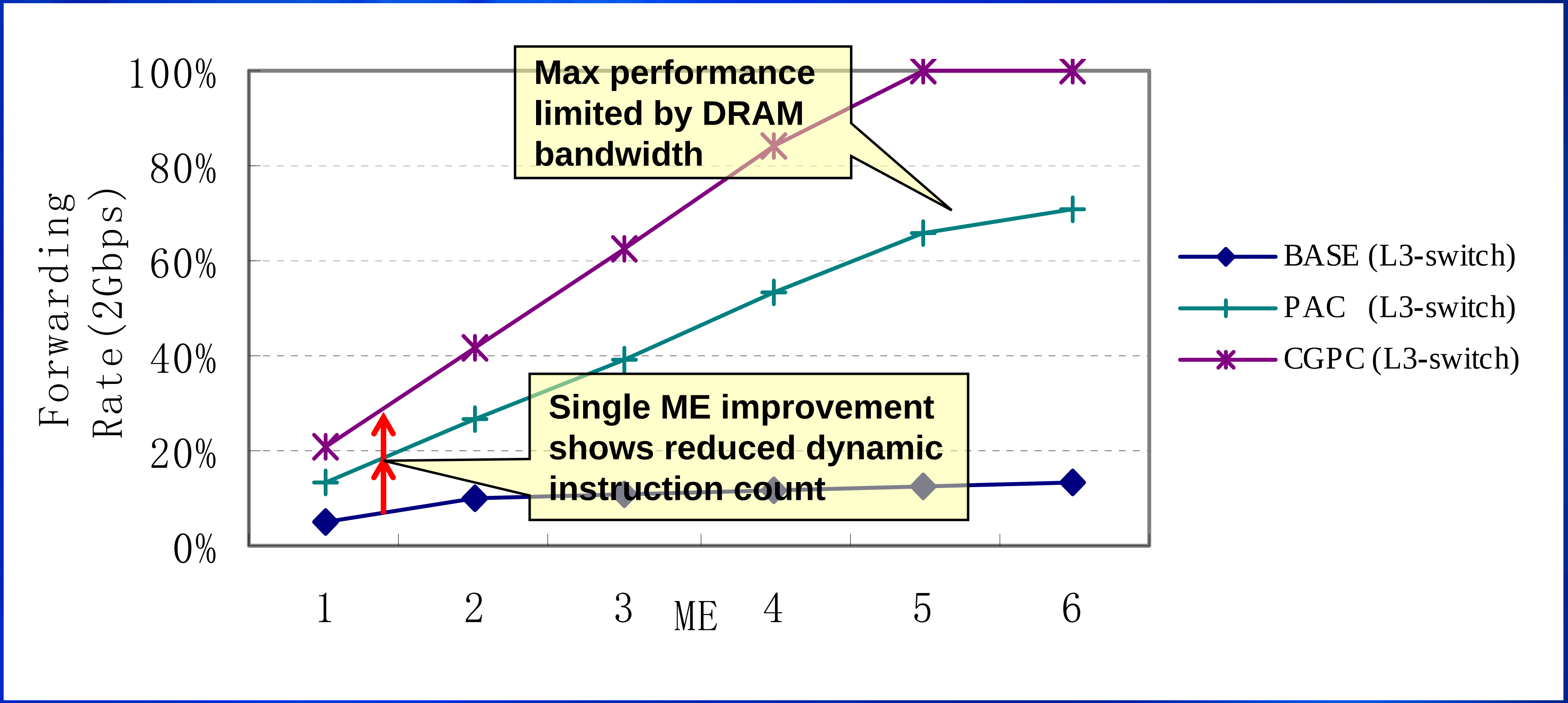


- Radisys ENP2611* board
  - IXP2400
  - 8MB SRAM, 64MB DRAM
  - 3 x 1Gbps optical ports
- IXP2400 runtime system
  - Linux on Intel XScale®
  - Language runtime system
- Benchmarks
  - L3-Switch† – L2 bridge & L3 routing using dest IP
  - MPLS† – Fast routing using label stack
  - Firewall – WAN / LAN isolation

† Evaluated using Network Processor Forum traces

* Third party brands/names property of their respective owners

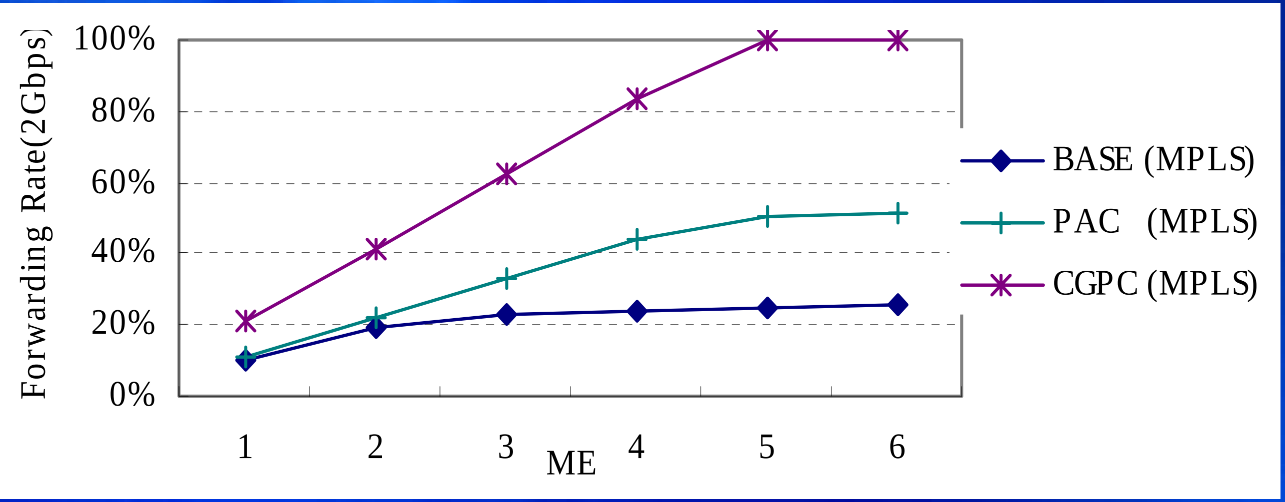
L3-Switch forwarding rate *

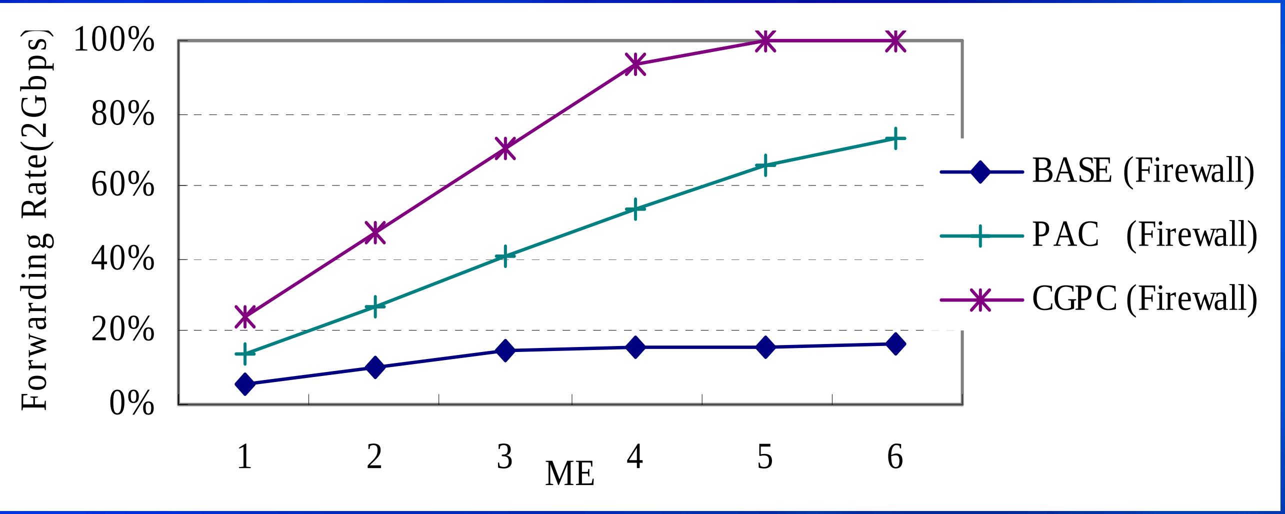


*** min-sized 64B packets**

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

MPLS & Firewall Performance





Packet access count and aggregate access size

		DRAM Access Count	Aggregate Access Size (Bytes)	Instruction Count
L3-Switch	**BASE**	29	696	2033
	PAC	13	200	1190
	CGPC	2	72	770
MPLS	**BASE**	16	384	1851
	PAC	9	212	1428
	CGPC	2	48	1495
Firewall	**BASE**	24.2	580	1742
	PAC	4.4	140	572
	CGPC	1	32	375

PAC & CGPC has approximate instruction count due to MPLS consisting of many dynamic packet accesses

PAC & CGPC significantly reduce packet DRAM accesses, aggregate memory access size and instruction count

Performance summary

- All benchmarks exhibit similar trends and performance curves
  - CGPC shows 5.8x performance speedup
- PAC & CGPC can efficiently reduce aggregate memory access size and instruction count
  - PAC: reduce 70% memory access size
  - CGPC: Reduce 90% memory access size
- CGPC is also effective to reduce dynamic packet accesses

Conclusions

- Packet access optimizations are critical to the performance of high-level programming environments
  - Performance is limited by instruction count and memory bandwidth
  - Efficiently relieve memory bandwidth contention and reduce instruction count
- PAC and CGPC are effective on performance improvement
  - Reduce aggregate memory access size and improve performance by 5.8x
  - With CGPC, achieve 2Gbps line rate on three typical network applications on IXP2400

Related work

- Shangri-la
  - Michael K. Chen, X. Li, R.Lian, J. Lin, L. Liu, T. Liu, R. Ju. Shangri-La: achieving high performance from compiled network applications while enabling ease of programming, In PLDI'05, Chicago, IL, June 2005
- Click
  - Kohler, E., Morris, R., Chen, B., Jannotti, J. and Kaashoek, M.F. The Click Modular Router. In ACM TCS, 18(3) pp. 263-297, August 2000.
- Memory access combining
  - Davidson, J. and Jinturkar, S. Memory Access Coalescing: A Technique for Eliminating Redundant Memory Accesses. In PLDI'94, Orlando, FL, June 1994.
- Packet buffer caching
  - S. Iyer, R.R. Kompella, and N. McKeown. Analysis of a memory architecture for fast packet buffers. In Proc. IEEE Workshop High Performance Switching and Routing(HPSR), 2001.

Q&A