# Network Processor: Architecture, Performance Evaluation and Applications

Yan Luo
Yan_Luo@uml.edu
http://faculty.uml.edu/yluo/

Electrical and Computer Engineering
University of Massachusetts Lowell

# Outline

- Network Processor Architecture
- NePSim Simulator
- Low Power Designs
- Content-Aware Switch

# Packet Processing in the Future Internet

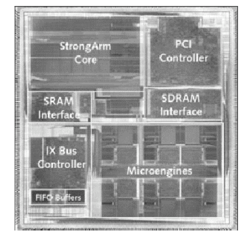
Future Internet

More packets
&
Complex packet processing

ASIC

+

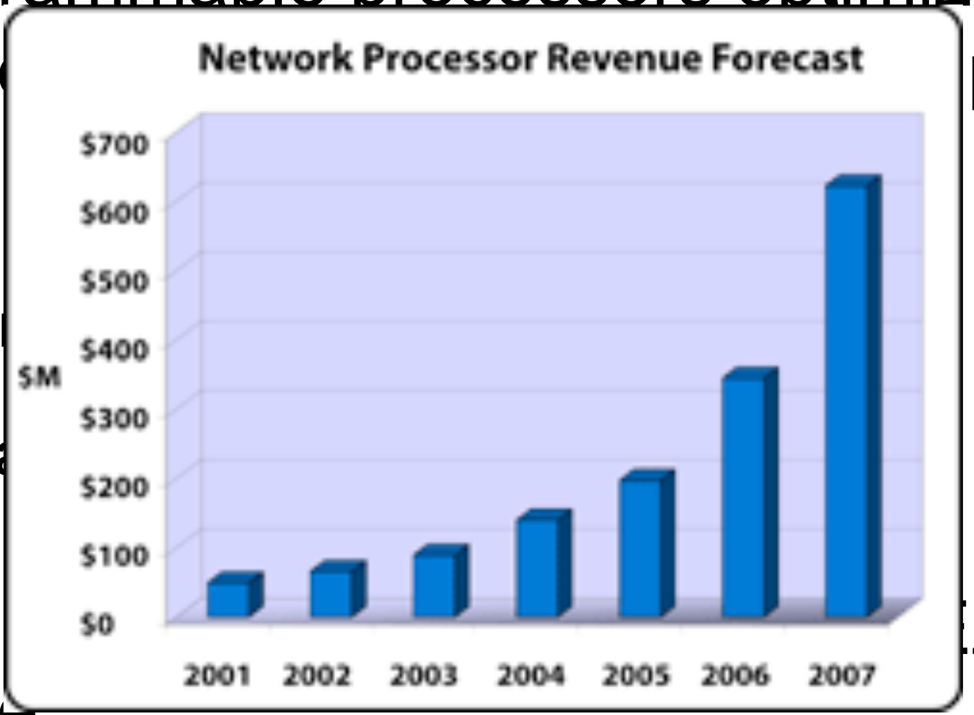
General-Purpose Processors




- High processing power
- Support wire speed
- Programmable
- Scalable
- Optimized for network applications
- …

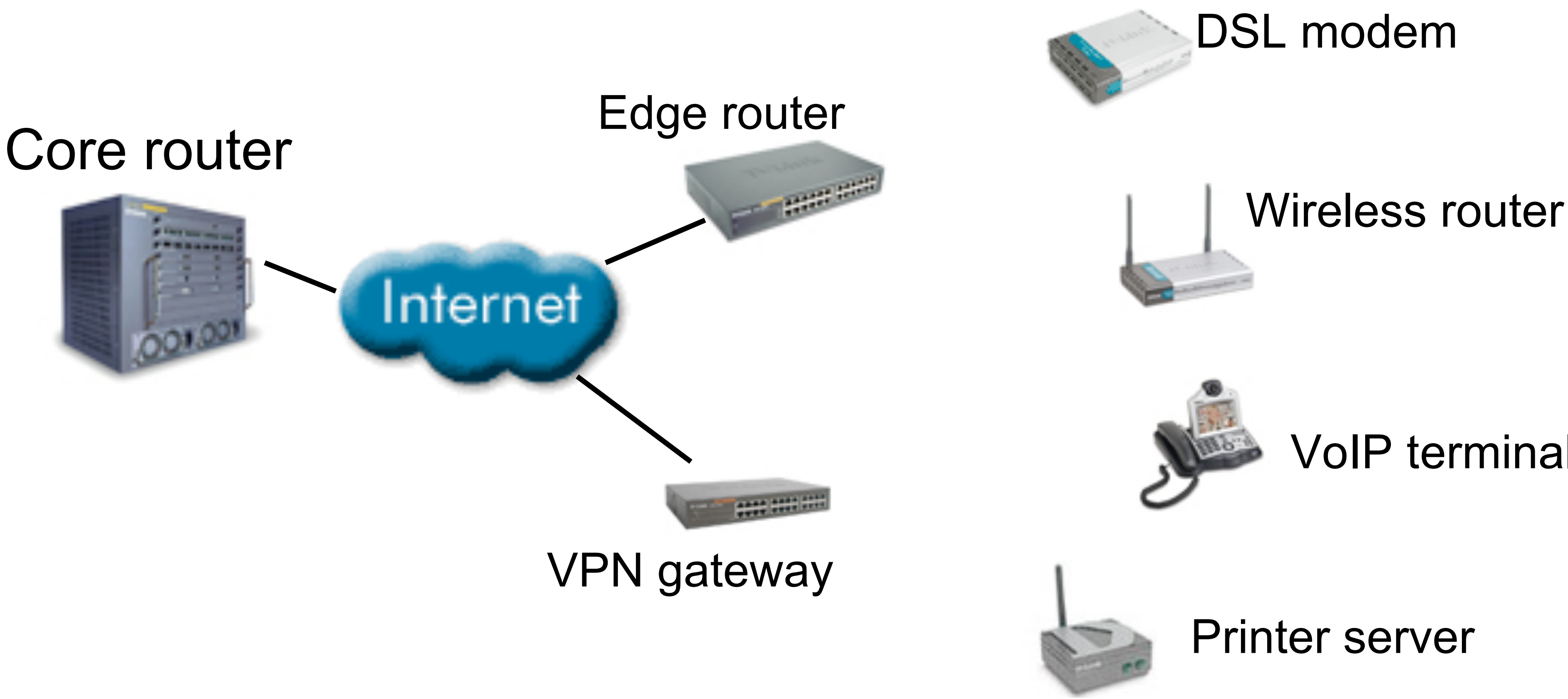
# What is Network Processor ?

- Programmable processors optimized for netw processing
  - H
  - P
  - F
- Main Ezchip, Agere




Semico Research Corp. Oct. 14, 2003

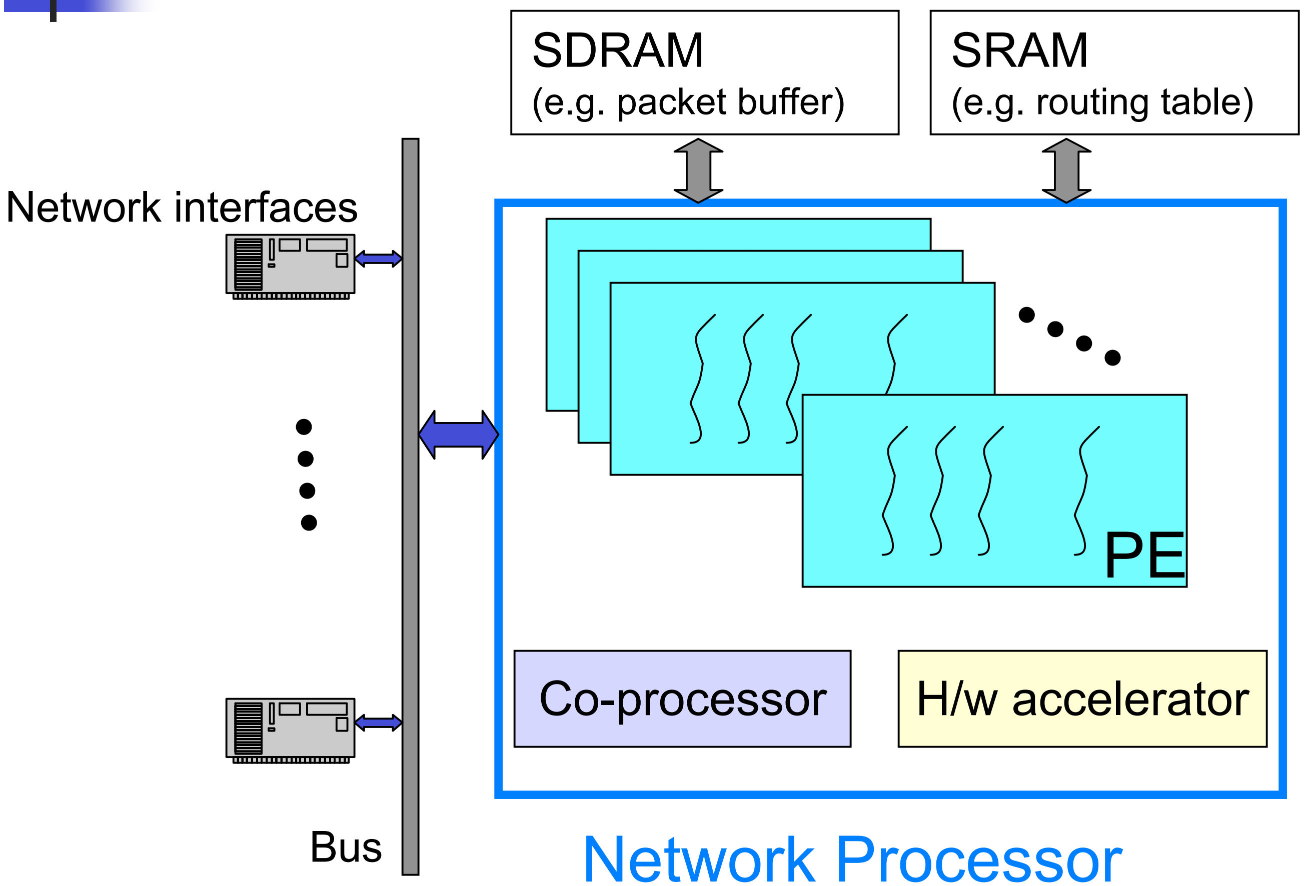
# Applications of Network Processors



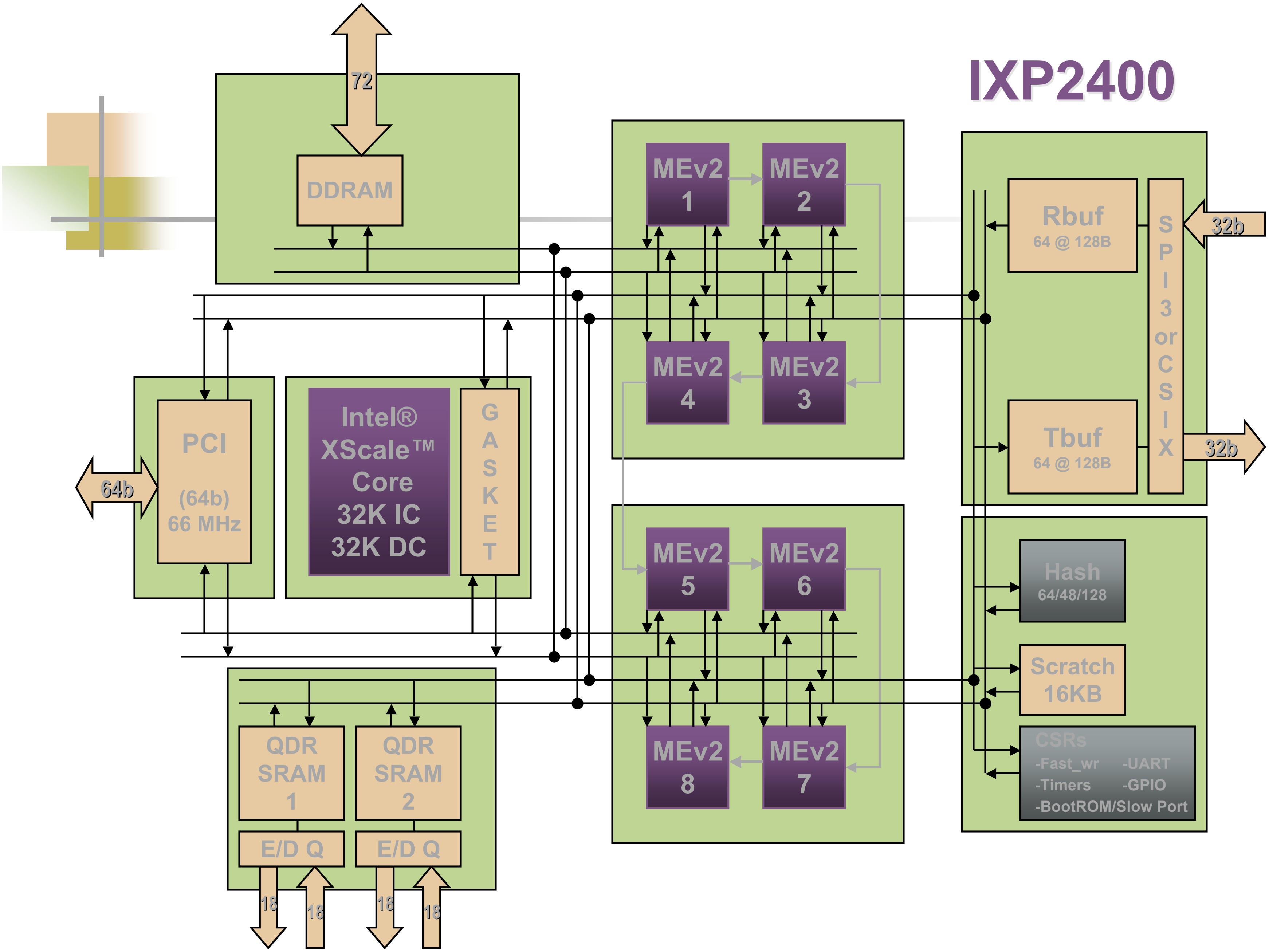
# Commercial Network Processors

| Vendor | Product | Line speed | Features |
| --- | --- | --- | --- |
| AMCC | nP7510 | OC-192/ 10 Gbps | Multi-core, customized ISA, multi-tasking |
| Intel | IXP2850 | OC-192/ 10 Gbps | Multi-core, h/w multi-threaded, coprocessor, h/w accelerators |
| Hifn | 5NP4G | OC-48/ 2.5 Gbps | Multi-threaded multiprocessor complex, h/w accelerators |
| EZchip | NP-2 | OC-192/ 10 Gbps | Classification engines, traffic managers |
| Agere | PayloadPlus | OC-192/ 10 Gbps | Multi-threaded, on-chip traffic management |

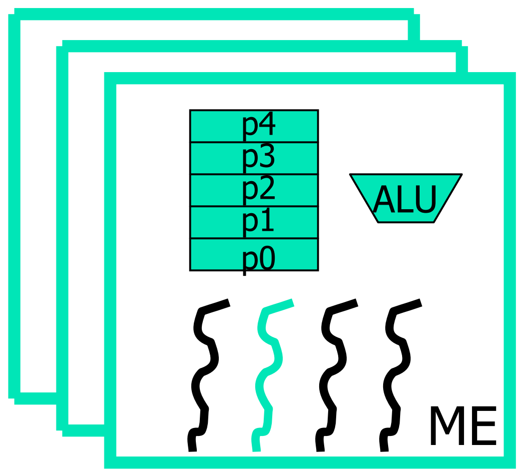
# Typical Network Processor Architecture



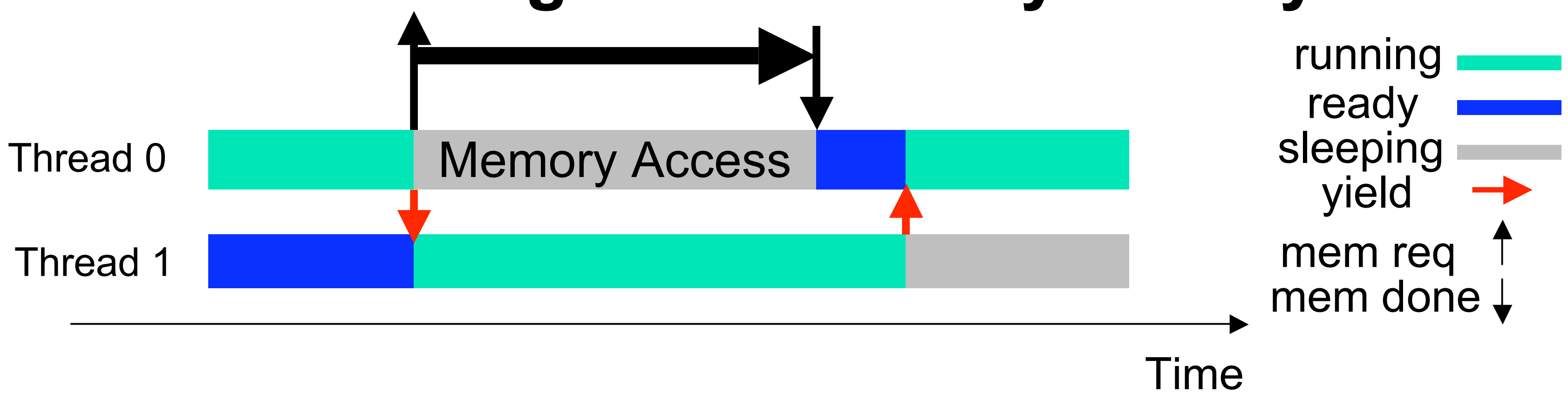
# IXP2400


72
DDRAM
MEv2 1
MEv2 2
MEv2 4
MEv2 3
MEv2 5
MEv2 6
MEv2 8
MEv2 7
Rbuf
64 @ 128B
Tbuf
64 @ 128B
SPI3 or CSIX
32b
32b
PCI
(64b)
66 MHz
64b
Intel® XScale™ Core
32K IC
32K DC
GASKET
Hash
64/48/128
Scratch
16KB
CSRs
-Fast_wr
-UART
-Timers
-GPIO
-BootROM/Slow Port
QDR SRAM 1
QDR SRAM 2
E/D Q
E/D Q
18
18
18
18

# Hardware Multithreading




**Multithreading hides memory latency**



# Network Processor Research Overview

- NPs have become popular and attracted more and more attention
- Performance has been the primary interest of the NP community
- Power consumption of NPs is becoming a big concern

•A ST200 edge router can support up to 8 NP boards each of which consumes 95~150W, The total power of such a 88.4cm x 44cm x 58cm router can reach **2700W** when two chasis are supported in a single rack! – *Laurel Networks ST series router data sheet*

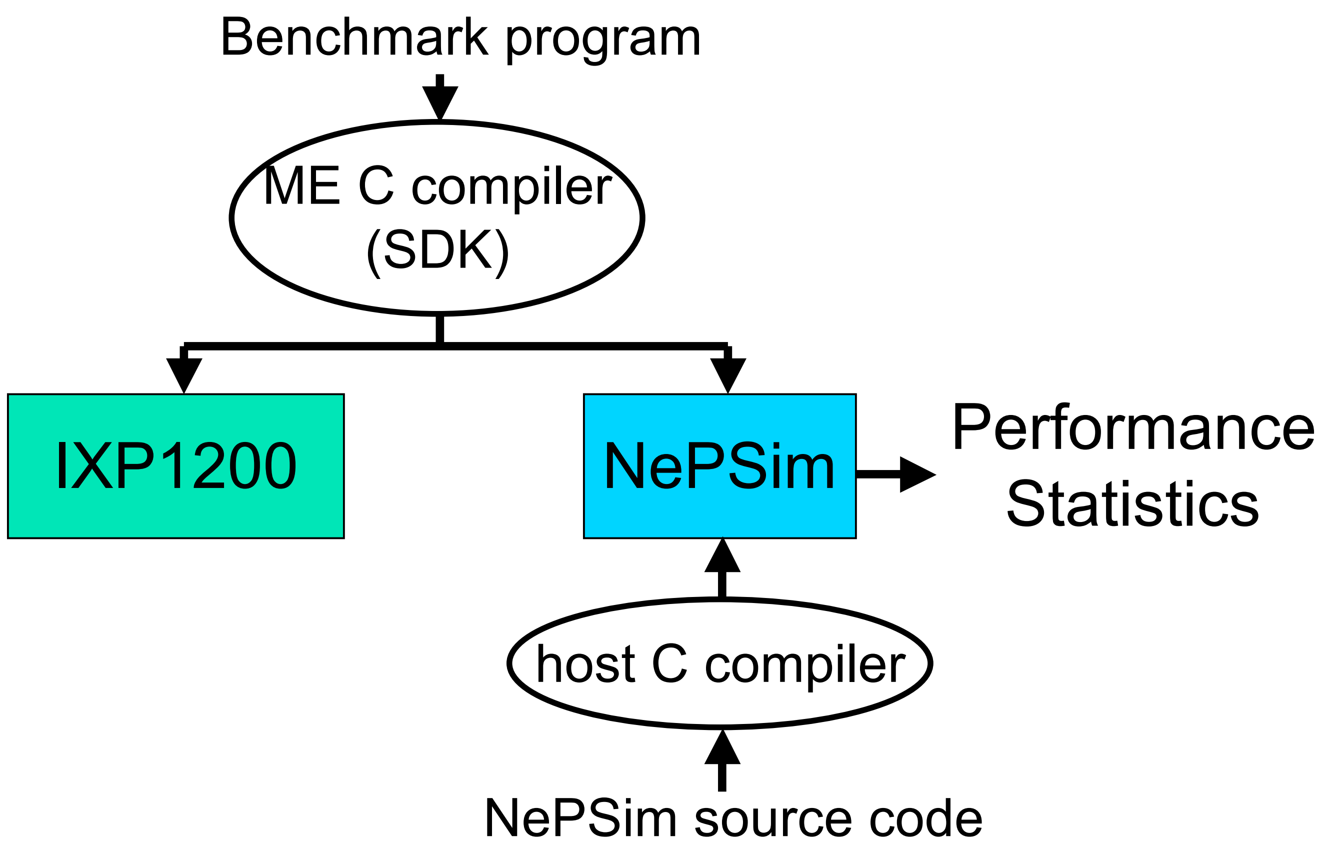
# NP Research Tools

- Intel IXA SDK
  - + accuracy, visualization
  - - close-source, low speed, inflexibility, no power model
- SimpleScalar
  - + open-source, popularity, power model (wattch)
  - - disparity with real NP, inaccuracy
- **NePSim**
  - **+ open-source, real NP, power model, accuracy**
  - - currently target IXP1200 only
  - IEEE Micro Special Issue on NP, Sept/Oct 2004, Intel IXP Summit, Sept 2004

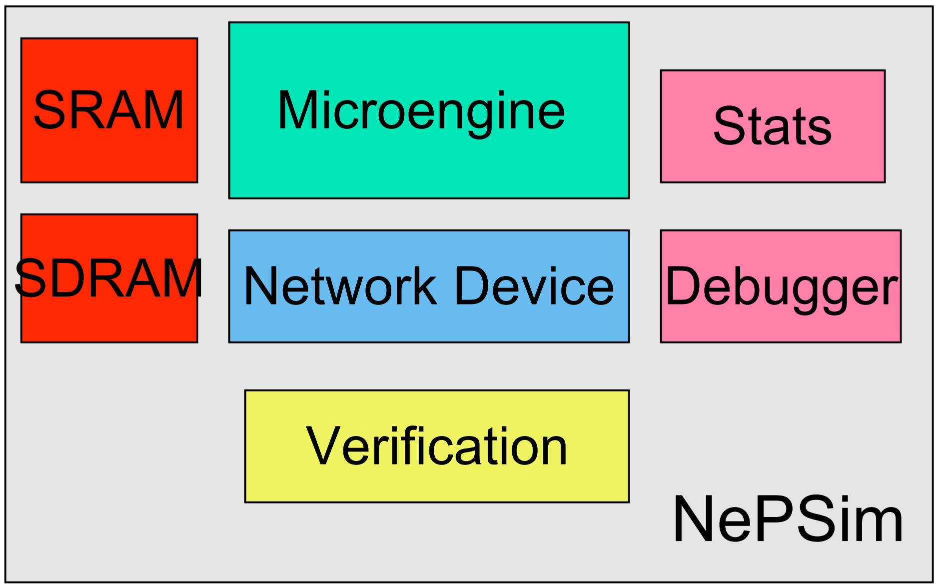
# Objectives of NePSim

- An open-source simulator for a real NP (Intel® IXP1200, later IXP2400/2800…)
- Cycle-level accuracy of performance simulation
- Flexibility for users to add new instructions and functional units
- Integrated power model to enable power dissipation simulation and optimization
- Extensibility for future NP architectures
- Fast simulation speed

# NePSim Overview

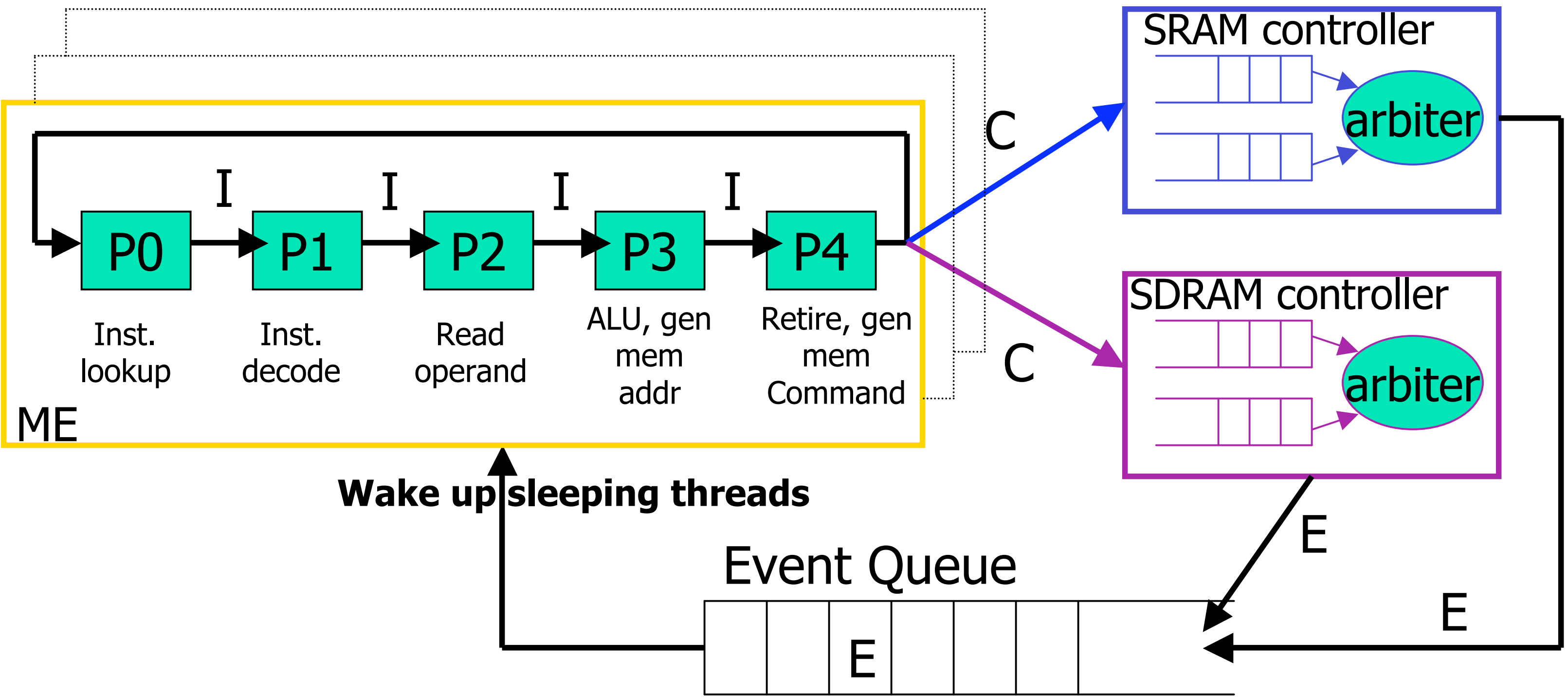


# NePSim Software Architecture



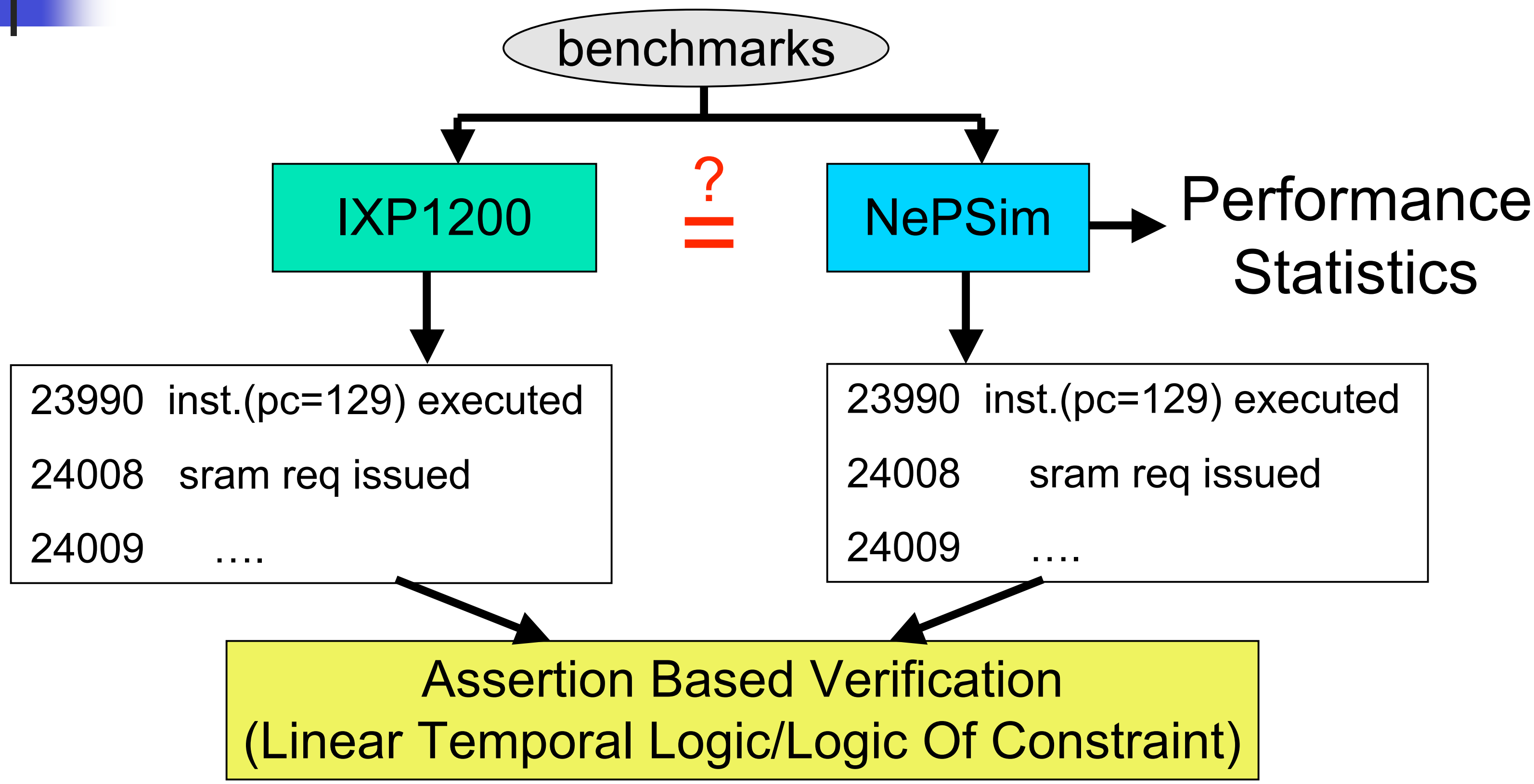

- Microengine (six)
- Memory (SRAM/SDRAM)
- Network Device
- Debugger
- Statistic
- Verification

# NePSim Internals




**I**: instruction
**C**: command
**E**: event

# Verification of NePSim




X. Chen, Y. Luo, H. Hsieh, L. Bhuyan, F. Balarin, "Utilizing Formal Assertions for System Design of Network Processors," *Design Automation and Test in Europe (DATE)*, 2004.

# Power Model

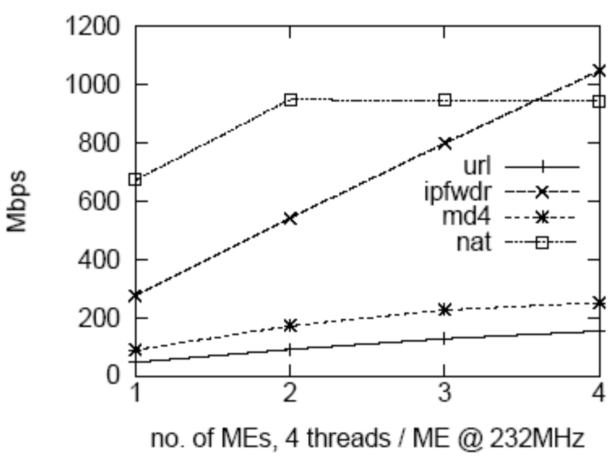
| H/W component | Model Type | Tool | Configurations |
|---|---|---|---|
| GPR per ME | Array | XCacti | 2 64-entry files, 1 read/write port per file |
| XFER per ME | Array | XCacti | 4 32-entry files, 1 read/write port per file |
| Control register per ME | Array | XCacti | 1 32-entry file, 1 read/write port |
| Control store, scratchpad | Cache w/o tag path | XCacti | 4KB, 4byte per block, direct mapped, 10-bit address |
| ALU , shifter | ALU and shifter | Wattch | 32bit |
| Command FIFO, command queue in controller, etc | Array | Wattch | See paper |
| Command bus arbiter, context arbiter | Matrix, rr arbiter | Orion | See paper |

# Benchmarks

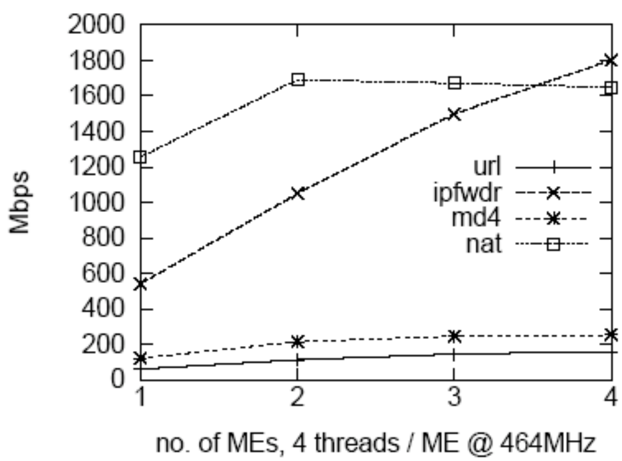
- Ipfwdr
  - IPv4 forwarding(header validation, trie-based lookup)
  - Medium SRAM access
- url
  - Examing payload for URL pattern, used in content-aware routing
  - Heavy SDRAM access
- Nat
  - Network address translation
  - medium SRAM acess
- Md4
  - Message digest (compute a 128-bit message “signature”)
  - Heavy computation and SDRAM access

# Performance implications

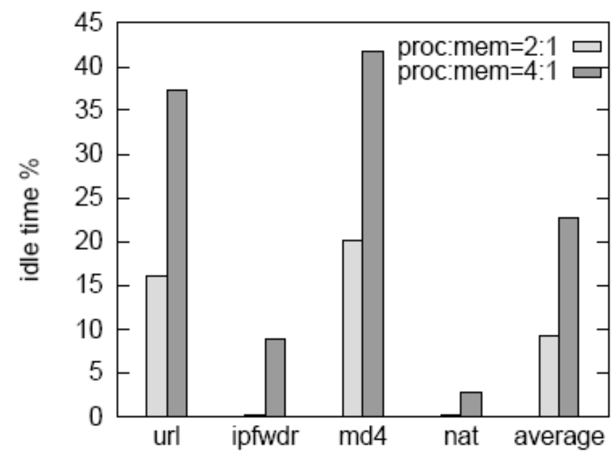
- More MEs do not necessarily bring performance gain
- More ME cause more memory contention
- ME idle time is abundant (up to 42%)
- Faster ME core results in more ME idle time with the same memory
- Non-optimal rcv/xmit configuration for NAT (transmitting ME is a bottleneck)




Throughput vs number of MEs at 232MHz



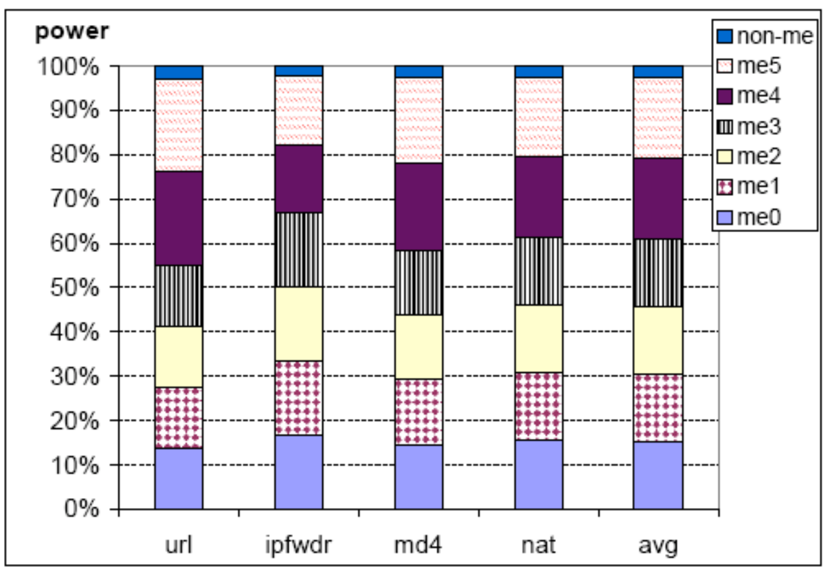

Throughput vs number of MEs at 464MHz

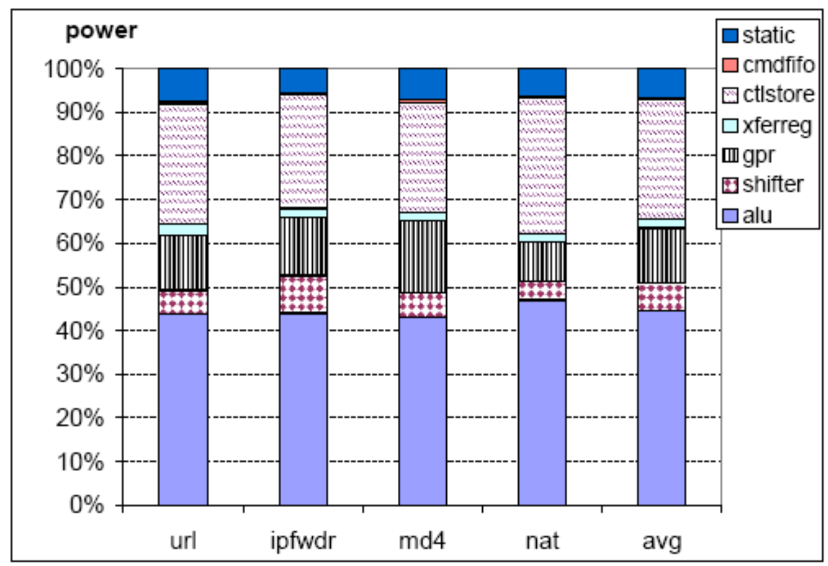



Idle time vs ME/memory speed ratio

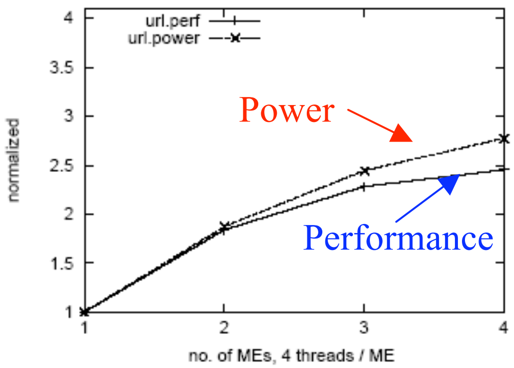
# Power breakdown

- Power dissipation by rcv and xmit MEs is similar across benchmarks
- Transmitting MEs consume ~5% more than receiving
- ALU consumes significant power ~45% (wattch model)
- Control store uses ~28% (accessed almost every cycle)
- GPRs burn ~13% , shifter ~7%, static ~7%

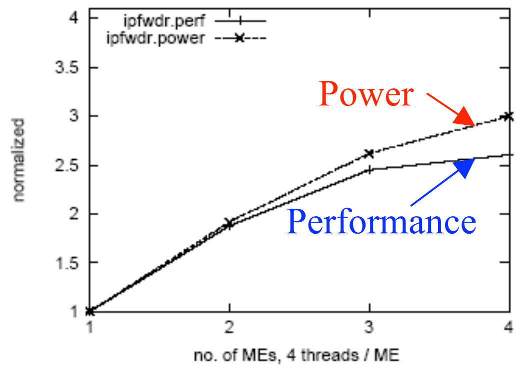


# Power efficiency observations



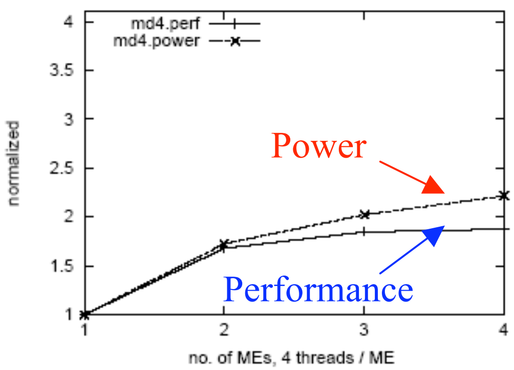

url



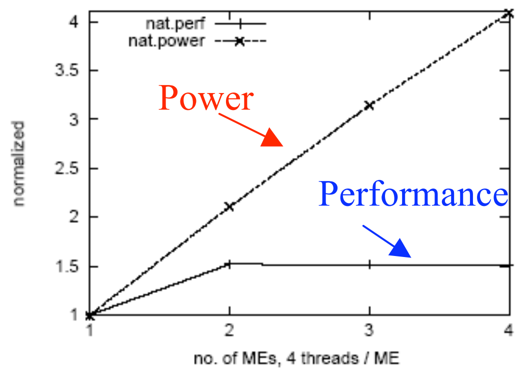

ipfwdr

- Power consumption increases faster than performance
- More MEs/threads bring more idle time due to memory contention

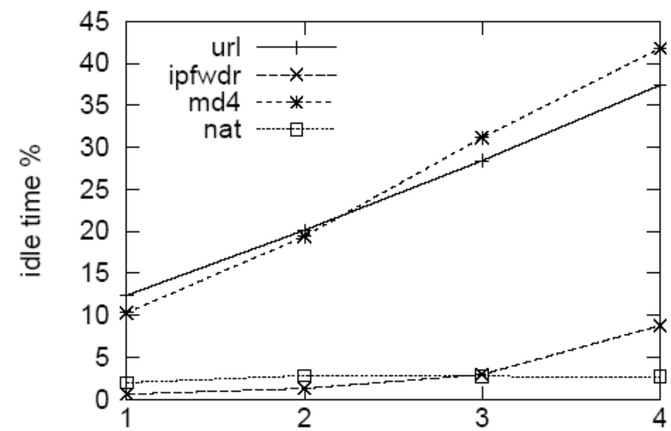
**We can reduce power consumption of MEs while they waiting for memory accesses**




md4

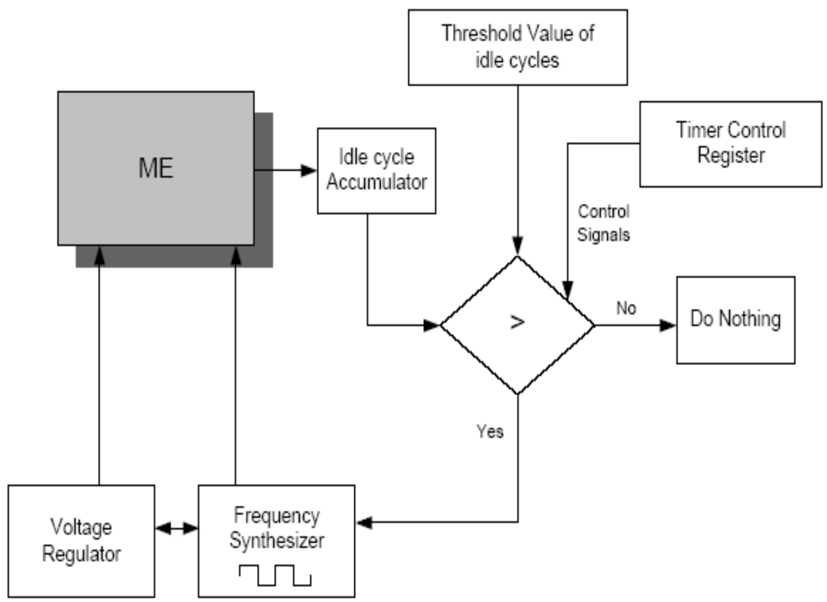



nat



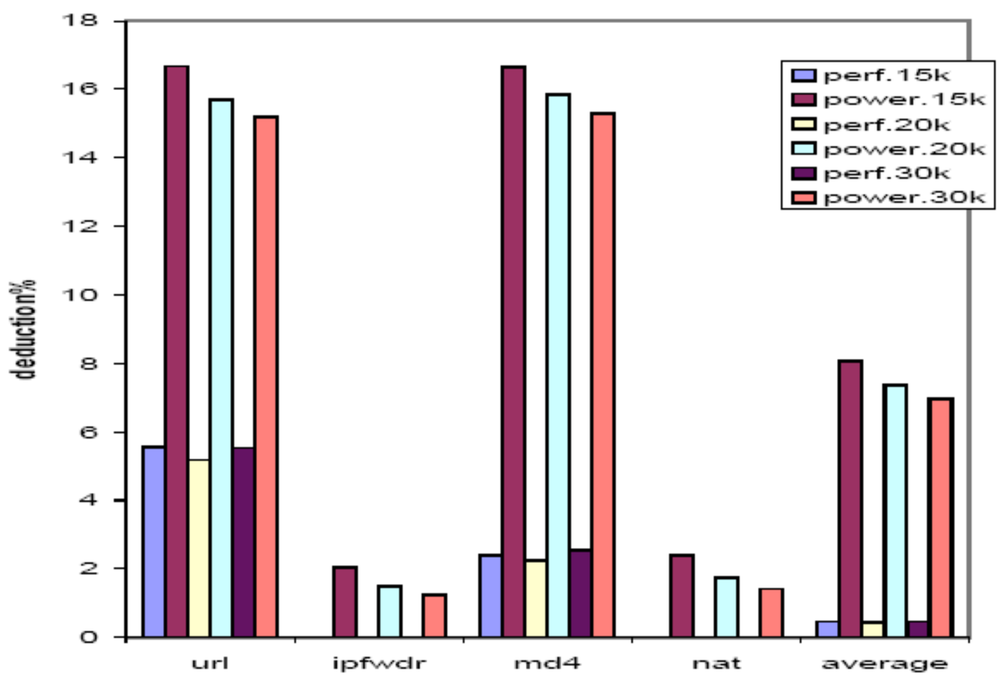

Idle time vs # of MEs

# Dynamic Voltage Scaling in NPs




- During the ME idle time, all threads are put to ``wait" state and the MEs are running with the lowest activity.
- Applying DVS while MEs are not very active can reduce the total power consumption substantially.
- DVS control scheme
  - Observes the ME idle time (%) periodically.
  - When idle > threshold, scale down the voltage and frequency (VF in short) by one step unless the minimum allowable VF is hit.
  - Idle < threshold, scale up the VF by one step unless they are at maximum allowable values.

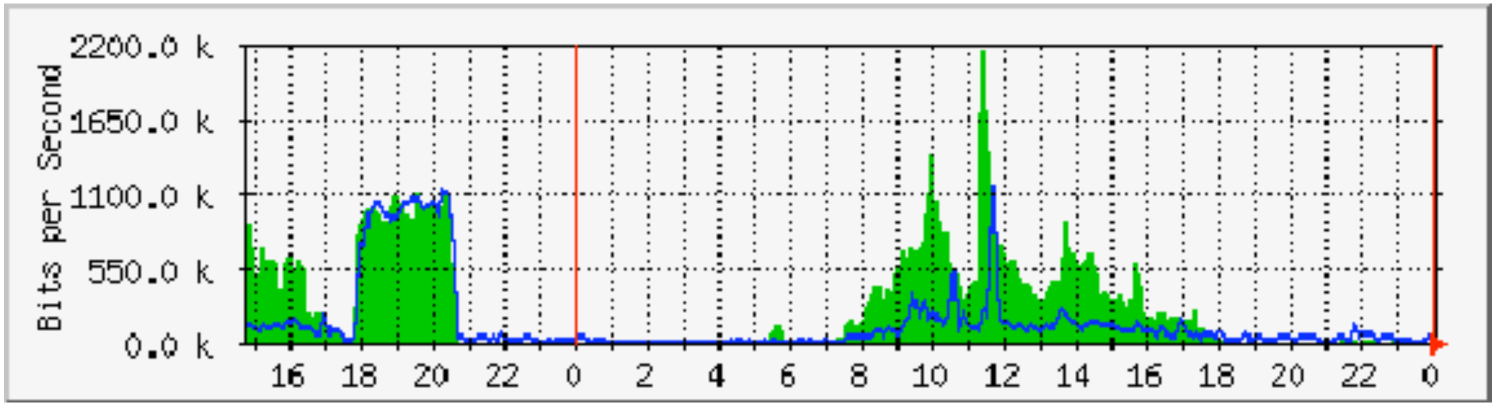
# DVS Power-performance




Power and performance reduction by DVS

- Initial VF=1.3V, 600MHz
- DVS period: every 15K, 20K or 30K cycles make a DVS decision to reduce or increase FV.
- Up to 17% power savings with less than 6% performance loss
- On average 8% power saving with <1% performance degradation

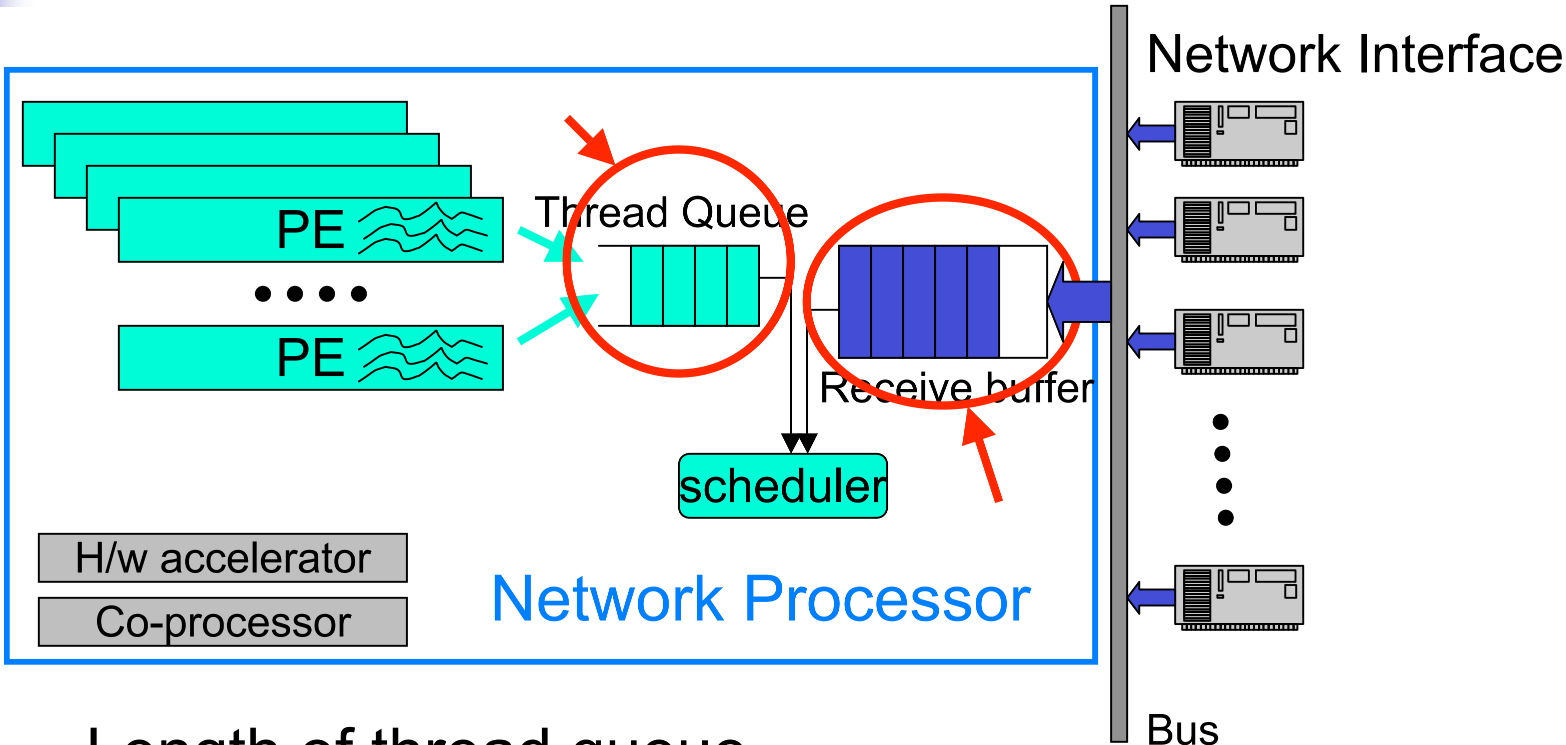
# Real-time Traffic Varies Greatly




- Shutdown unnecessary PEs, re-activate PEs when needed
- Clock gating retains PE instructions

Yan Luo, Jia Yu, Jun Yang, Laxmi Bhuyan, Low Power Network Processor Design Using Clock Gating, *IEEE/ACM Design Automation Conference (DAC), Anaheim, California, June 13-17, 2005*

# Indicators of Gating/Activating PEs




- Length of thread queue
- Fullness of internal buffers

# PE Shutdown Control Logic

If (thread_queue_length > T)

increment counter;

If (counter exceeds threshold)

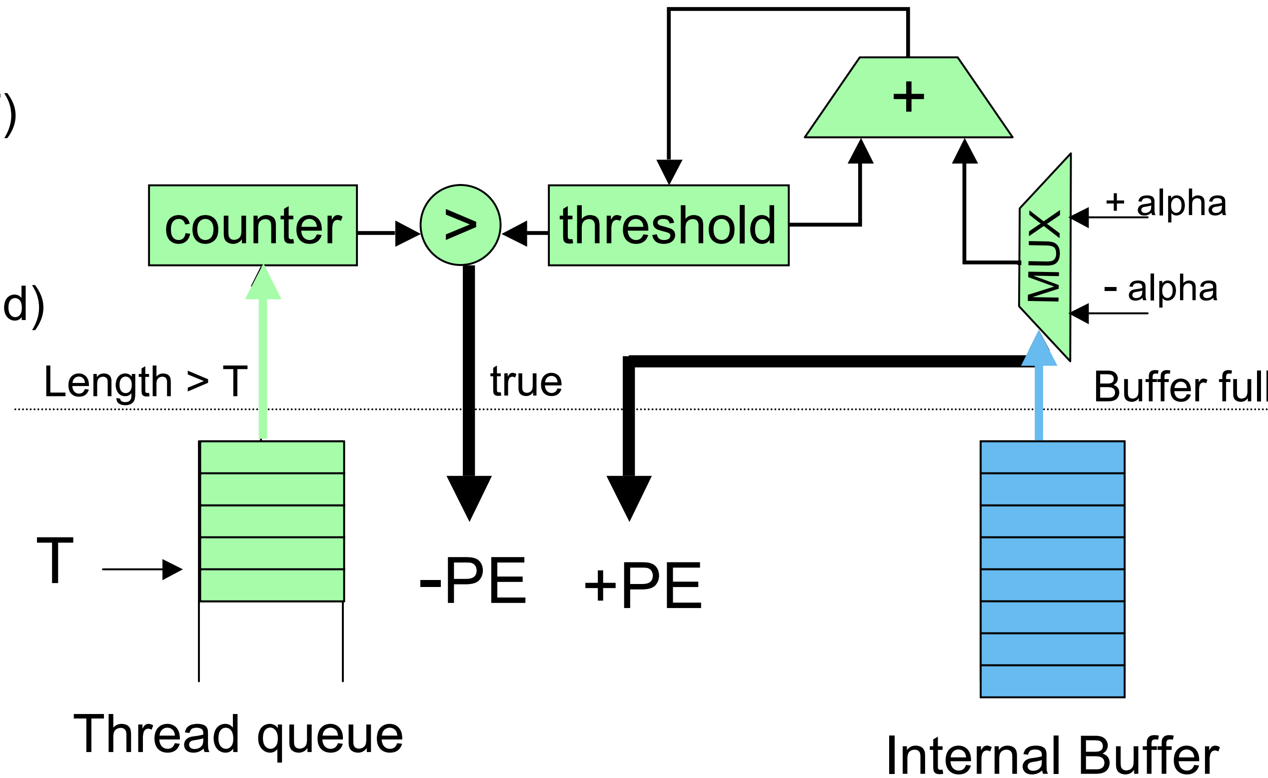
{ turn-off-a-PE;

decrement threshold }

If (buffer is full)

{ turn-on-a-PE;

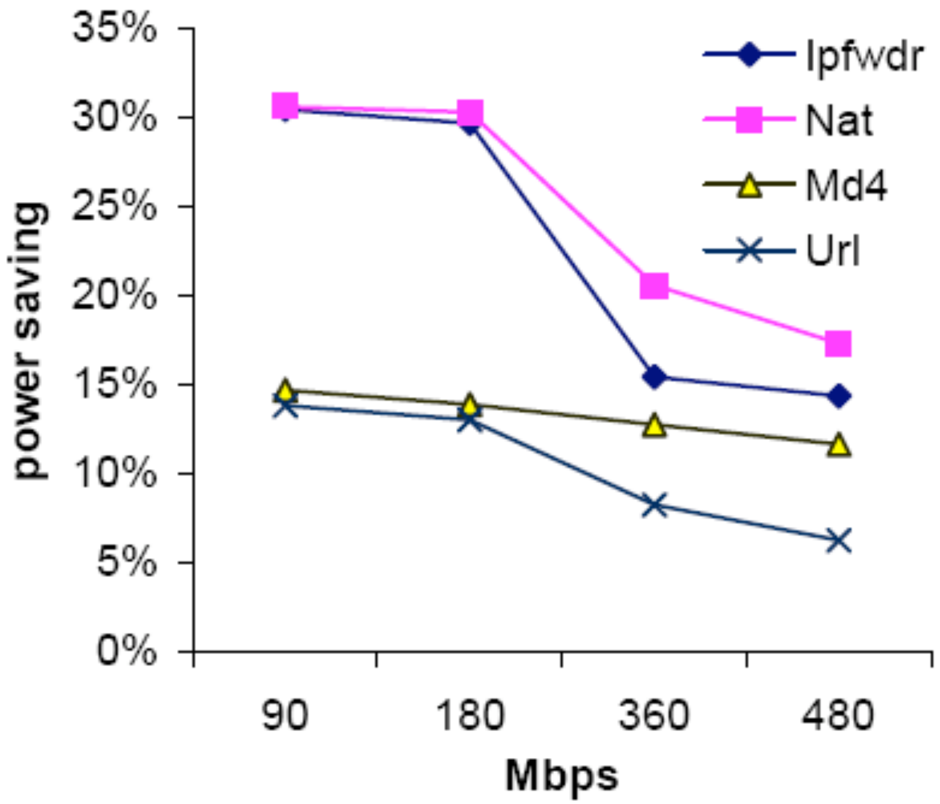
increment threshold }



# Challenges of Clock Gating PEs

- Terminating threads safely
  - Threads request memory resources
  - Stop unfinished threads result in resource leakage
- Reschedule packets to avoid “orphan” ports
  - Static thread-port mapping prohibits shutting down PEs
  - Dynamically assign packets to any waiting threads
- Avoid “extra” packet loss
  - Burst packet arrival can overflow internal buffer
  - Use a small extra buffer space to handle burst

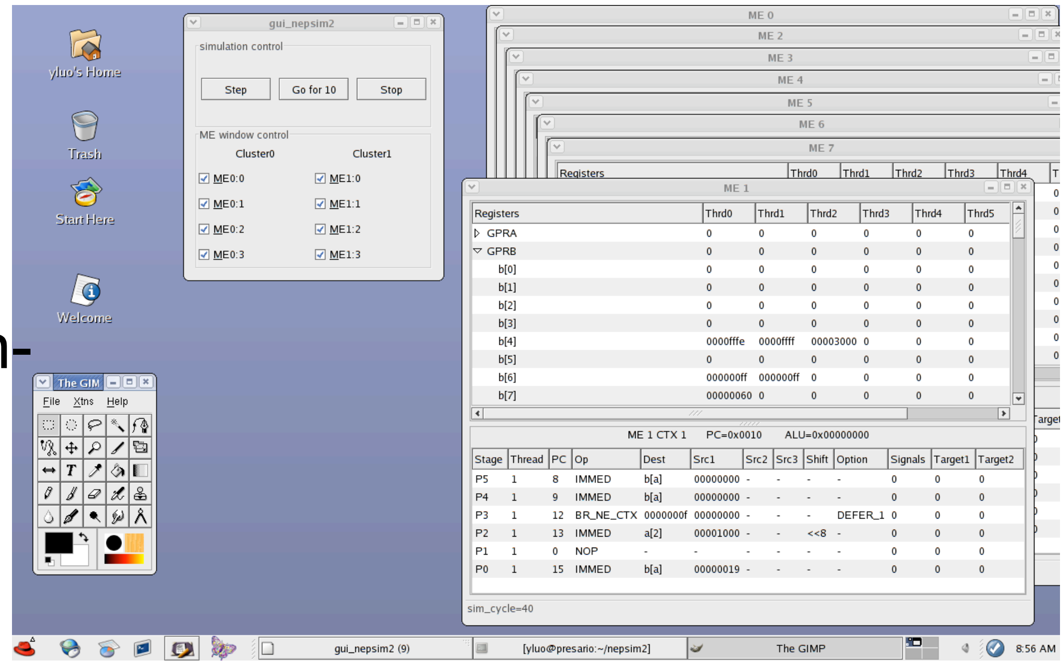
# Experiment Results of Clock Gating




<4% reduction on system throughput

# NePSim 2.0

- Extension of NePSim to model IXP2400/2800
- ME instruction set V. 2
- Modularized Network-On-Chip (bus, crossbar etc.)
- Power modeling of SRAM/DRAM
- Graphical user interface for debugging and monitoring



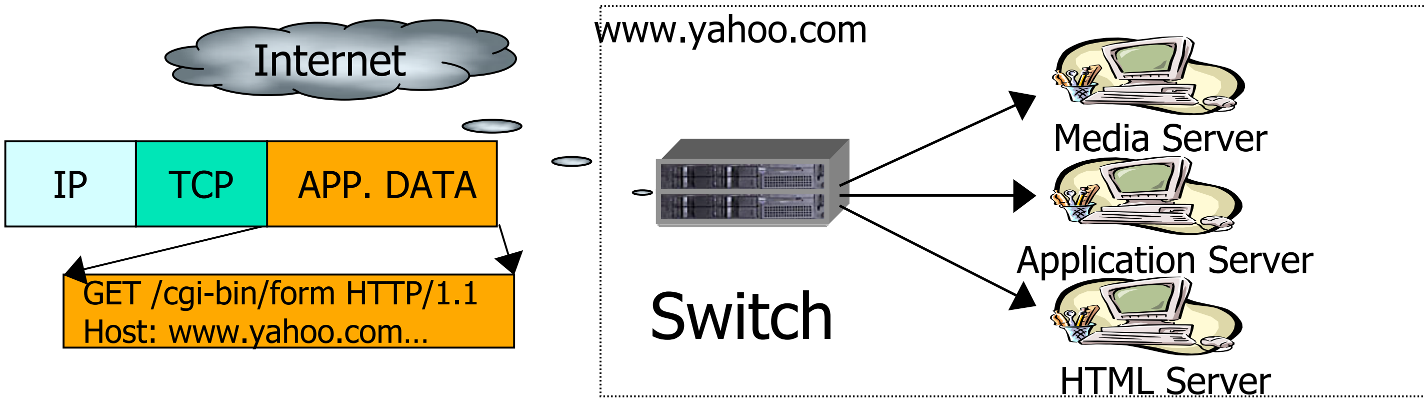
# Design and Implementation of A Content-aware Switch using A Network Processor

Li Zhao, Yan Luo, Laxmi Bhuyan
University of California, Riverside
Ravi Iyer
Intel Corporation

# Outline

- Motivation
- Background
- Design and Implementation
- Measurement Results
- Conclusions

# Content-aware Switch




- Front-end of a web cluster, one VIP
- Route packets based on layer 5 information
  - Examine application data in addition to IP& TCP
- Advantages over layer 4 switches
  - Better load balancing: distribute packets based on content type
  - Faster response: exploit cache affinity
  - Better resource utilization: partition database

# Processing Elements in Content-aware Switches

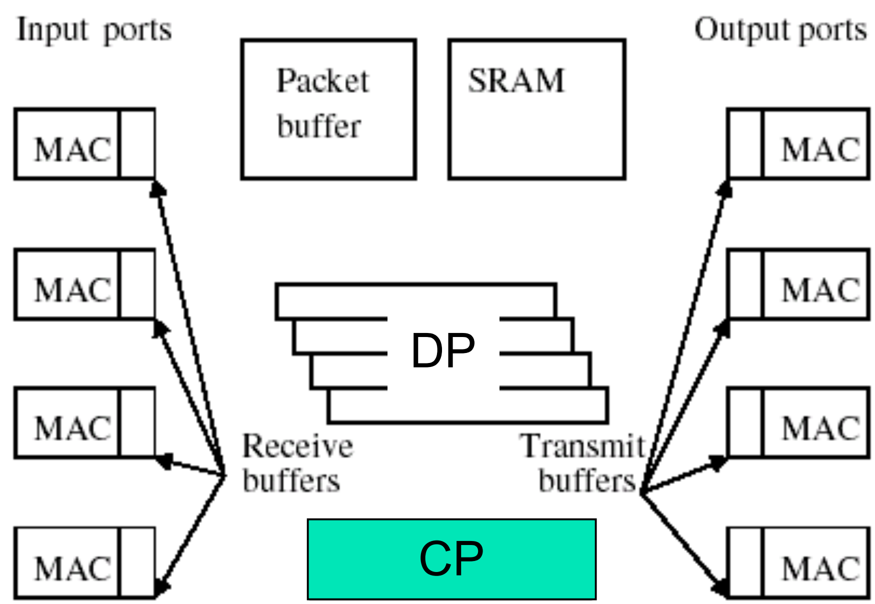
- ASIC (Application Specific Integrated Circuit)
  - High processing capacity
  - Long time to develop
  - Lack the flexibility
- GP (General-purpose Processor)
  - Programmable
  - Cannot provide satisfactory performance due to overheads on interrupt, moving packets through PCI bus, ISA not optimized for networking applications
- NP (Network Processor)
  - Operate at the link layer of the protocol, optimized ISA for packet processing, multiprocessing and multithreading → high performance
  - Programmable so that they can achieve flexibility

# Outline

- Motivation
- Background
  - NP architecture
  - Mechanism to build a content-aware switch
- Design and Implementation
- Measurement Results
- Conclusion

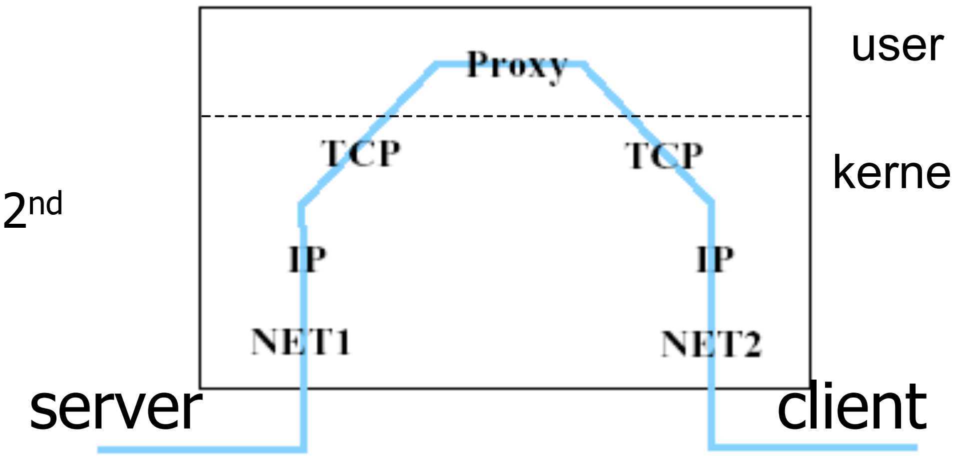
# Background on NP

- Hardware
  - Control processor (CP): embedded general purpose processor, maintain control information
  - Data processors (DPs): tuned specifically for packet processing
  - Communicate through shared DRAM
- NP operation on packets
  - Packet arrives in receive buffer
  - Header Processing
  - Transfer the packet to transmit buffer

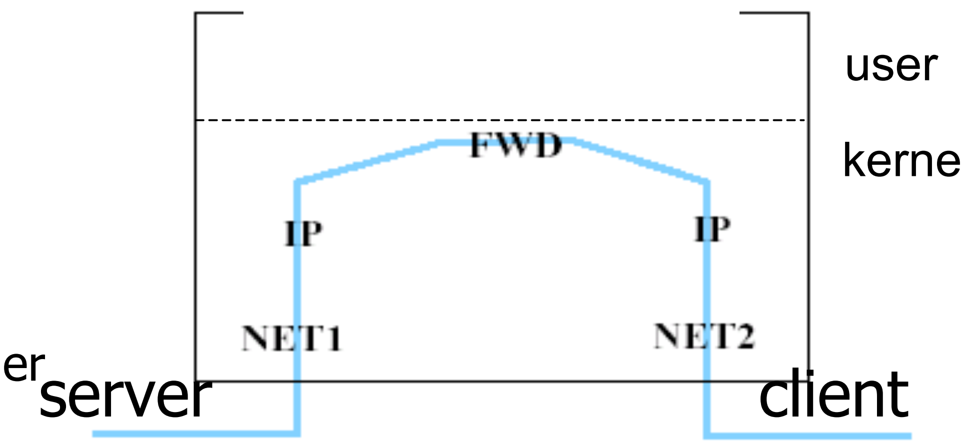


# Mechanisms to Build a CA Switch

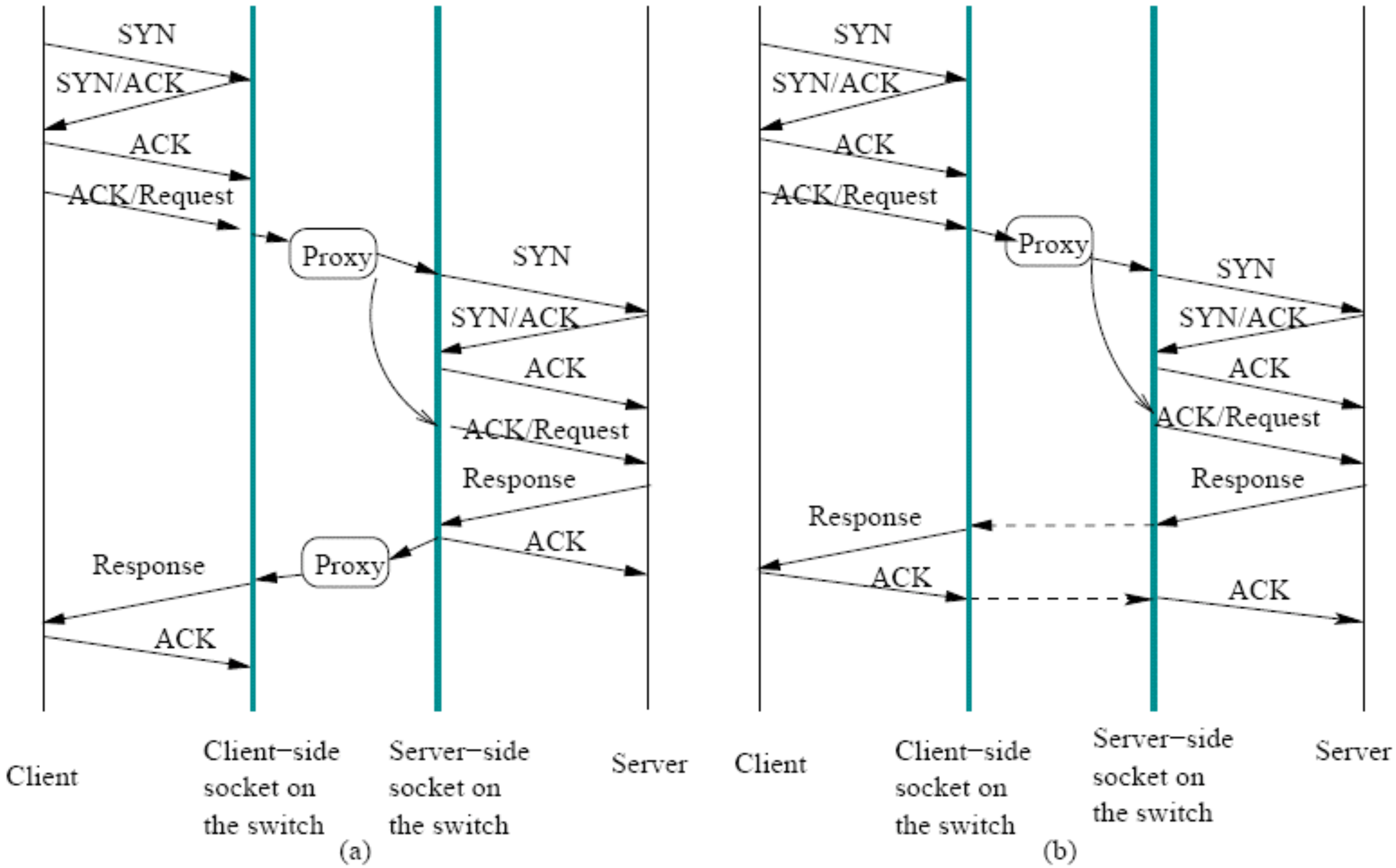
- TCP gateway
  - An application level proxy
  - Setup 1st connection w/ client, parses request →server, setup 2nd connection w/ server
  - Copy overhead




- TCP splicing
  - Reduce the copy overhead
  - Forward packet at network level between the network interface driver and the TCP/IP stack
  - Two connections are spliced together
  - Modify fields in IP and TCP header



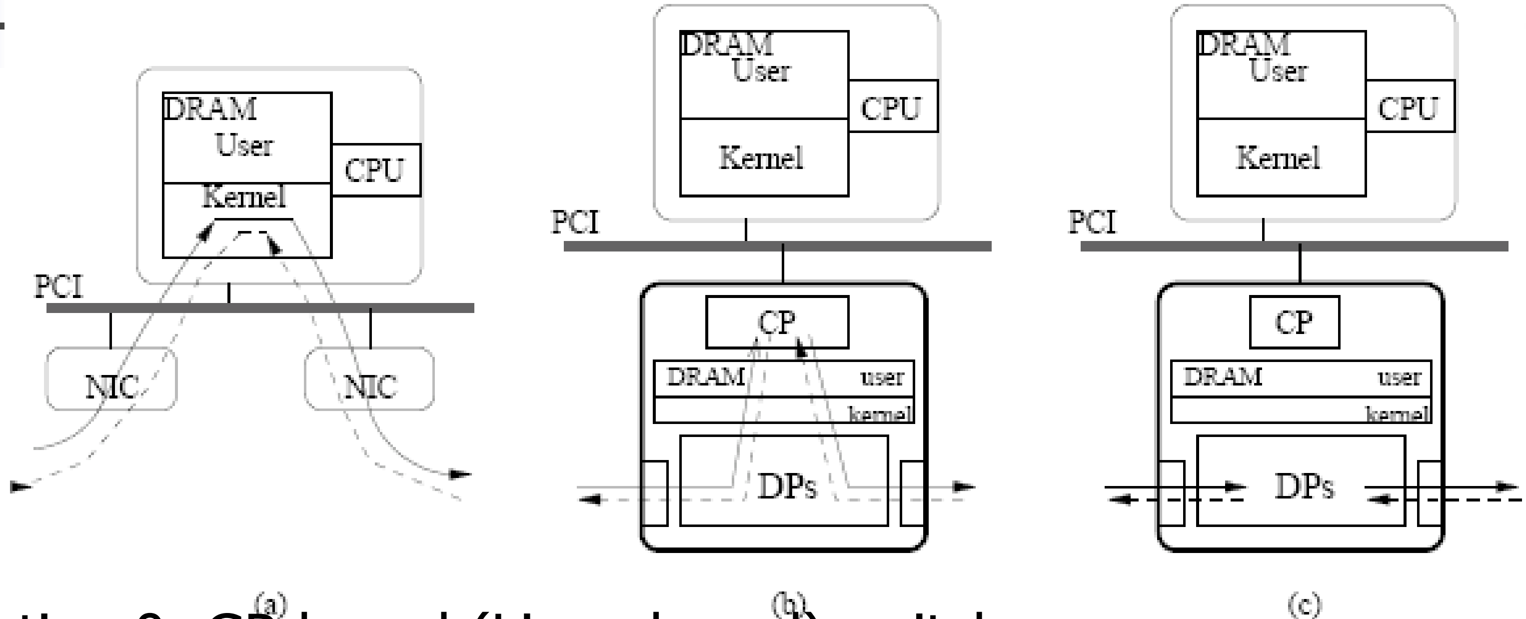
# Operations on a Content-Aware Switch



# Outline

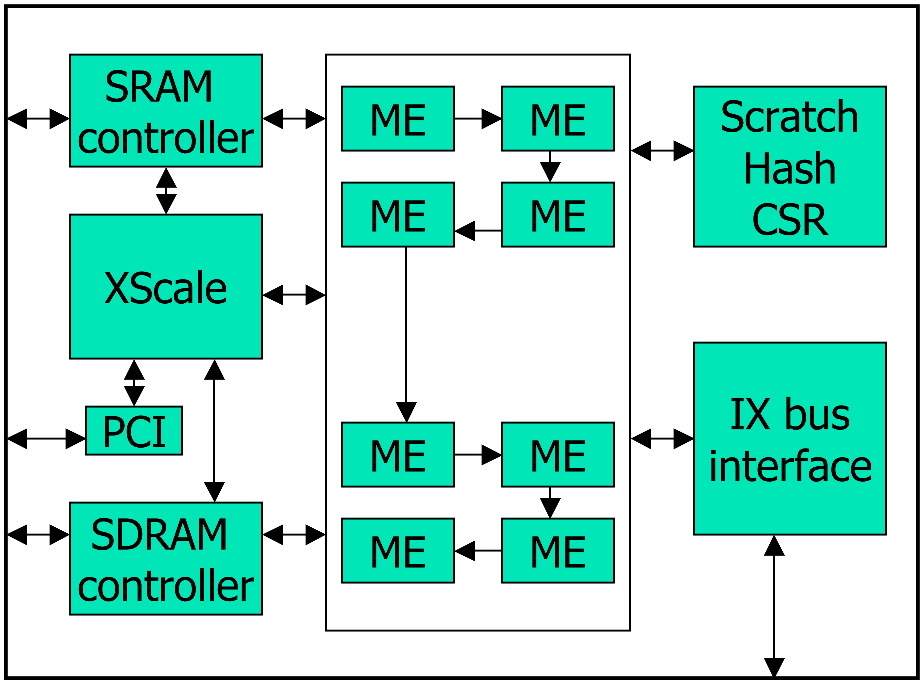
- Motivation
- Background
- Design and Implementation
  - Discussion on design options
  - Resource allocation
  - Processing on MEs
- Measurement Results
- Conclusion

# Design Options




- Option 0: GP-based (Linux-based) switch
- Option 1: CP setup & and splices connections, DPs process packets sent after splicing
  - Connection setup & splicing is more complex than data forwarding
  - Packets before splicing need to be passed through DRAM queues
- Option 2: DPs handle connection setup, splicing & forwarding

# IXP 2400 Block Diagram

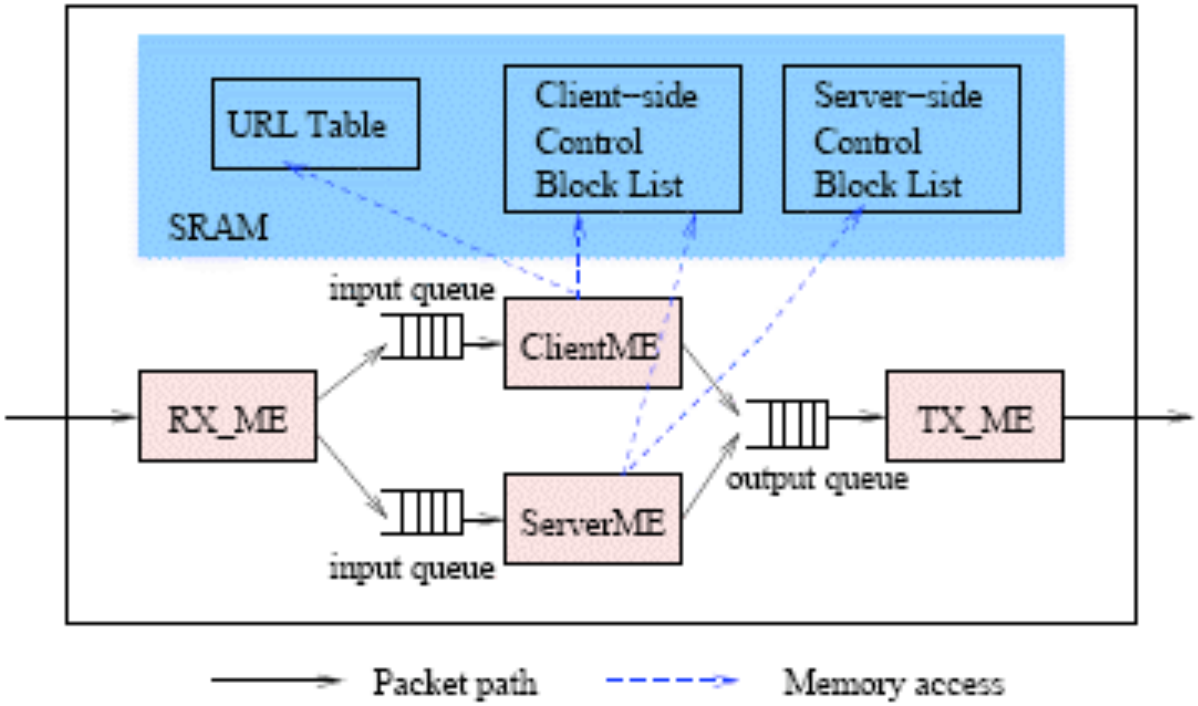



- XScale core
- Microengines(MEs)
  - 2 clusters of 4 microengines each
- Each ME
  - run up to 8 threads
  - 16KB instruction store
  - Local memory
- Scratchpad memory, SRAM & DRAM controllers

# Resource Allocation

Client port

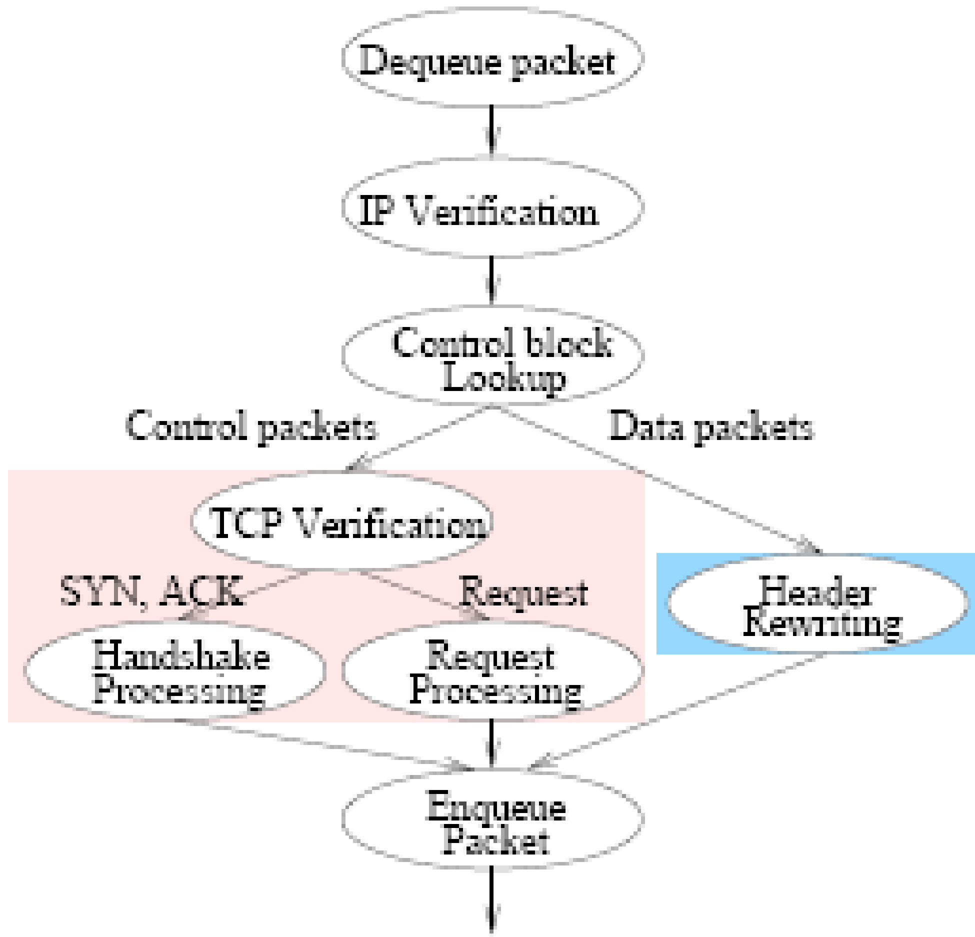
Server port




- Client-side control block list: record states for connections between clients and switch, states for forwarding data packets after splicing
- Server-side control block list: record state for connections between server and switch
- URL table: select a back-end server for an incoming request

# Processing on MEs

- Control packets
  - SYN
  - HTTP request
- Data packets
  - Response
  - ACK



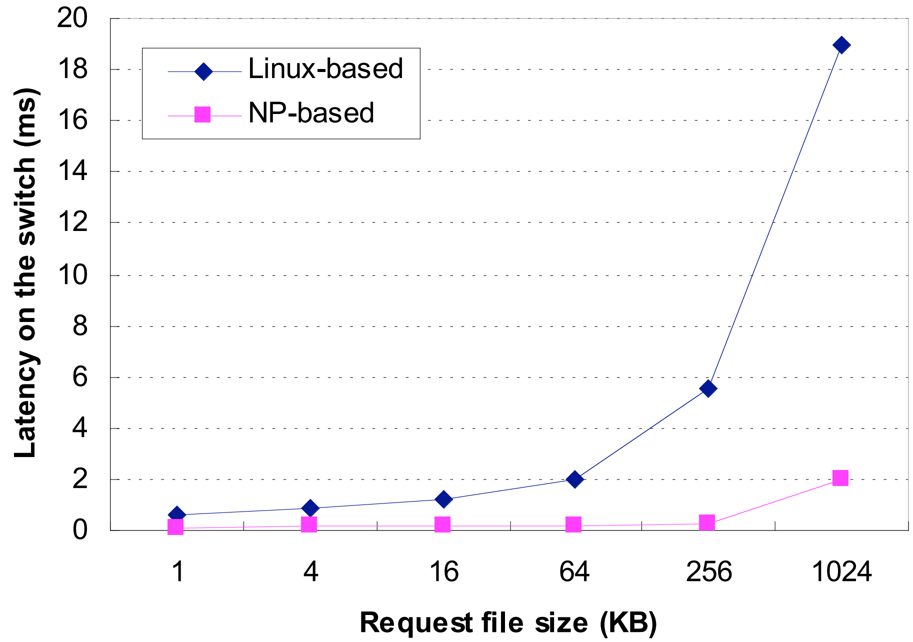
# Outline

- Motivation
- Background
- Design and Implementation
- Measurement Results
- Conclusion

# Experimental Setup

- Radisys ENP2611 containing an IXP2400
  - XScale & ME: 600MHz
  - 8MB SRAM and 128MB DRAM
  - Three 1Gbps Ethernet ports: 1 for Client port and 2 for Server ports
- Server: Apache web server on an Intel 3.0GHz Xeon processor
- Client: Httperf on a 2.5GHz Intel P4 processor
- Linux-based switch
  - Loadable kernel module
  - 2.5GHz P4, two 1Gbps Ethernet NICs

# Measurement Results

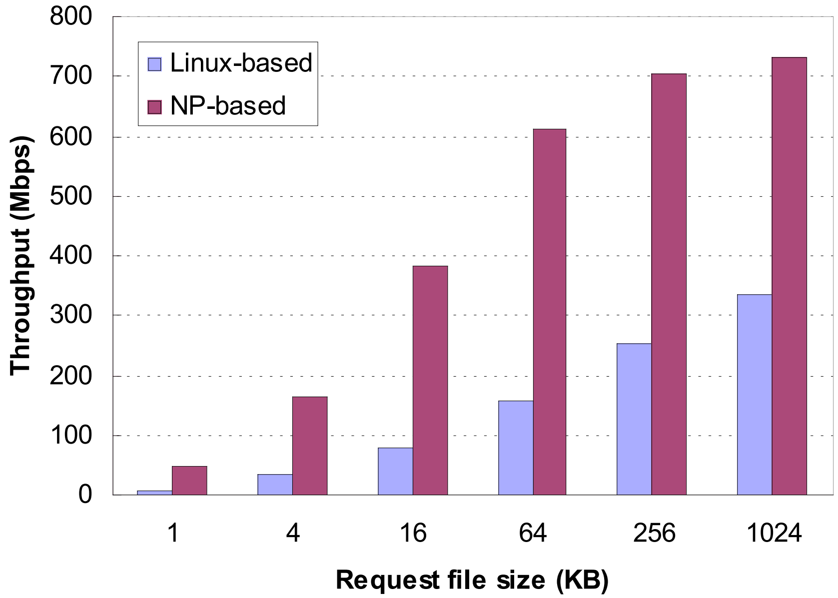



- Latency reduced significantly
  - 83.3% (0.6ms → 0.1ms) @ 1KB
- The larger the file size, the higher the reduction
  - 89.5% @ 1MB file

# Analysis – Three Factors

| Linux-based | NP-based |
|---|---|
| Interrupt: NIC raises an interrupt once a packet comes | polling |
| NIC-to-mem copy<br>Xeon 3.0Ghz Dual processor w/ 1Gbps Intel Pro 1000 (88544GC) NIC, 3 us to copy a 64-byte packet by DMA | No copy: Packets are processed inside w/o two copies |
| Linux processing: OS overheads<br>Processing a data packet in splicing state: 13.6 us | IXP processing: Optimized ISA<br>6.5 us |

# Measurement Results

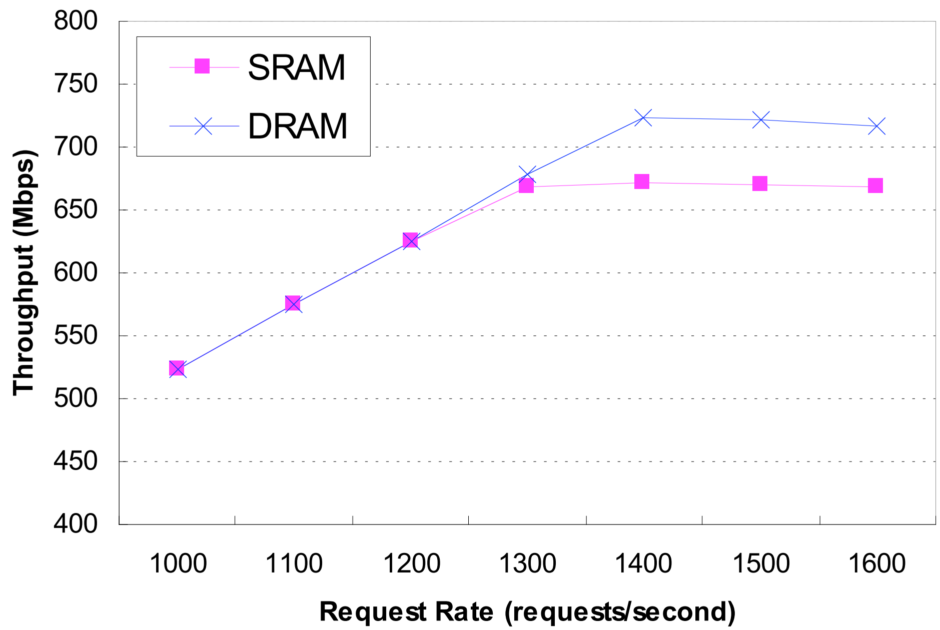



- Throughput is increased significantly
  - 5.7x for small file size @ 1KB, 2.2x for large file @ 1MB
- Higher improvement for small files
  - Latency reduction for control packets > data packets
  - Control packets take a larger portion for small files

# An Alternative Implementation

- SRAM: control blocks, hash tables, locks
  - Can become a bottleneck when thousands of connections are processed simultaneously; Not possible to maintain a large number due to its size limitation
- DRAM: control blocks, SRAM: hash table and locks
  - Memory accesses can be distributed more evenly to SRAM and DRAM, their access can be pipelined; increase the # of control blocks that can be supported

# Measurement Results



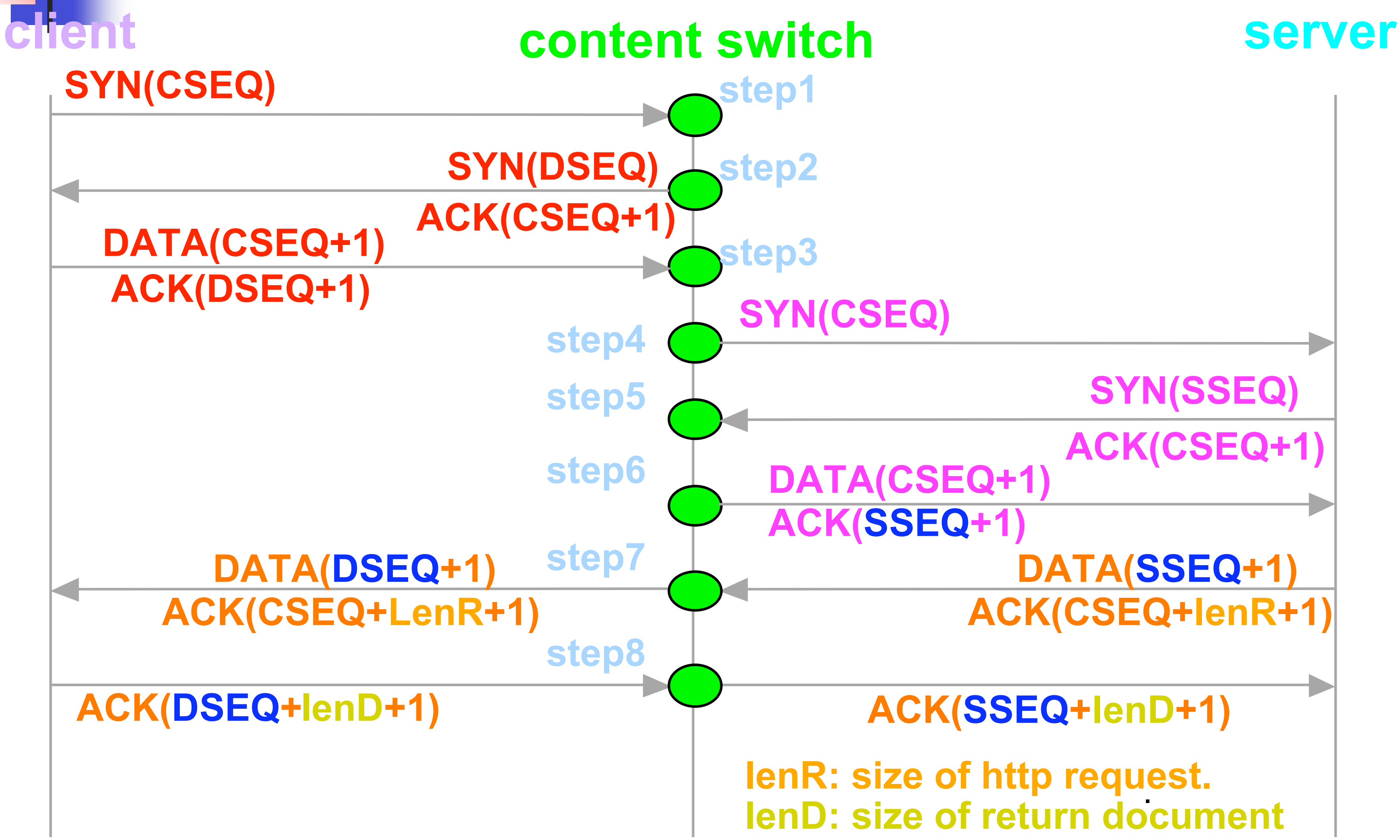

- Fix request file size @ 64 KB, increase the request rate
- 665.6Mbps vs. 720.9Mbps

# Conclusions

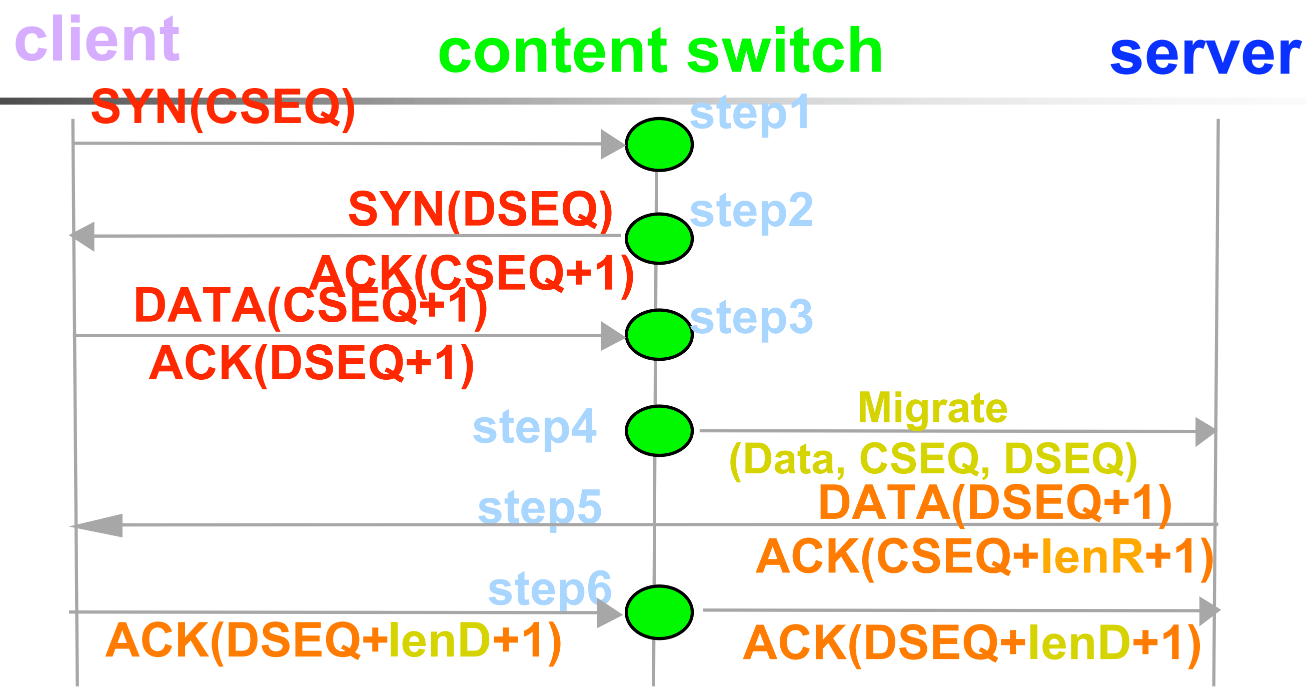
- Designed and implemented a content-aware switch using IXP2400
- Analyzed various tradeoffs in implementation and compared its performance with a Linux-based switch
- Measurement results show that NP-based switch can improve the performance significantly

# Backups

# TCP Splicing

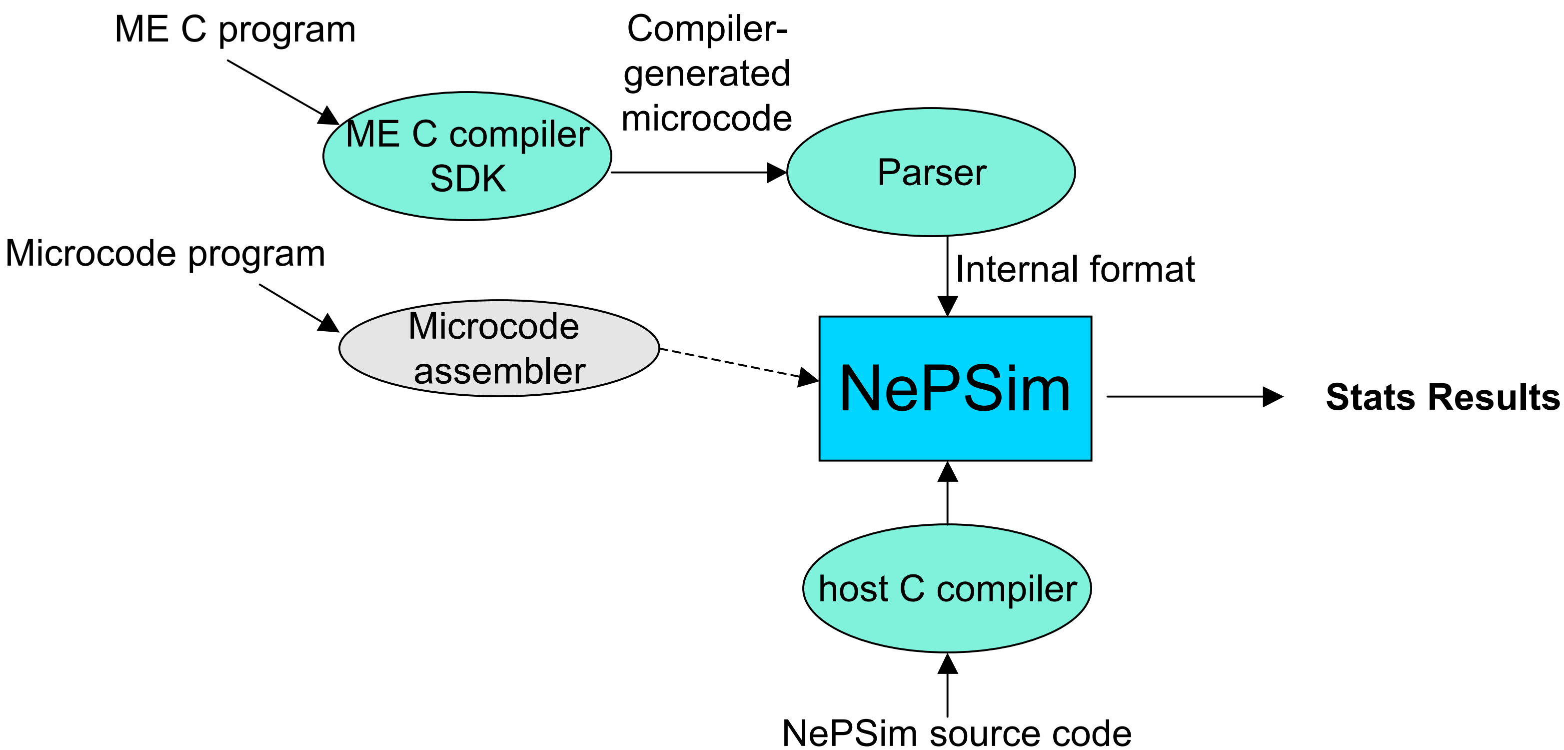


# TCP Handoff




- Migrate the created TCP connection from the switch to the back-end sever
  - Create a TCP connection at the back-end without going through the TCP three-way handshake
  - Retrieve the state of an established connection and destroy the connection without going through the normal message handshake required to close a TCP connection
- Once the connection is handed off to the back-end server, the switch must forward packets from the client to the appropriate back-end server

# NePSim Overview



# NePSim Internals (I)

- Instruction
  - Opcode: ALU, memory ref., CSR access etc.
  - operands: GPR, XFER, immed,
  - Shift: shift amount
  - Optional token: ctx_swap, ind_ref, …
- Command ( for memory, fbi accesses)
  - Opcode: sram_read, sram_write, sdram_read, …
  - Thread id: ME, thread
  - Functional unit: sram, sdram, scratchpad, fbi
  - Address: source or destination address
  - Reg: source or destination XFER register
  - Optional token: ctx_swap, ind_ref, …
- Event
  - <cycle time, command>