



# Network Processor: Architecture and Applications

---

Yan Luo  
For 16.480/552





# Outline

---

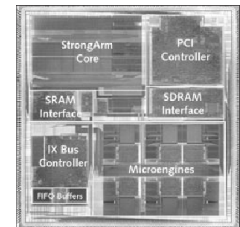
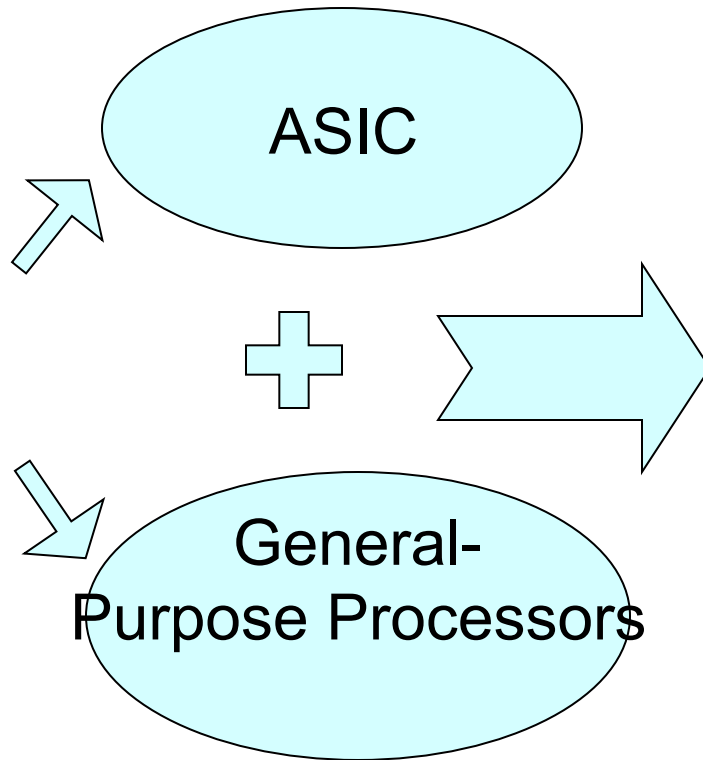
- Overview of Network Processors
- Network Processor Architectures
- Applications
- Case Studies
  - Wireless Mesh Network
  - Deep Packet Inspection with Netronome NP Card
  - A Content-Aware Switch



# Packet Processing in the Future Internet

## Future Internet

More packets  
&  
Complex packet  
processing

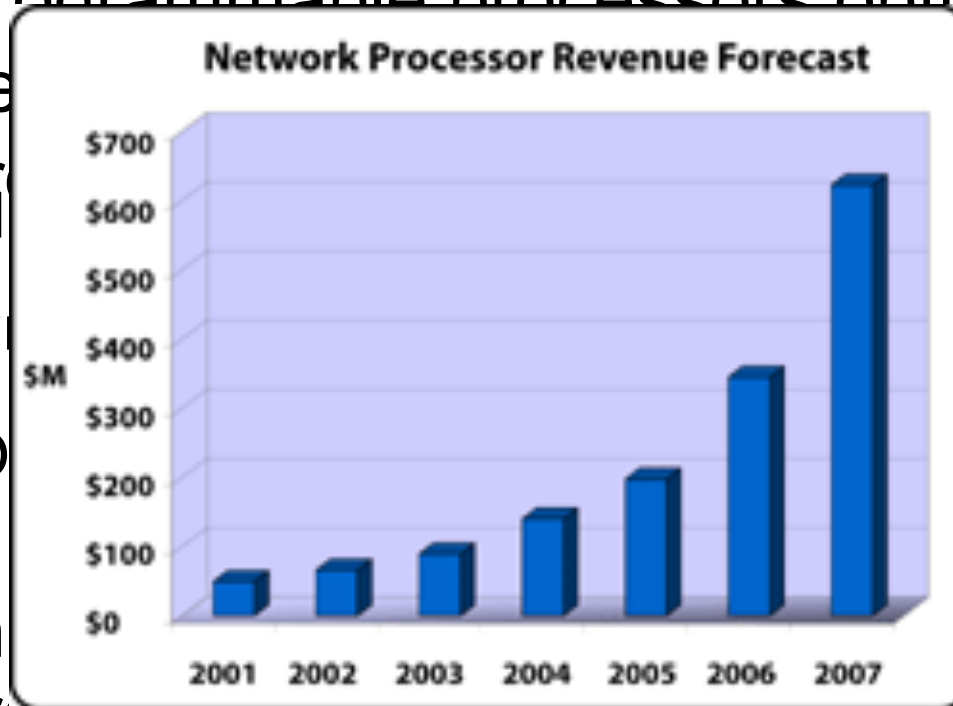


- High processing power
- Support wire speed
- Programmable
- Scalable
- Optimized for network applications
- ...



# What is Network Processor ?

- Programmable processors optimized for network processing
- High performance
- Programmable
- On-chip
- Mainstream Aggressive



Semico Research Corp. Oct. 14, 2003



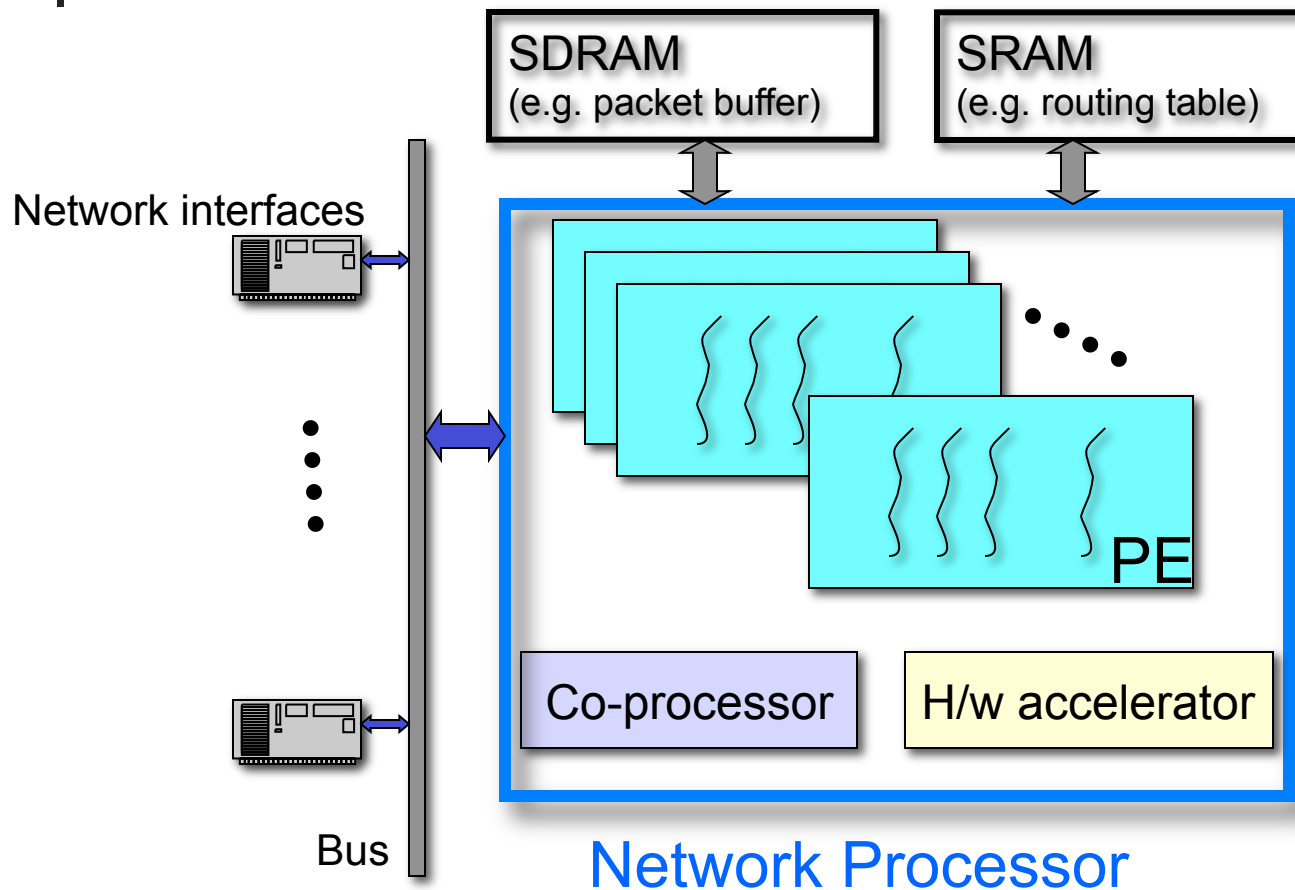


# Commercial Network Processors

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

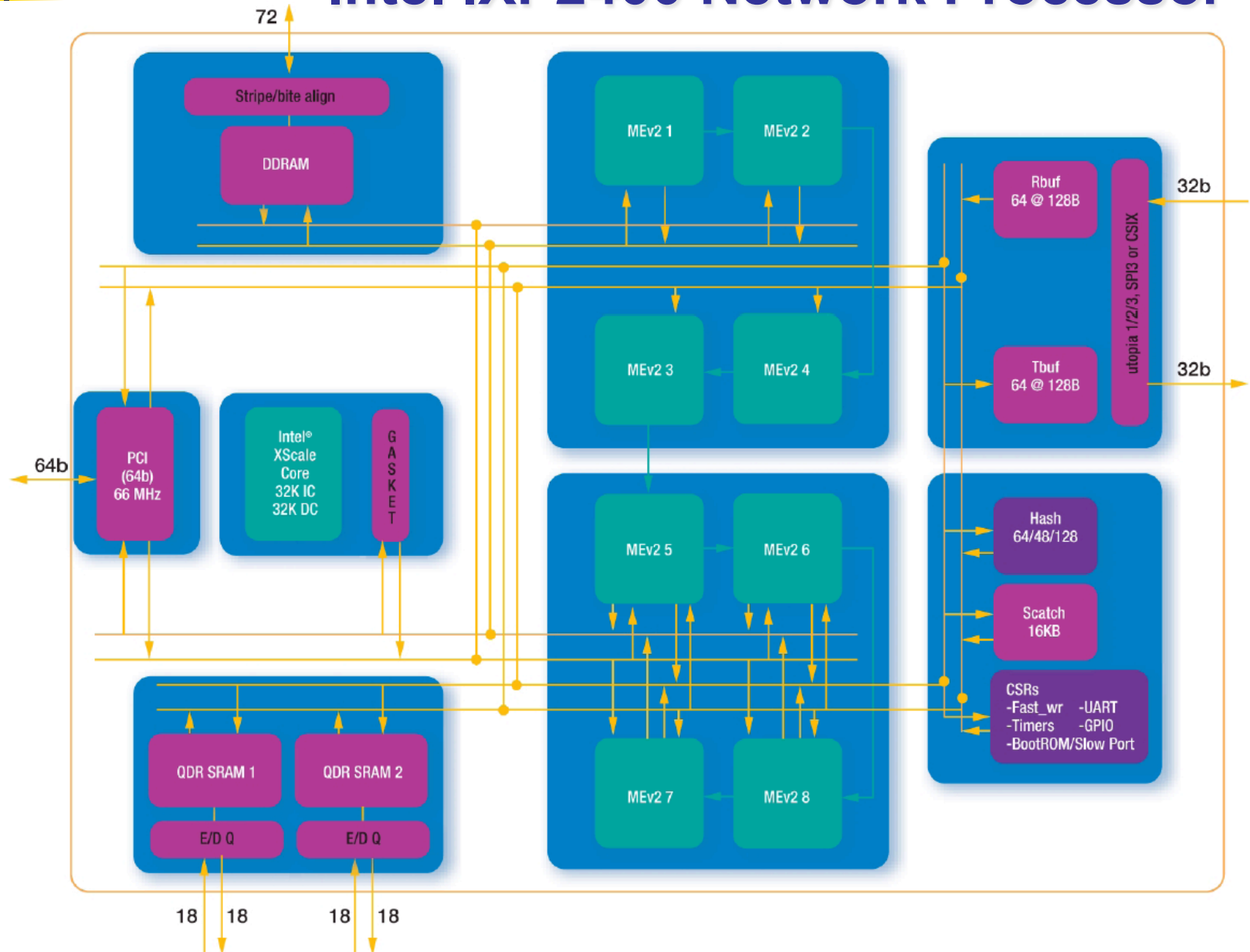


# Typical Network Processor Architecture



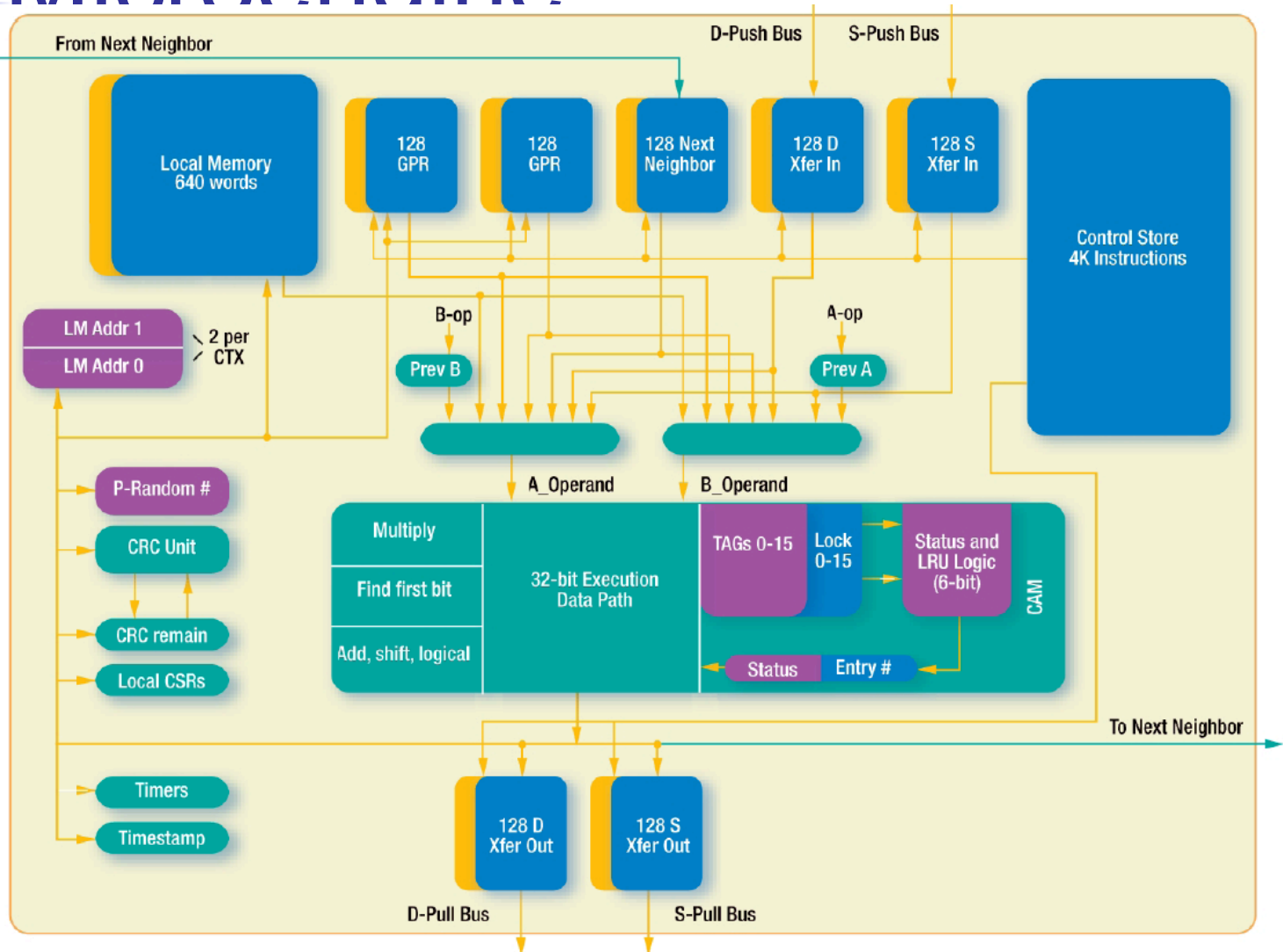


# Intel IXP2400 Network Processor





# Microengine



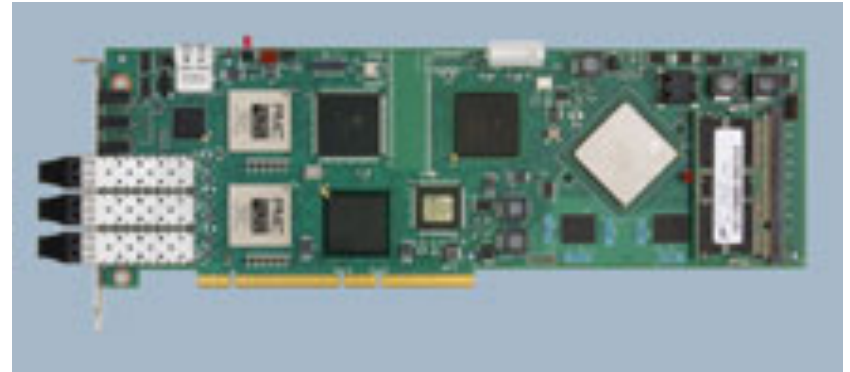


# Snapshots of IXP2xxx Based Systems



## ADI Roadrunner Platform

- IPv4 Forwarding/NAT
- Forwarding w/ QoS / DiffServ
- ATM RAN
- IP RAN
- IPv6/v4 dual stack forwarding

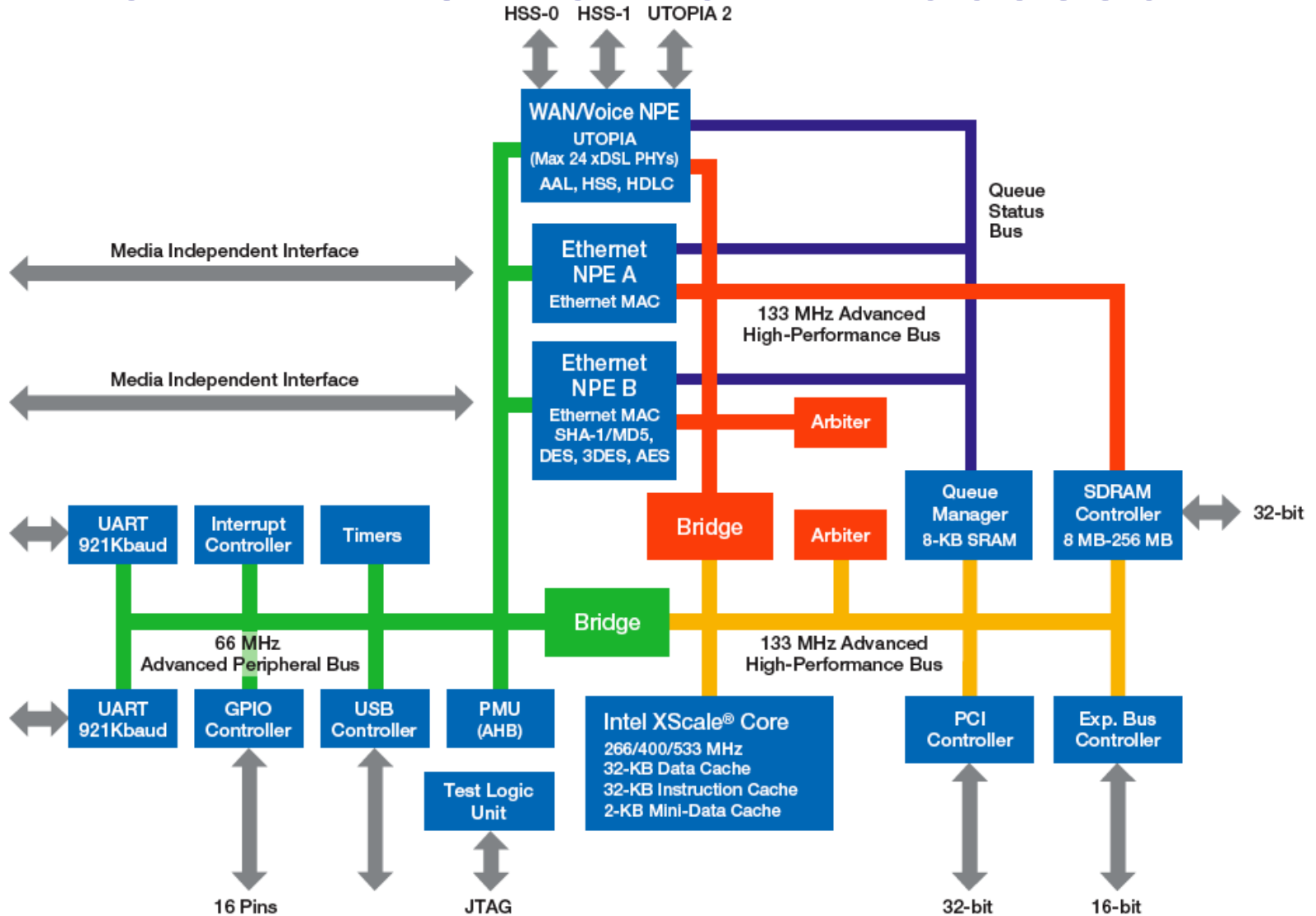


## Radisys ENP2611 PCI Packet Processing Engine

- multiservice switches,
- routers, broadband access devices,
- intrusion detection and prevention (IDS/IPS)
- Voice over IP (VoIP) gateway
- Virtual Private Network gateway
- Content-aware switch

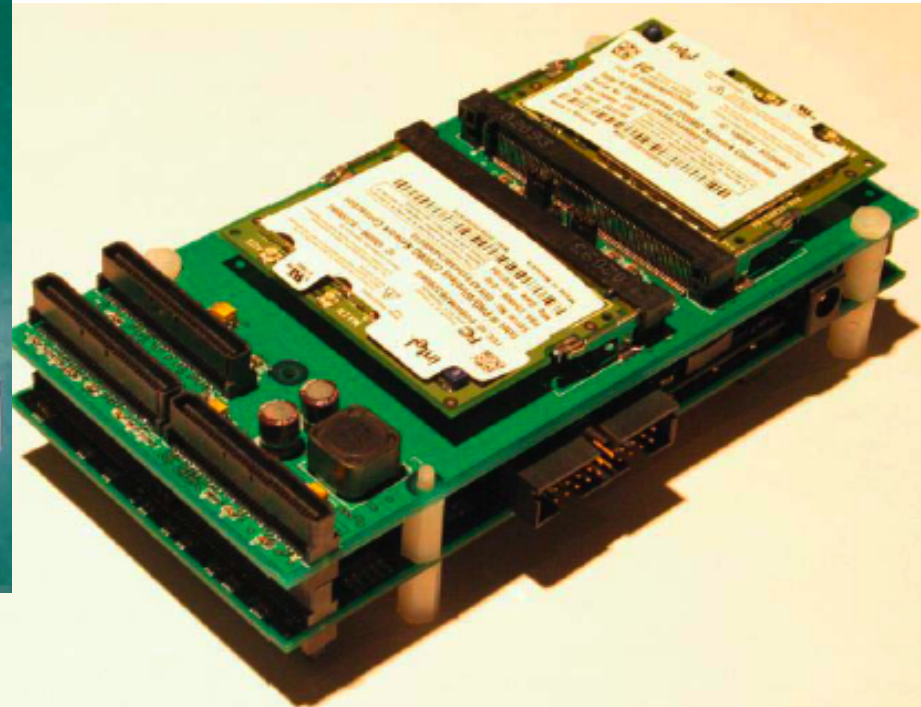


# Intel IXP425 Network Processor



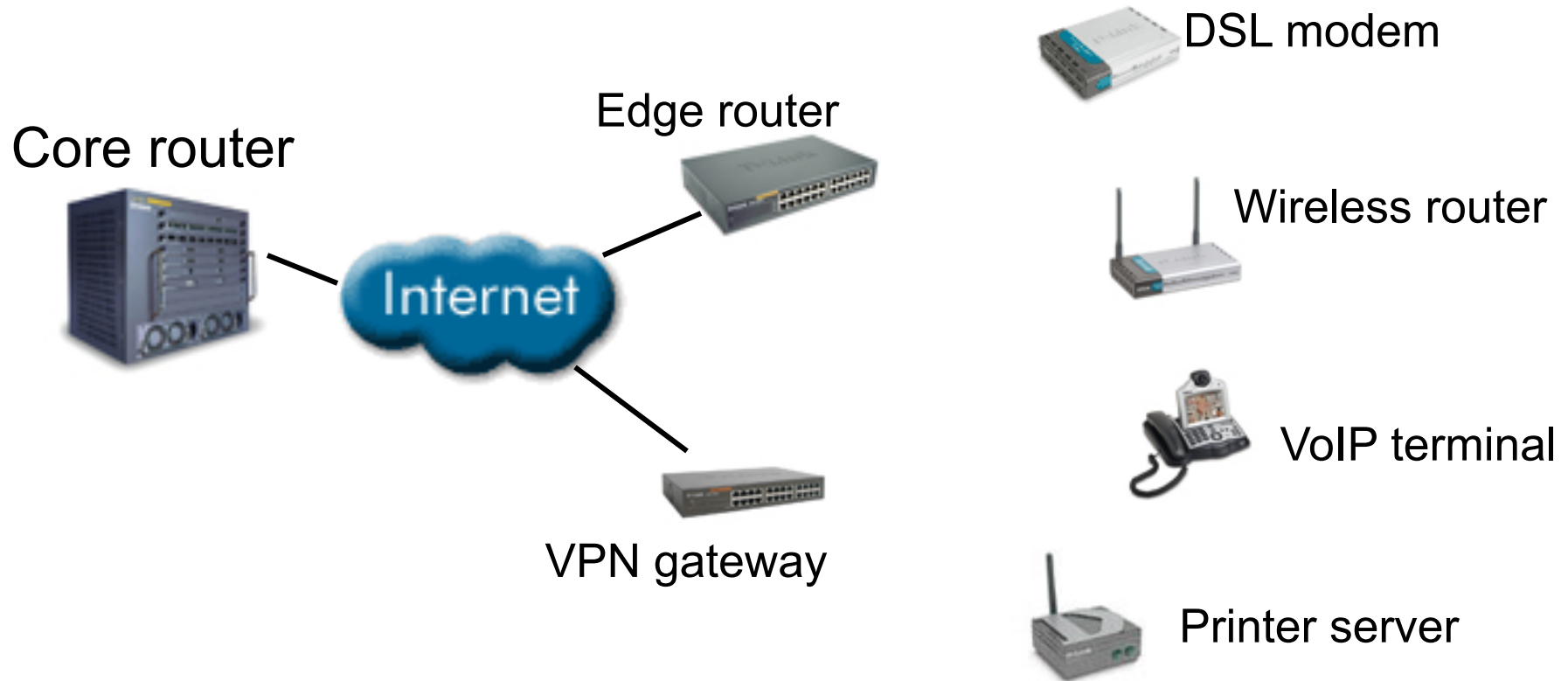


# StarEast: IXP425 Based Multi-radio Platform



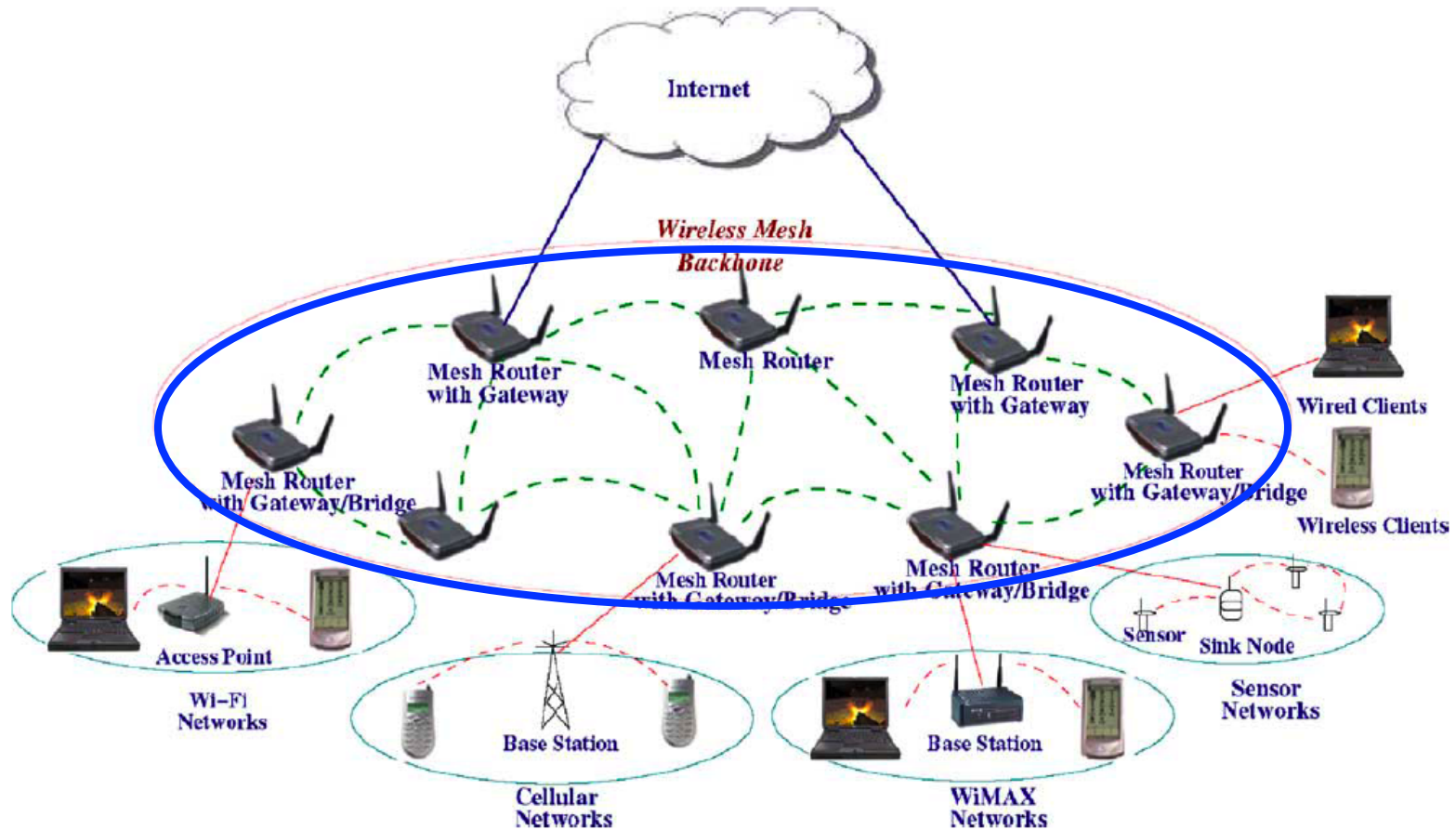


# Applications of Network Processors



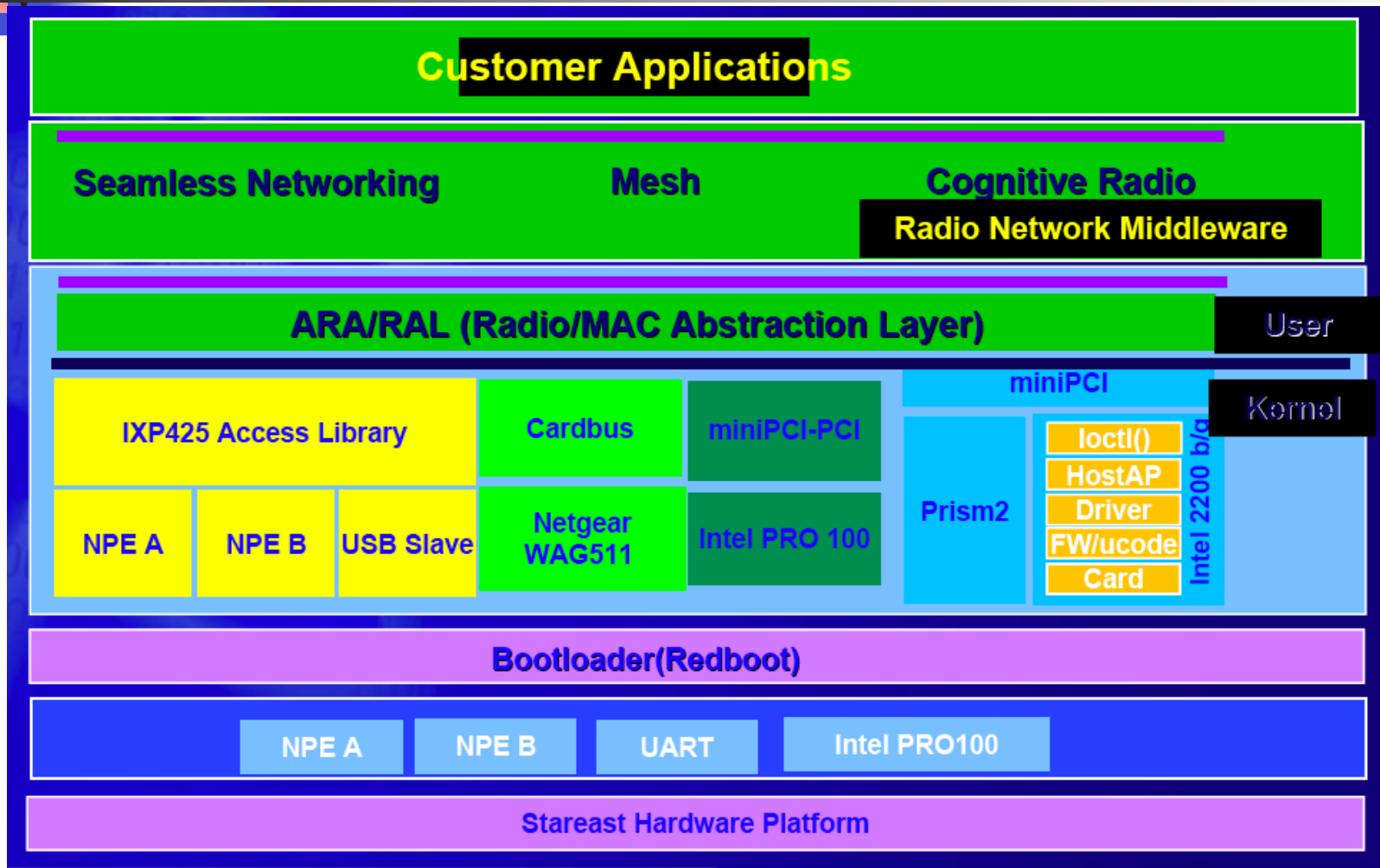


# Case Study 1: Wireless Mesh Network





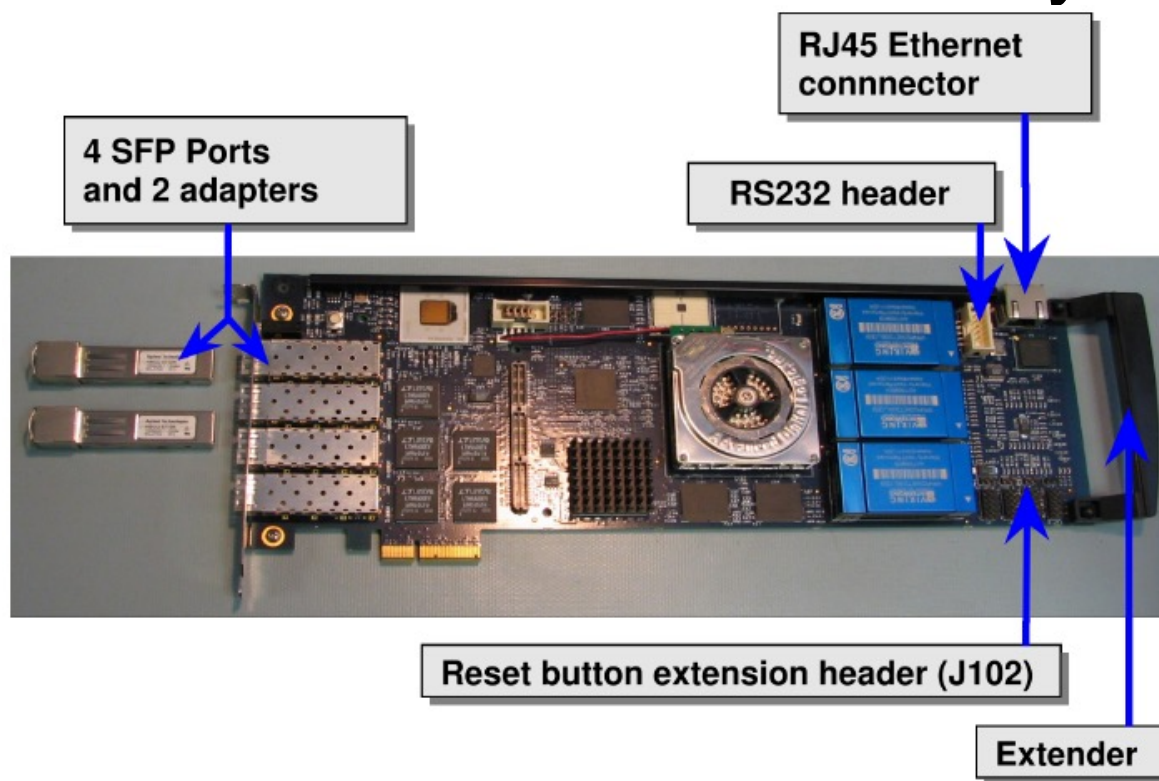
# Software Stack on StarEast





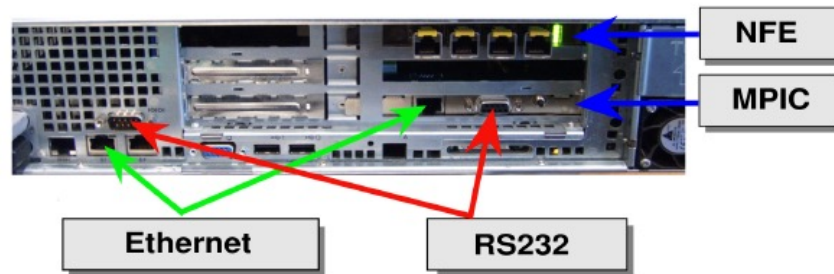
# Case Study 2: Deep Packet Inspection with Netronome NP Card

- NFE-i8000 from Netronome Systems Inc.





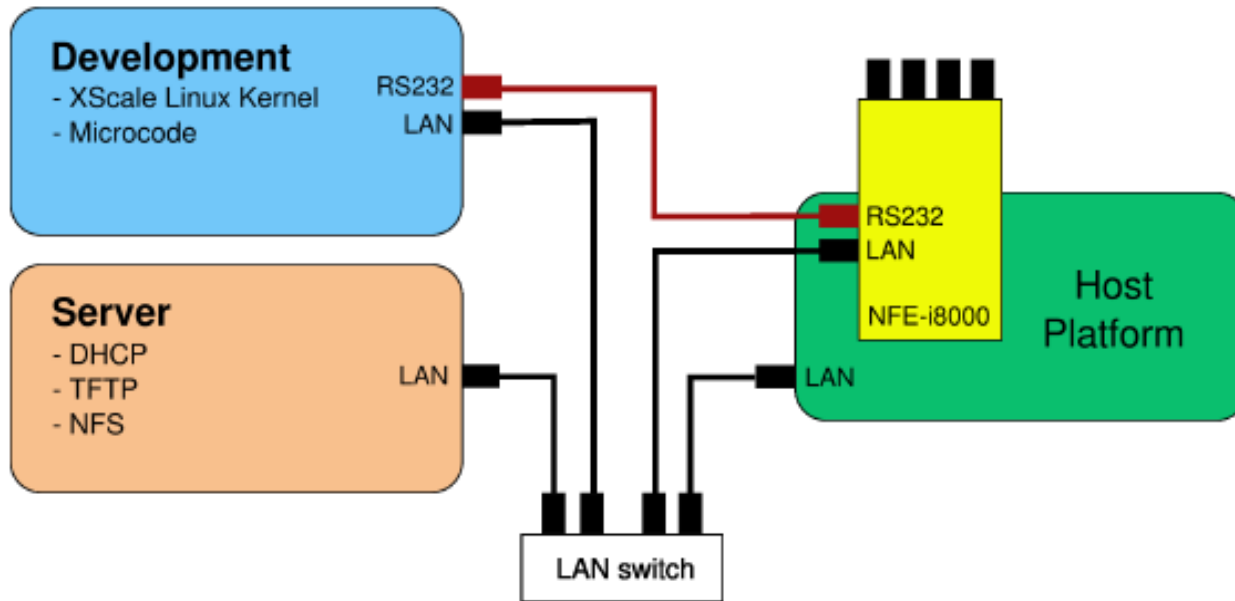
# Setup the NP Card for Test



1. Connect the RS232 serial cable between a host platform RS232 serial port (male DM9 or DM25 connector) and the RS232 serial port (female DB9 connector) on the MPIC.
  - The factory default settings for RS232 are 115200 baud, 8 data bit, No parity and 1 stop bit.
2. Connect the Cross-over Ethernet cable between the first RJ45 Ethernet port of the Host and the RJ45 Ethernet port on the MPIC.
3. Connect the power cable to the PSU of the Host platform and power it up.
4. Finally, insert the distribution media into the media drive.



# Development Environment







# Dynamic Host Configuration Protocol (DHCP)

---

- The DHCP service
  - assigns, upon request from a client, a dynamic IP Address which is reserved and unused on the network.
  - Additional information such as routing and other network information
- Upon boot-up, the NFE' s hardware
  - Boot Monitor (RedBoot) will generate and send BOOTP request for an IP address
  - be serviced by DHCP service.
- DHCP service be configured to provide additional information
  - such as the path of the Network File System (NFS) mount point to the NFE hardware platform when it boots the Linux operating system.



# Trivial File Transfer Protocol (TFTP)

- The TFTP service transfers, upon request from a client, a specified file to the client.
- To use this service, a client must have an IP Address, statically assigned or dynamically acquired from a DHCP service, before it can send a file request to this service.
  - That is why we need DHCP
- The NFE hardware platform will, after being assigned an IP address and if so configured, download the **operating system image file** using this service.
  - Most embedded system does not have a bootable disk
  - Because the processor needs a OS to manage the resources
- After the file has been downloaded into memory, the platform will execute the file to boot the operating system.





# Network File System (NFS)

---

- The NFS service provides a remote file system, mounted and accessed across the network.
- This service makes it possible for remote systems without storage hardware, but connected to a network, to boot the Linux operating system.
- Files in this file system may be read, written and/or executed by all the clients platforms (permissions permitting) which mount the file system.
  - The **Netronome Systems** provided MontaVista Linux Kernel image depends on this service.





# Host and Target

---

- Host
  - The development host
  - Run services and development software
    - Linux, Windows, Cross-compilers
- Target: the network processor
  - Xscale - Run Linux, Initialization Tasks
    - MontaVista Linux Pro 3.1
  - Microengines - the packet processing units, programmable with microcode.
    - Intel IXA SDK 4.2





# Boot Monitor and Diagnosis

---

- RedBoot
- Flash File System
- Diagnosis Utilities





# RedBoot

---

- RedBoot is a complete bootstrap environment for embedded systems.
- RedBoot allows download and execution of embedded applications via serial or ethernet,
  - including embedded Linux and eCos applications.
  - can be used for both product development and deployed products in the field (flash update and network booting).
- Ethernet download and debug support is included, allowing RedBoot
  - to retrieve its IP parameters via BOOTP or DHCP,
  - program images to be downloaded using TFTP.
  - Images can also be downloaded over serial, using X- or Y-modem.





# RedBoot (2)

---

- RedBoot can be used to communicate with GDB (the GNU Debugger)
  - debug applications via serial or ethernet,
  - including the ability to interrupt a running application started by GDB.
- An interactive command-line interface is provided to allow
  - management of the Flash images,
  - image download,
  - RedBoot configuration, etc., accessible via serial or ethernet.
  - For unattended or automated startup, boot scripts can be stored in Flash allowing for example loading of images from Flash or a TFTP server.





# Booting of the Board

---

Recovery loader initializing...

Netronome Systems NFEi8000 Boot Monitor version 1.10

SRAM channel 0: Deskew=8, DLL=8 (default=8), ExtPipeline=0

Found TCAM.

Found 8MB SRAM (1 x 8MB).

SRAM channel 2: Deskew=7, DLL=11 (default=11), ExtPipeline=0

Found 16MB SRAM (2 x 8MB).

SRAM channel 3: Deskew=7, DLL=11 (default=11), ExtPipeline=0

Found 16MB SRAM (2 x 8MB).

SRAM channel 1: Deskew=4, DLL=12 (default=12), ExtPipeline=3

DCM locked [PCB Rev = 4].

Found FPGA.

Confluence CPLD Revision 2.00

Armstrong FPGA Revision 6.14

Serial number 07010015

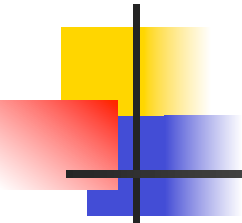
+... waiting for BOOTP information - enter ^C to abort

Ethernet eth0: MAC address 00:15:4d:00:02:30

IP: 192.168.10.4/255.255.255.0, Gateway: 192.168.10.1

Default server: 192.168.10.1





RedBoot(tm) bootstrap and debug environment [ROM]  
Non-certified release, version 1.10 - built 11:50:02, Sep 1 2006

Platform: Netronome Systems NFEi8000 (XScale) IXP2855 A step  
Portions Copyright (C) 2000-2004 Red Hat, Inc.  
Portions Copyright (C) 2000-2005 Intel Corporation.  
Portions Copyright (C) 2004-2006 Netronome Systems, Inc.

DRAM: 0x00000000-0x30000000, [0x00016d88-0x2ffdd000] available  
FLASH: 0xc4000000 - 0xc5000000, 128 blocks of 0x00020000 bytes each.

== Executing boot script in 10.000 seconds - enter ^C to abort

RedBoot> load -r -v -m tftp -b 0x00280000 -h 192.168.10.1 zImageN

-

Raw file loaded 0x00280000-0x003311cf, assumed entry at 0x00280000

RedBoot> exec -c "console=ttyS0,115200 ip=bootp root=nfs"

Using base address 0x00280000 and length 0x000b11d0

Uncompressing Linux..... done, bo.Linux version 2.4.20\_mvl31-perch (root@jade) (gcc  
version 3.3.1 (MontaVista 3.36CPU: XScale-IXP2800 [690541ac] revision 12 (ARMv5TE))

CPU: D undefined 5 cache

CPU: I cache: 32768 bytes, associativity 32, 32 byte lines, 32 sets

CPU: D cache: 32768 bytes, associativity 32, 32 byte lines, 32 sets

Machine: Intel Perch Island board

Ignoring unrecognised tag 0x00000000





....

IP-Config: Complete:

device=eth0, addr=192.168.10.4, mask=255.255.255.0, gw=192.168.10.1,  
host=i8000.netronome.com, domain=, nis-domain=(none),  
bootserver=192.168.10.1, rootserver=192.168.10.1, rootpath=/opt/netronomefsNET4: Unix domain sockets 1.0/  
SMP for Linux NET4.0.

RAMDISK: Couldn't find valid RAM disk image starting at 0.

Freeing initrd memory: 4096K

Looking up port of RPC 100003/2 on 192.168.10.1

Looking up port of RPC 100005/1 on 192.168.10.1

VFS: Mounted root (nfs filesystem).

.....

nothing was mounted

Starting portmap daemon: portmap.

Cleaning: /tmp /var/lock /var/run/etc/init.d/rcS: /var/run/utmp: Read-only filemINIT: Entering runlevel: 3

Starting kernel log daemon: klogd.

Starting devfsd: Started device management daemon for /dev  
done.

Starting internet superserver: inetd.

Starting OpenBSD Secure Shell server: sshd.

MontaVista(R) Linux(R) Professional Edition 3.1

i8000.netronome.com login:





# Flash Memory

**Table 2.1. Flash Images and their locations**

Name	Flash Address	Memory Address	Length	Entry Point
Recovery Loader	0xC4000000	0xC4000000	0x00100000	0x00000000
Boot Monitor	0xC4100000	0xC4100000	0x00040000	0x00000000
Persistent storage	0xC4200000	0xC4200000	0x00080000	N/A
RAM Disk	0xC4300000	0x2C600000	0x00300000	0x2C600000
Diags	0xC4D00000	0x00280000	0x000A0000	0x00280000
Linux	0xC4DA0000	0x00280000	0x000C0000	0x00280000

- Some flash file system commands
  - fis list
  - fis erase
  - fis write





# Example: Store OS Kernel in Flash Memory

---

1. Activate the **Boot Monitor** as described in Section 2.2.
2. Make sure the Linux Kernel image is available on a TFTP server. The image is called `kernel_v1.1` and is available on the distribution at the following location:  
`/Hardware/NFE/i8000/IXP/XScale/Linux/Montavista/v3.1PRO/Images/v1.1`
3. Load the image into the NFE-i8000 hardware platform RAM (replacing xyz with the IP address of the TFTP server):

```
RedBoot> load -r -v -m tftp -b 0x00280000 -h xyz kernel_v1.1
```

The following message is displayed:

```
Raw file loaded 0x00280000-0x003311cf, assumed entry at 0x00280000
```

To calculate the size of the image, subtract the first value in the message from the second value and add 1:

$0x003311cf - 0x00280000 + 1 = 0xb11d0$ .





```
RedBoot> fis list
```

The output will list images that are stored in flash memory:

Name	FLASH addr	Mem addr	Length	Entry point
... (reserved)	0xC4000000	0xC4000000	0x00100000	0x00000000
RedBoot	0xC4100000	0xC4100000	0x00040000	0x00000000
linuxdata	0xC4200000	0xC4200000	0x00080000	0xFFFFFFFF
ramdisk	0xC4300000	0x2C600000	0x00300000	0x2C600000
diag	0xC4D00000	0x00280000	0x000A0000	0x00280000
linux	0xC4DA0000	0x00280000	0x000C0000	0x00280000
FIS directory	0xC4FE0000	0xC4FE0000	0x0001F000	0x00000000
RedBoot config	0xC4FFF000	0xC4FFF000	0x00001000	0x00000000
...				

To delete the Linux image, enter:

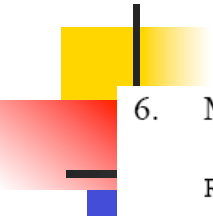
```
RedBoot> fis delete linux
```

and at the prompt to confirm writing, enter: y

5. Store the new image to flash memory:

```
RedBoot> fis create -b 0x280000 -l 0xc0000 -r 0x280000 -e 0x280000 -f 0xC4DA0000 linux
```



- 
6. Make sure an empty image in flash memory named linuxdata is present:

```
RedBoot> fis list
```

The output will list images that are stored in flash:

Name	FLASH addr	Mem addr	Length	Entry point
...				
(reserved)	0xC4000000	0xC4000000	0x00100000	0x00000000
RedBoot	0xC4100000	0xC4100000	0x00040000	0x00000000
<b>linuxdata</b>	<b>0xC4200000</b>	<b>0xC4200000</b>	<b>0x00080000</b>	<b>0xFFFFFFFF</b>
ramdisk	0xC4300000	0x2C600000	0x00300000	0x2C600000
diag	0xC4D00000	0x00280000	0x000A0000	0x00280000
linux	0xC4DA0000	0x00280000	0x000C0000	0x00280000
FIS directory	0xC4FE0000	0xC4FE0000	0x0001F000	0x00000000
RedBoot config	0xC4FFF000	0xC4FFF000	0x00001000	0x00000000
...				

If the image, linuxdata, is not present, it must be created. Enter:

```
RedBoot> fis create -l 0x80000 -f 0xc4200000 -b 0xc4200000 linuxdata
```

7. Next, erase the contents of this area:

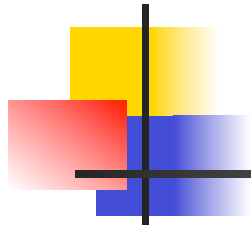
```
RedBoot> fis erase -l 0x80000 -f 0xc4200000
```

and at the prompt to confirm writing, enter: **y**

8. Load the Linux image without a Ramdisk, enter:

```
RedBoot> fis load linux
```





9. To execute the Linux Kernel image, enter one of the commands below. The first is to mount an NFS share as the root filesystem and the second is to use the Ramdisk image that has been loaded into memory as the root filesystem:

```
RedBoot> exec -c "console=ttyS0,115200 ip=bootp root=nfs"
```

or

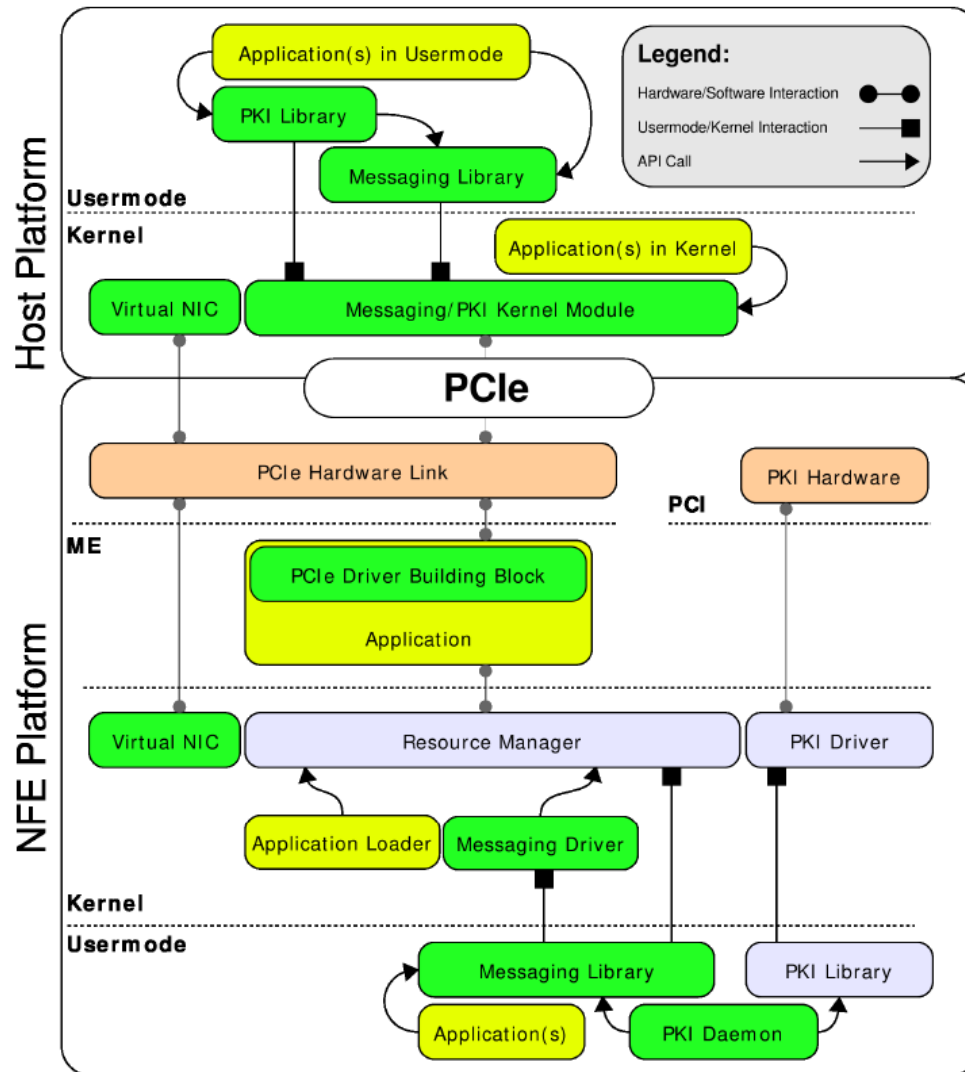
```
RedBoot> exec -c "console=ttyS0,115200 ip=off root=/dev/ram initrd=0x2c600000"
```





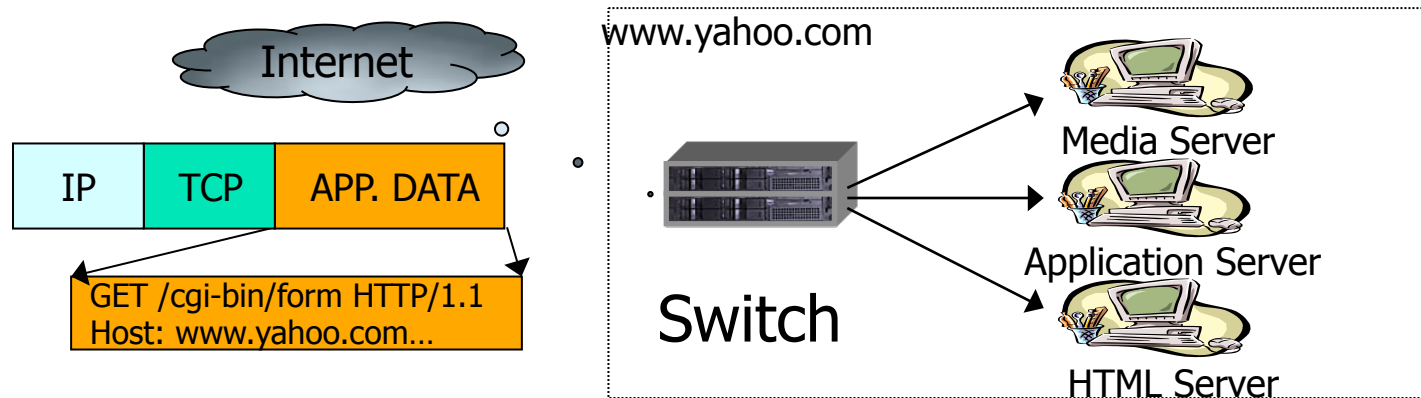


# System Block Diagram





# Case Study 3: Content-aware Switch



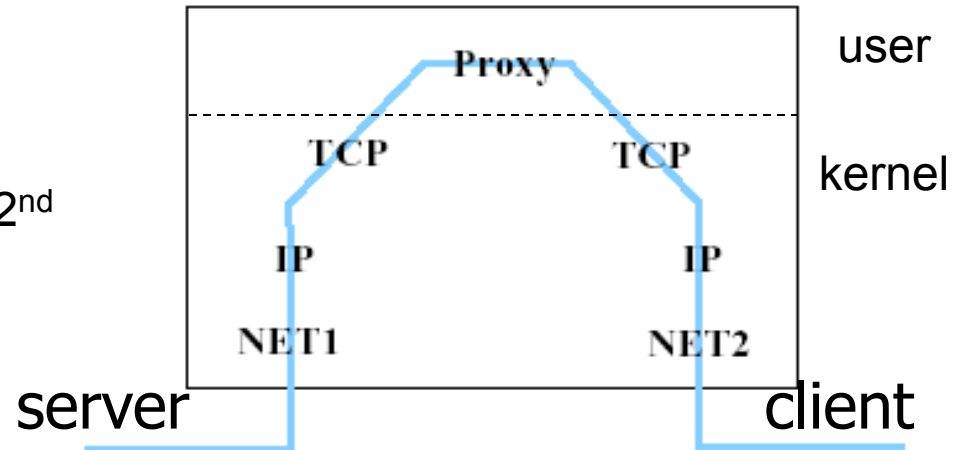
- Front-end of a Web cluster, only one Virtual IP
- Route packets based on Layer 5 information
  - Examine application data in addition to IP& TCP
- Advantages over layer 4 switches
  - Better load balancing: distributed based on content type
  - Faster response: exploit cache affinity
  - Better resource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



# Mechanisms to Build a Content-aware Switch

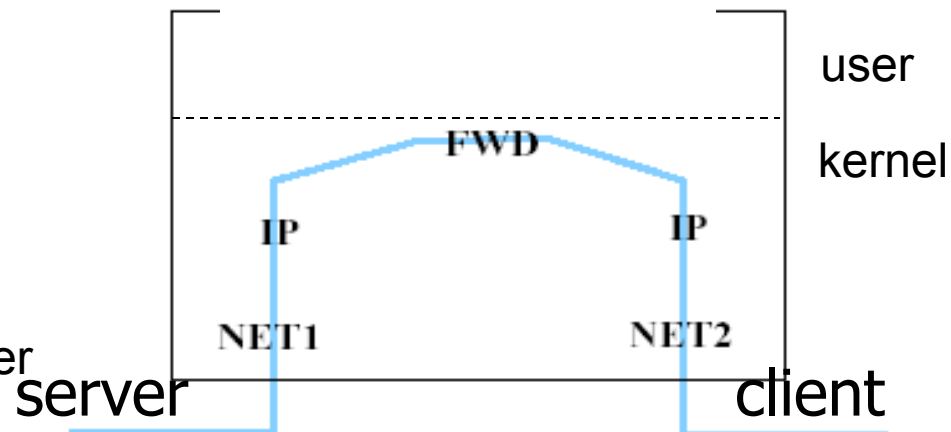
## ■ TCP gateway

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



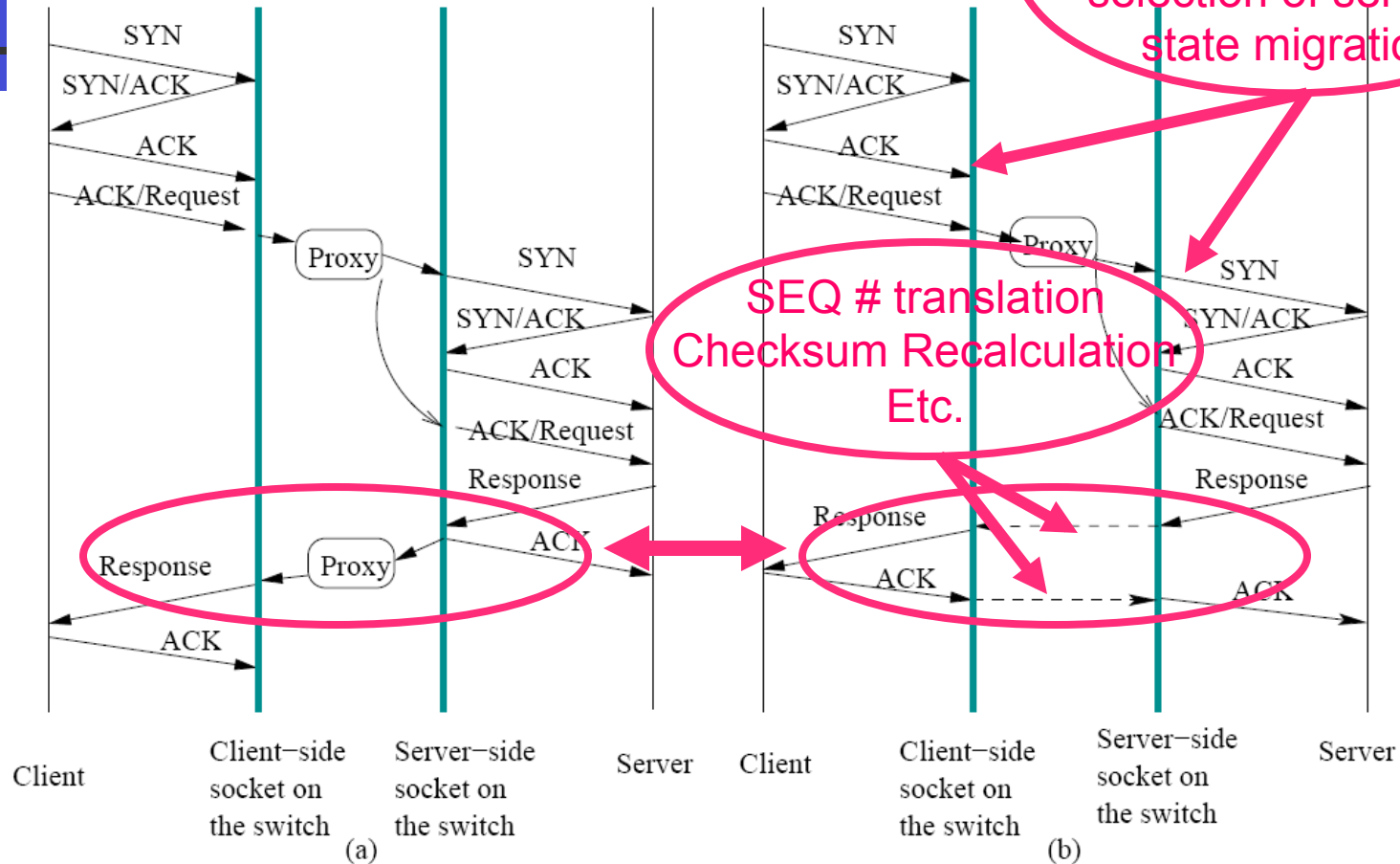
## ■ TCP splicing

- 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





# Anatomy of TCP Splicing

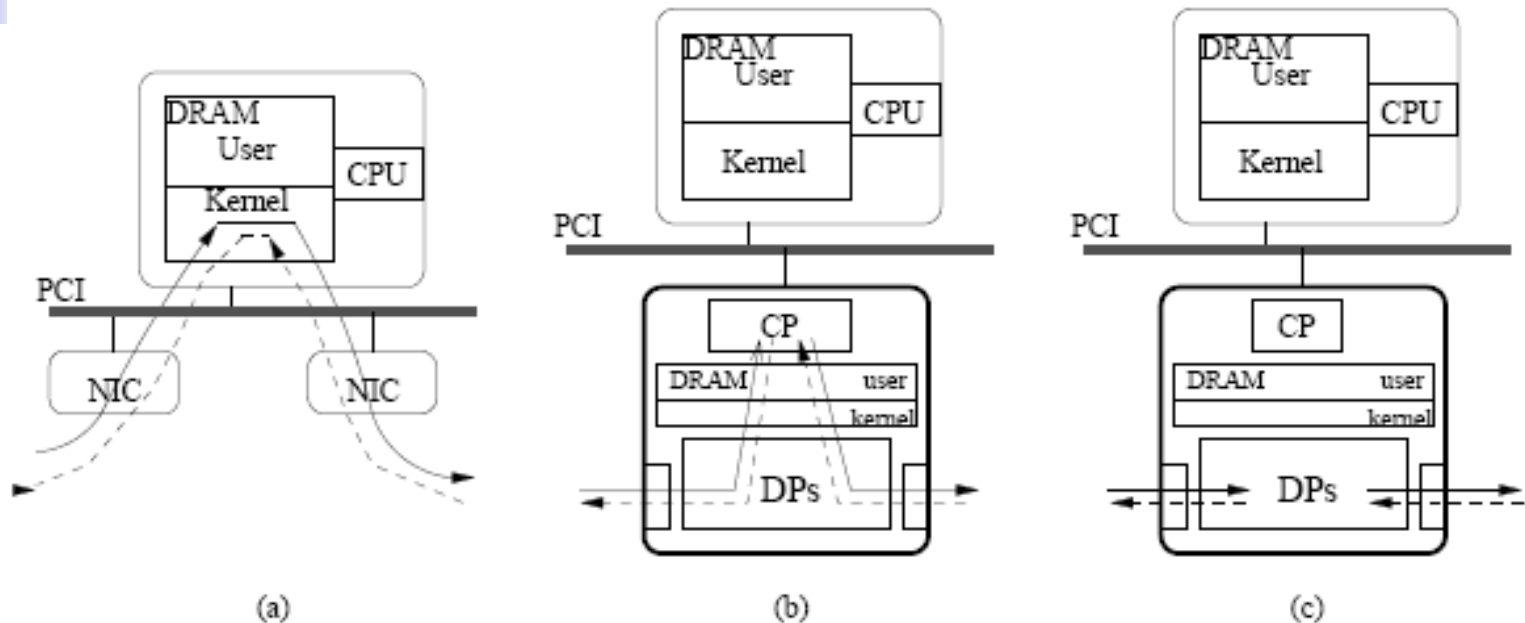


Without TCP Splicing

With TCP Splicing



# Design Options



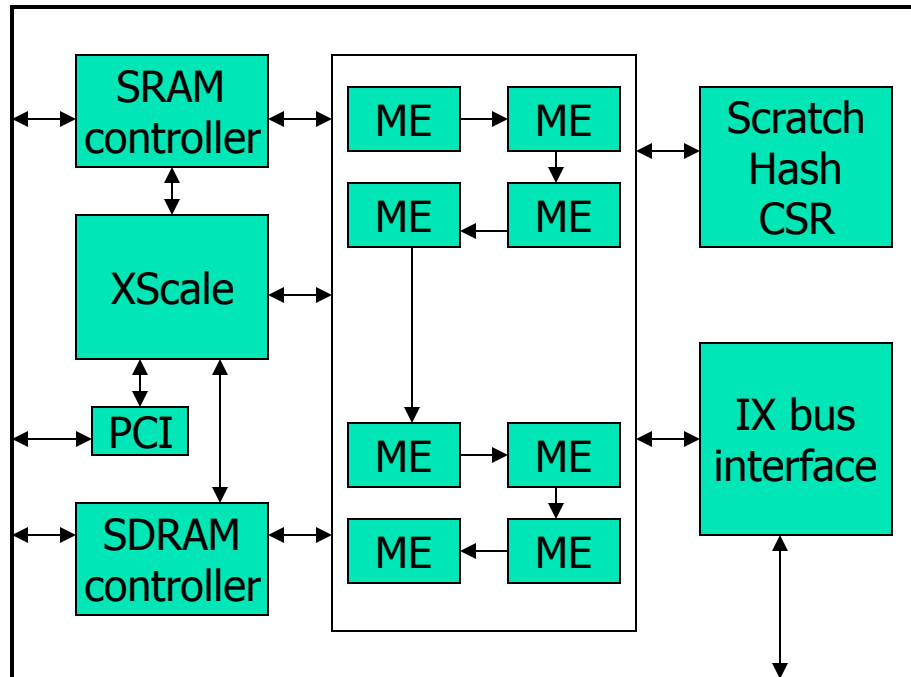
- Option 0: GP-based (Linux-based) switch
- Option 1: CP setup & and splices connections, DPs process packets sent after splicing

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

## SRAM (8MB)

- Client side CB list
- Server side CB list
- server selection table
- Locks

## DRAM (256MB)

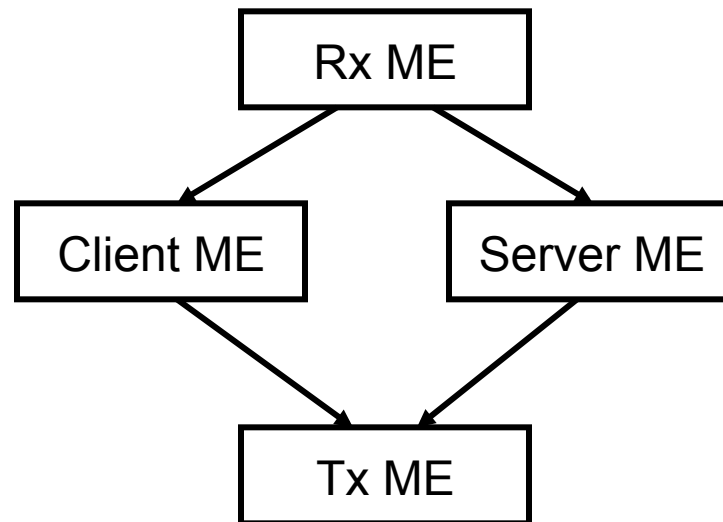
Packet buffer

## Scratchpad (16KB)

Packet queues

- Client-side control block list
  - record states for connections between clients and SpliceNP, states after splicing
- Server-side control block list
  - record states for connections between server and SpliceNP

## Microengines







# Comparison of Functionality

- *A lite version of TCP due to the limited instruction size of microengines.*

## Processing a SYN packet

Step	Functionality	TCP	Linux Splicer	SpliceNP
1	Dequeue packet	Y	Y	Y
2	IP header verification	Y	Y	Y
3	IP option processing	Y	Y	N
4	TCP header verification	Y	Y	Y
5	Control block lookup	Y	Y	Y
6	Create new socket and set state to LISTEN	Y	Y	No socket, only control block
7	Initialize TCP and IP header template	Y	Y	N
8	Reset idle time and keep-alive timer	Y	Y	N
9	Process TCP option	Y	Y	Only MSS option
10	Send ACK packet, change state to SYN_RCVD	Y	Y	Y





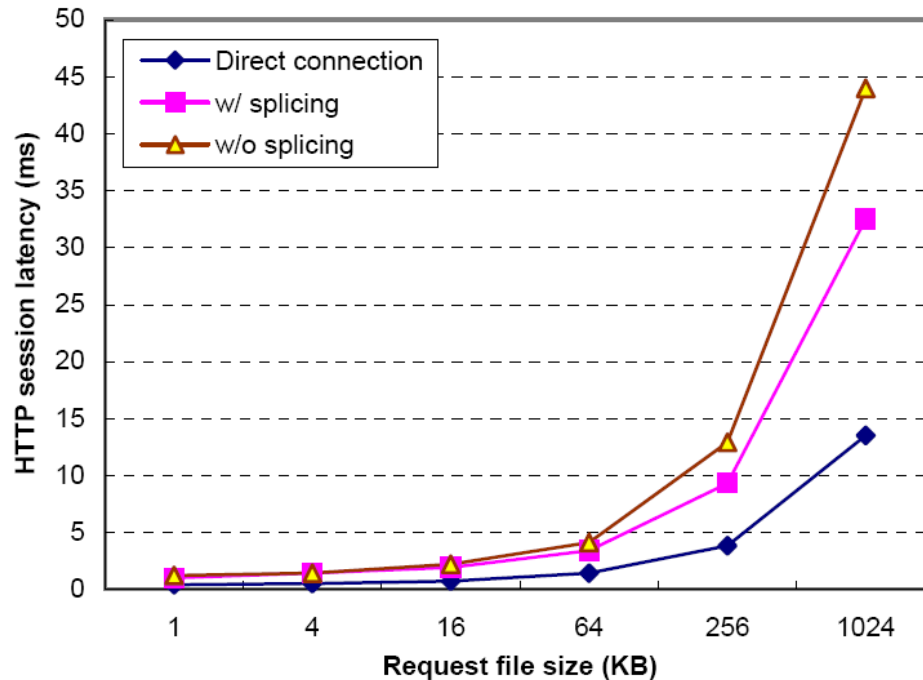
# Experimental Setup

---

- 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



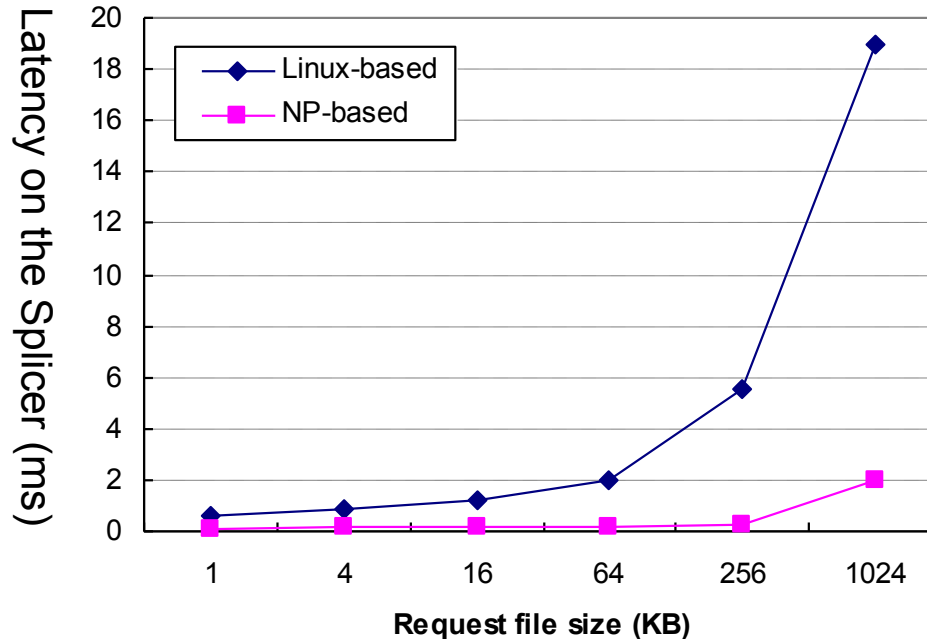
# Latency on a Linux-based TCP Splicer



- Latency is reduced by TCP splicing



# Latency vs Request File Size



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





# Comparison of Packet Processing Latency

**Table 5: Processing latency for control and data packets**

Packet Type		IXP2400		Linux Latency (us)	Latency reduction
		Microengine	Latency (us)		
Control Packet	SYN	clientME	7.2	48	85%
	ACK/Request	clientME	8.8	52	83%
	SYN/ACK	serverME	8.5	42	80%
Data Packet	Data	serverME	6.5	13.6	52%
	ACK	clientME	6.5	13.6	52%



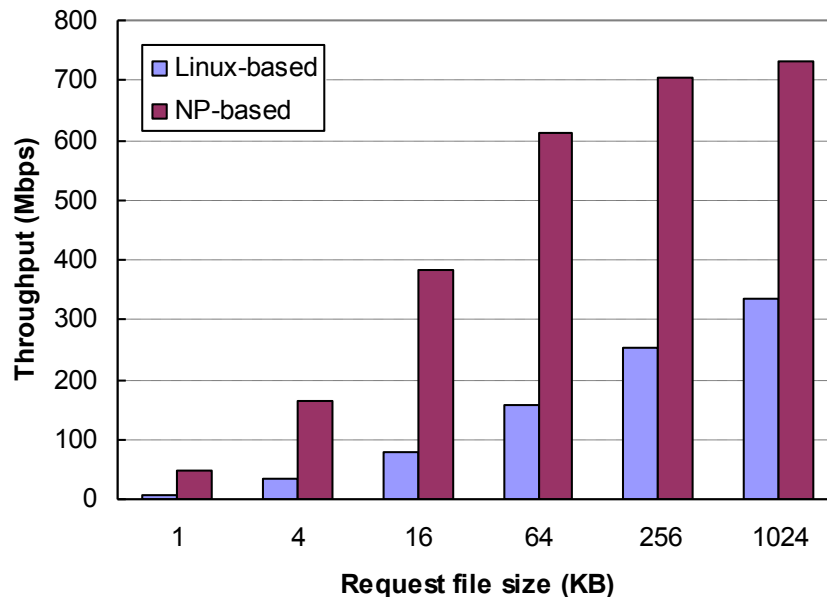


# Analysis of Latency Reduction

Linux-based	NP-based
Interrupt: NIC raises an interrupt once a packet comes	polling
NIC-to-mem copy 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 without two copies
Linux processing: OS overheads Processing a data packet in splicing state: 13.6 us	IXP processing: Optimized ISA 6.5 us



# Throughput vs Request File Size



- 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