# Router Construction II

Outline

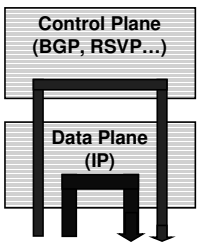
- Network Processors
- Adding Extensions
- Scheduling Cycles

## Observations

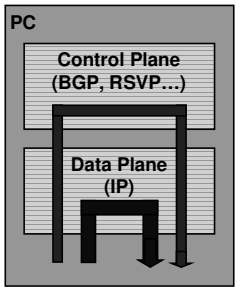
- Emerging commodity components can be used to build IP routers
  - switching fabrics, network processors, …
- Routers are being asked to support a growing array of services
  - firewalls, proxies, p2p nets, overlays, ...



## Router Architecture



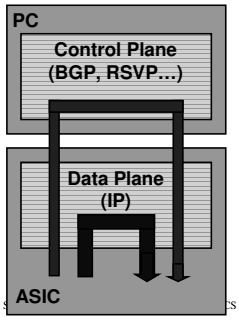
## Software-Based Router




+ Cost
+ Programmability
– Performance (~300 Kpps)
– Robustness



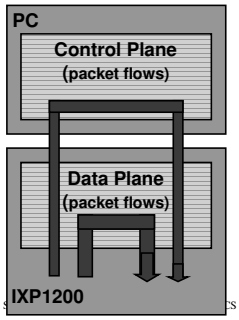
## Hardware-Based Router




– Cost
– Programmability
+ Performance (25+ Mpps)
+ Robustness

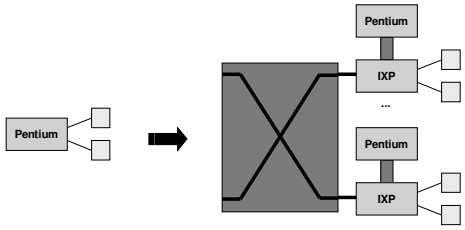


## NP-Based Router Architecture



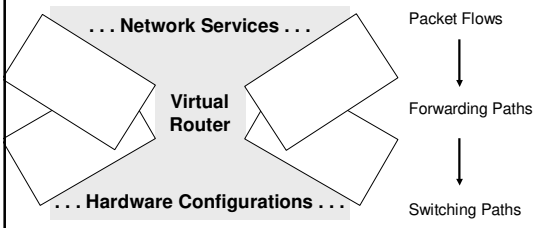

+ Cost ($1500)
+ Programmability
? Performance
? Robustness

# In General...





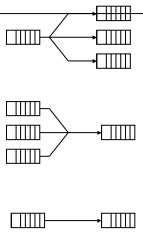

# Architectural Overview



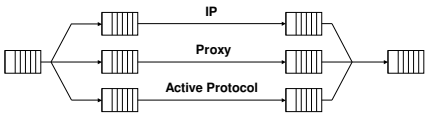



# Virtual Router

- Classifiers
- Schedulers
- Forwarders

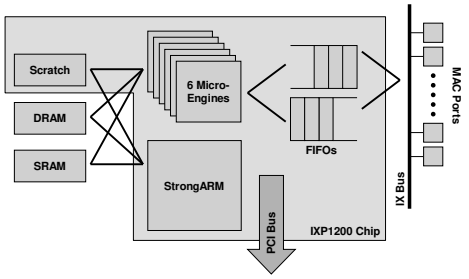


# Simple Example



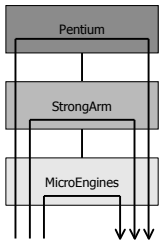



# Intel IXP

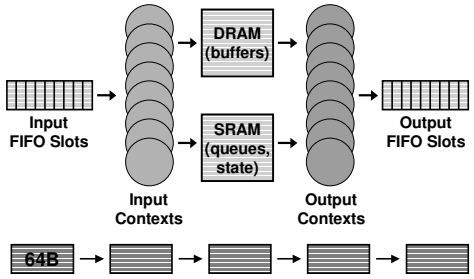





# Processor Hierarchy



# Data Plane Pipeline






# Data Plane Processing

**INPUT context loop**

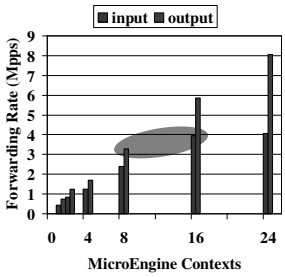
```
wait_for_data
copy in_fifo→regs
Basic_IP_processing
copy regs→DRAM
if (last_fragment)
     enqueue→SRAM
```

**OUTPUT context loop**

```
if (need_data)
  select_queue
  dequeue←SRAM
copy DRAM→out_fifo
```



# Pipeline Evaluation




Measured independently

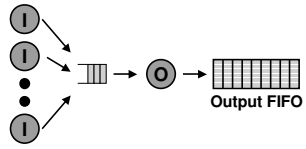
100Mbps Ether ≈ 0.142Mpps

## What We Measured

- Static context assignment
  - 16 input / 8 output
- Infinite offered load
- 64-byte (minimum-sized) IP packets
- Three different queuing disciplines



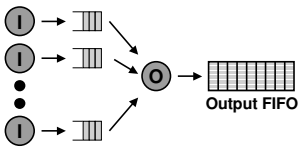
## Single Protected Queue




- Lock synchronization
- Max 3.47 Mpps
- Contention lower bound 1.67 Mpps

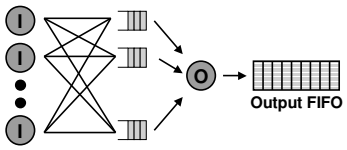


## Multiple Private Queues




- Output must select queue
- Max 3.29 Mpps

## Multiple Protected Queues




- Output must select queue
- Some QoS scheduling (16 priority levels)
- Max 3.29 Mpps



## Data Plane Processing

**INPUT context loop**

```
wait_for_data
copy in_fifo→regs
Basic_IP_processing
copy regs→DRAM
if (last_fragment)
     enqueue→SRAM
```

**OUTPUT context loop**

```
if (need_data)
  select_queue
  dequeue←SRAM
copy DRAM→out_fifo
```



## Cycles to Waste

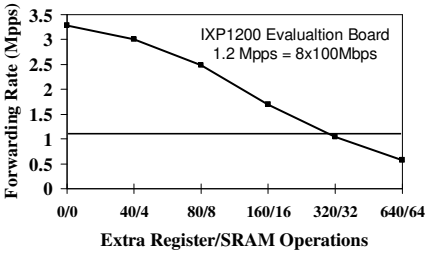
**INPUT context loop**

```
wait_for_data
copy in_fifo→regs
Basic_IP_processing
nop
nop
…
nop
copy regs→DRAM
if (last_fragment)
     enqueue→SRAM
```

**OUTPUT context loop**

```
if (need_data)
  select_queue
  dequeue←SRAM
copy DRAM→out_fifo
```

## How Many "NOPs" Possible?



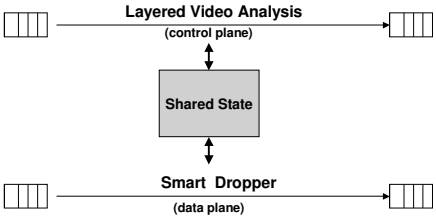



## Data Plane Extensions

| Processing | Memory Ops | Register Ops |
|---|---|---|
| Basic IP | 6 | 32 |
| TCP Splicer | 6 | 45 |
| TCP SYN Monitor | 1 | 5 |
| ACK Monitor | 3 | 15 |
| Port Filter | 5 | 26 |
| Wavelet Dropper | 2 | 28 |



## Control and Data Plane

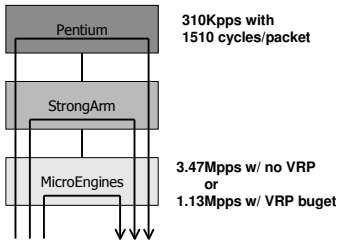


## What About the StrongARM?

- Shares memory bus with MicroEngines
  - must respect resource budget
- What we do
  - control IXP1200 ↔ Pentium DMA
  - control MicroEngines
- What might be possible
  - anything within budget
  - exploit instruction and data caches
- We recommend against
  - running Linux



## Performance

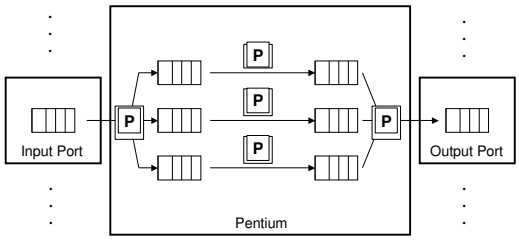





## Pentium

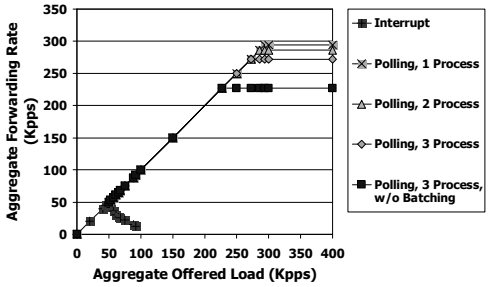
- Runs protocols in the control plane
  - e.g., BGP, OSPF, RSVP
- Run other router extensions
  - e.g., proxies, active protocols, overlays
- Implementation
  - runs Scout OS + Linux IXP driver
  - CPU scheduler is key

# Processes



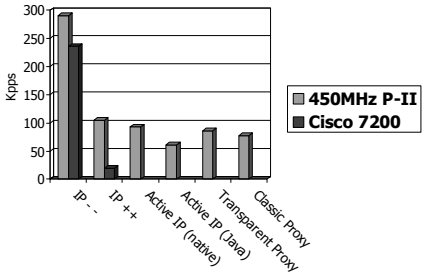



# Performance






# Performance (cont)



## Scheduling Mechanism

- Proportional share forms the base
  - each process reserves a cycle rate
  - provides isolation between processes
  - unused capacity fairly distributed
- Eligibility
  - a process receives its share only when its source queue is not empty and sink queue is not full
- Batching
  - to minimize context switch overhead

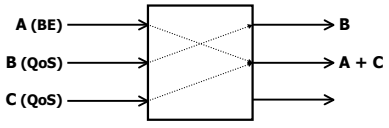


## Share Assignment

- QoS Flows
  - assume link rate is given, derive cycle rate
  - conservative rate to input process
  - keep batching level low
- Best Effort Flows
  - may be influenced by admin policy
  - use shares to balance system (avoid livelock)
  - keep batching level high

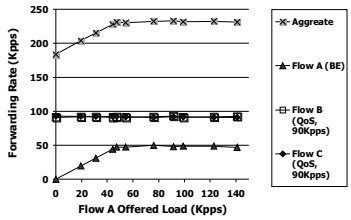


## Experiment



# Mixing Best Effort and QoS

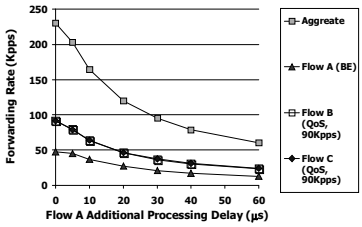
- Increase offered load from A





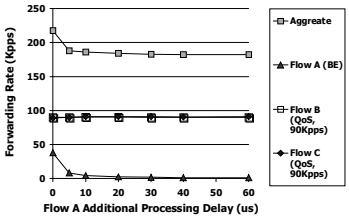

# CPU vs Link

- Fix **A** at 50Kpps, increase its processing cost



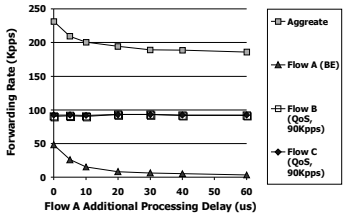



# Turn Batching Off




- CPU efficiency: 66.2%

# Enforce Time Slice




- CPU efficiency: 81.6% (30us quantum)

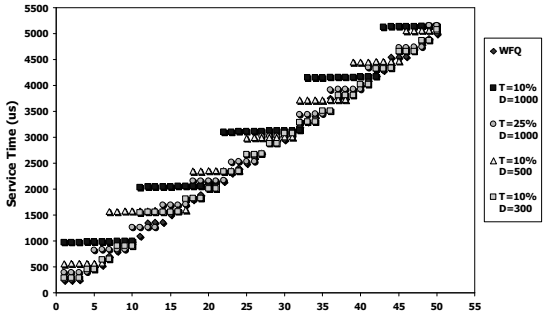


# Batching Throttle

- Scheduler Granularity: G
  - flow processes as many packets as possible w/in G
- Efficiency Index: E, Overhead Threshold: T
  - keep the overhead under T%, then 1 / (1+T) < E
- Batch Threshold: $B_i$
  - don't consider Flow $i$ active until it has accumulated at least $B_i$ packets, where $C_{sw}$ / ($B_i$ x $C_i$) < T
- Delay Threshold: $D_i$
  - consider a flow that has waited $D_i$ active



# Dynamic Control

- Flow specifies delay requirement D
- Measure context switch overhead offline
- Record average flow runtime
- Set E based on workload
- Calculate batch-level B for flow

# Packet Trace