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**

- Control Plane (BGP, RSVP, ...
- Data Plane (IP)

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

---

**Hardware-Based Router**

- Control Plane (BGP, RSVP, ...
- Data Plane (IP)
- ASIC

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

---

**NP-Based Router Architecture**

- Control Plane (packet flows)
- Data Plane (packet flows)
- IXP1200

+ 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

Pipeline Evaluation
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

<table>
<thead>
<tr>
<th>Processing</th>
<th>Memory Ops</th>
<th>Register Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic IP</td>
<td>6</td>
<td>32</td>
</tr>
<tr>
<td>TCP Splicer</td>
<td>6</td>
<td>45</td>
</tr>
<tr>
<td>TCP SYN Monitor</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>ACK Monitor</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>Port Filter</td>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>Wavelet Dropper</td>
<td>2</td>
<td>28</td>
</tr>
</tbody>
</table>

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
  - 310Kpps with 1510 cycles/packet
- StrongARM
  - 3.47Mpps w/ no VRP
  - 1.13Mpps w/ VRP bug
- MicroEngines

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
### 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

![Experiment Diagram]

- A (BE) → B
- B (QoS) → A + C
- C (QoS) → 

---
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_i / (B_i * C) < 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