Advanced Networking:
Switch Fabrics, Fast IP Routers, Network Processors, and the End-to-End Arguments
Lancaster University MSc CSM008 Unit 4

James P.G. Sterbenz
Computing Department – InfoLab21
Lancaster University
jgps@comp.lancs.ac.uk

Department of Electrical Engineering & Computer Science
Information Technology & Telecommunications Research Center
The University of Kansas
jgps@ittc.ku.edu
+1 508 944 3067

16 January 2007
Abstract

Switch and Router Architecture notes for Lancaster University Advanced Networking graduate course CSM008.

Advanced Networking

Unit 4

AN4.1 Switch fabrics
AN4.2 Fast IP routers
AN4.3 Lookup and classification
AN4.4 Packet output scheduling
AN4.5 Network processors
AN4.6 End-to-end arguments
Advanced Networking

4.1 Switch Fabrics

AN4.1 Switch fabrics
AN4.2 Fast IP routers
AN4.3 Lookup and classification
AN4.4 Packet output scheduling
AN4.5 Network processors
AN4.6 End-to-end arguments
Switch Fabric Architecture

Blocking Characteristics

- Blocking should be avoided
- Blocking characteristics (among different outputs)
  - strictly nonblocking: under all conditions
  - wide-sense nonblocking: if particular algorithm is used
  - rearrangeably nonblocking: if existing paths are rearranged
  - virtually nonblocking: with extremely low probability
Switch Fabric Architecture

Contention and Buffering

- Contention (burst collisions) in a non-blocking fabric
  - occurs when traffic destined for *same* output
  - requires buffering even for well-behaved traffic
Switch Fabric Architecture

Contention and Buffering

- Unbuffered fabric
- Internal buffers
Switch Fabric Architecture

Contestation and Buffering

- Input queueing
  - suffers from head-of-line blocking

- Output queueing
  - requires either:
    - internal speedup
    - internal expansion
Switch Fabric Architecture

Head-of-Line Blocking

- Head-of-line blocking
  - input blocking of packet destined for different output
- Output buffers solve
  - but requires switch fabric speedup
Switch Fabric Architecture

Virtual Output Queueing

- Virtual output queueing
  - parallel buffers
  - non-FIFO buffers
Switch Fabric Architecture

Single Stage: Bus as a Switch

- Simple design shared medium bus
  - point of contention: only one input active at a time
  - 2nd/3rd generation routers
    - suitable for small switches
- Multicast
  - inherent broadcast
Exercise 2:
Design Switch Fabric

• Consider fast packet switch design from:
  – course notes
  – Exercise 1
• Inputs and outputs may be arbitrarily placed
  – for topological and diagrammatic convenience
• Requirements
  – scalable in speed and number of ports
  – avoid significant blocking
• Explain
  – scalability as $O(f(n))$ where $n$ is number of ports
  – blocking characteristics
Switch Fabric Architecture
Single Stage:  Shared Memory Switch

• Simple design
  – packets written by input
  – packets read by output

• Shared memory
  – point of contention
  – speedup necessary
    • but access times not scaling with Moore’s

• Multicast
  – multiple writes or
  – multicast output demux
Switch Fabric Architecture

Single Stage: Basic 2×2 Switch Element

- **States**
  - point-to-point
    - straight
    - cross
  - multicast
- **Types**
  - buffered or unbuffered
  - self routing or externally controlled
Switch Fabric Architecture

Single Stage: Crossbar Switch

- **Crosspoint switch element**
  - electronic
    - multicast possible
  - optical MEMS
    - rotating mirror

![Crossbar Switch Diagram]
Switch Fabric Architecture

Single Stage: Crossbar Switch

• Square array of crosspoint elements
  – $O(n^2)$ growth complexity
  – reasonable for moderate $n$

• Strictly nonblocking

• Multicast
  – inherent topology
  – requires arbitration
Switch Fabrics
Multistage Switches

- Large switches built from single stage elements
  - $2 \times 2$ elements or $n \times n$ crossbars
  - $O(n \log n)$ growth complexity
- Large switches require scalable switch fabrics
  - scalability of components with number of ports
  - regular interconnection topologies
Switch Fabrics
Multistage Switches

- Example
  - self-routing delta fabric
Switch Fabric Architecture

Native Multicast

- Multicast support in network conserves bandwidth
- Multicast support in switches
  - prevents multiple transmissions of a single multicast packet
  - reduces delay in subsequent packets
Switch Fabric Architecture

Native Multicast: Crossbar

- Crossbar multicast
  - switch points duplicate
  - except MEMS mirrors
  - output arbitration needed to resolve incompatible output sets
  - fanout splitting: not all copies need xmit simultaneously
Switch Fabric Architecture

Native Multicast: Multistage

- Multistage multicast
  - switch elements duplicate late
    - $2 \times 2$: replicate state
    - $n \times n$ internal
  - alternative: recirculation
  - multiple fabrics
    - copy stages
    - $c_id$ translation
    - routing stages

Copy stages:

Translate:

Routing stages:
Advanced Networking

4.2 Fast IP Routers

AN4.1 Switch fabrics
AN4.2 Fast IP routers
AN4.3 Lookup and classification
AN4.4 Packet output scheduling
AN4.5 Network processors
AN4.6 End-to-end arguments
Fast Datagram Switches

Motivation

- Connection-oriented fast packet switching
  - emerged in ATM standards, but ATM failed
- IP became waist of global network infrastructure
  - increased processing capability enabled fast IP lookups
  - apply fast packet switching to IP datagram forwarding
- Compelling need to apply fast packet switching to IP
  - Moore’s law enabled
Fast Datagram Switches

Architecture

- Fast packet switch core
- Input processing
  - IP lookup
  - Packet classification
- Output processing
  - Packet scheduling
    - Fair queueing
### Datagram Routing and Switching

**Example 5.6 IP Packets**

<table>
<thead>
<tr>
<th>04</th>
<th>hl</th>
<th>TOS</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>identification</td>
<td>flags</td>
<td>frag offset</td>
</tr>
<tr>
<td>TTL</td>
<td>protocol</td>
<td>header checksum</td>
<td></td>
</tr>
<tr>
<td>source address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>options</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[variable length]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[variable length]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>06</th>
<th>class</th>
<th>flow label</th>
</tr>
</thead>
<tbody>
<tr>
<td>payload length</td>
<td>next header</td>
<td>hop lim</td>
</tr>
<tr>
<td>source address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>extension header(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[variable length]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[variable length]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fast Datagram Switches

**Throughput**

- Packet processing rate critical
  - packet processing must sustain at least average rate
  - critical path must sustain peak line rate for min size packets
Advanced Networking

4.3 Lookup and Classification

AN4.1 Switch fabrics
AN4.2 Fast IP routers
AN4.3 Lookup and classification
AN4.4 Packet output scheduling
AN4.5 Network processors
AN4.6 End-to-end arguments
Fast Datagram Switches
IPv4 Address Assignment

- IP addresses not randomly assigned to hosts

  why?
Fast Datagram Switches
IPv4 Address Assignment

• IP addresses not randomly assigned to hosts
  – every table would have to contain every Internet host
    • billions of entries and would require exact match lookup
Fast Datagram Switches

IPv4 Address Hierarchy

- IP addresses assigned *hierarchically*
  - address aggregation dramatically improves scalability
  - forwarding table only needs to contain *network address*
  - routing advertisements only contain network address prefix

```
ISP_A
200.23.16.4
200.23.16.45
200.23.16.12
199.31.0.4

ISP_B

Tier1_X
200.23.16
199.31.0
```

16 January 2007
Advanced Networking: Fabrics and IP Routers
Fast Datagram Switches
IPv4 Class-Based Addressing Hierarchy

<table>
<thead>
<tr>
<th>Networks</th>
<th>Hosts</th>
<th>Class</th>
<th>Network Address</th>
<th>Host Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>16M</td>
<td>A</td>
<td>0</td>
<td>host</td>
</tr>
<tr>
<td>16K</td>
<td>64K</td>
<td>B</td>
<td>10</td>
<td>host</td>
</tr>
<tr>
<td>2M</td>
<td>256</td>
<td>C</td>
<td>110</td>
<td>host</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td>1110</td>
<td>multicast</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E</td>
<td>1111</td>
<td>reserved</td>
</tr>
</tbody>
</table>

- Divide IP address into 3 level hierarchy
  - class, network address, host address
  - byte aligned
  - simple IP address lookup (3 major cases)
  - class D for multicast addresses

Lecture NR
Fast Datagram Switches
IPv4 Subnets

16K networks × 64 subnets × 1024 hosts

- **Subnets** [RFC 0950 / STD 0005]
  - originally way to divide address class within organisation
  - example: 6b subnet to class B
  - subnet mask

- **Hosts in subnet** share upper IP address bits
  - natural to cluster similar IP addresses
  - efficient IP routing to subnet
  - switched layer 2 LAN with no layer 3 routing  

Lecture LL
Fast Datagram Switches
IPv4 Class-Based Addressing Problems

- Principle behind division
  - A: very large network providers
  - B: large organisations
  - C: LANs

- Reality: rigid structure
  - doesn’t match all organisations perfectly
  - doesn’t match many organisations well
    - especially class B: “three bears problem”

- Inefficient partitioning of address space
  - large fraction of unusable addresses
  - imminent exhaustion of IP address space led to...
Fast Datagram Switches
IPv4 Classless Addressing (CIDR)

- **CIDR**: classless interdomain routing [RFC 1519]
  - eliminate assignment of IP address blocks by class
  - $b_7 b_6 \cdot b_5 b_4 \cdot b_3 b_2 \cdot b_1 b_0 /x$
  - x-bit prefix = arbitrary number of network bits
  - example: 11001000 00010111 00010000 00000000
  - 200.23.16.0/23

- Service providers get variable IP block
  - based on need from RIR (or NIR)

- Significant improvement in IP address use
  - at the cost of significant increase in complexity of IP lookup

*how?*
Fast Datagram Switches
IPv4 Classless Addressing (CIDR)

- **CIDR**: classless interdomain routing [RFC 1519]
  - eliminate assignment of IP address blocks by class
  - $b_7b_6 . b_5b_4 . b_3b_2 . b_1b_0 /x$
  - x-bit prefix = arbitrary number of network bits
  - example: 11001000 00010111 00010000 00000000
  - 200.23.16.0/23

- Service providers get variable IP block
  - based on need from RIR (or NIR)

- Significant improvement in IP address use
  - at the cost of significant increase in complexity of IP lookup
  - IP lookup is *longest prefix match*
Fast Datagram Switches

Software IP Lookup

- Longest prefix match
- Critical parameters
  - worst case lookup time
    - brute force: $O(\log_2 n)$
    - $n$ tens of thousands
  - memory required
  - forwarding table update time

### Table

<table>
<thead>
<tr>
<th>prefix</th>
<th>$P_{out}$</th>
<th>$f_{state}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10100*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- hop count
- checksum fix

- payload
  - 101 011 01

- $P_{out}$
  - 101 011 01
  - payload
Fast Datagram Switches

Software IP Lookup Example: Trie

- Many algorithms
- Example: trie
  - sparse binary tree
  - valid prefixes are root
  - lookup time $O(a)$
    - $a =$ number of address bits
Fast Datagram Switches

Hardware IP Lookup

- **Ternary CAM**
  - 1, 0, x (don’t care)
  - expensive and complex
    - relative to RAM

- **Simultaneous match**
  - lookup time constant
    - $O(1)$

<table>
<thead>
<tr>
<th>prefix</th>
<th>$P_{out}$</th>
<th>$f_{state}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00xxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001xxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001xx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101xx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101xx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10100x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11xxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111xxx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- hop count
- checksum fix

- 101 011 01
  - payload

- $P_{out}$ 101 011 01
  - payload
Fast Datagram Switches
Hardware-Assisted Memory IP Lookup

- Multistage lookup
  - conventional SRAM
  - worst case lookup time
- $O(s)$ number of stages

```
101 011 01
```

```
p_{out} / index
```

```
short prefix table
```

```
101 011
```

```
2^{p-1}
```

```
101 011 block
```

```
p_{out}
```

```
long prefix table
```

```
p_{out}
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
```

```
101 011 01
```

```
payload
Fast Datagram Switches

Hardware vs. Software

• Input and Output processing tradeoff
  – custom hardware generally faster
  – network processor software more flexible
Fast Datagram Switches
Packet Classification

- Packet classification determines how packet treated
  - QoS or diffserv
  - policy based routing
  - security and DOS protection (e.g. firewalls)
  - layer 4 and 7 switching
  - active network processing
- Before queueing to meet most stringent delay class
Fast Datagram Switches
Packet Classification

- Multidimensional classification
  - policies may be hierarchal or overlap
  - precedence rules needed
- More complex than longest prefix match
- Hardware and software implementation tradeoffs
Fast Datagram Switches
Classification Strategies

- **Software**
  - algorithms that minimise instruction count
  - data structures that minimise memory accesses

- **Custom hardware**
  - functional blocks that store and manipulate classifiers
  - memories that minimise access time
Fast Datagram Switches
Hardware Classification Strategies

- Custom hardware
  - functional blocks that store and manipulate classifiers
  - memories that minimise access time
    - TCAM: ternary content addressable memory
    - challenge: longer match strings than for IP lookup
Fast Datagram Switches
Software Classification Strategies

• **Software**
  – algorithms that minimise instruction count
  – data structures that minimise memory accesses

• **Various strategies**
  – two dimensions
    • two dimensional tries
    • grid of tries prevents backtracking via precomputation
  – multidimensional
    • significantly more complex
4.4 Packet Output Scheduling

AN4.1 Switch fabrics
AN4.2 Fast IP routers
AN4.3 Lookup and classification
AN4.4 Packet output scheduling
AN4.5 Network processors
AN4.6 End-to-end arguments
Fast Datagram Switches

Output Scheduling

- Output scheduling
  - guaranteed service packets transmitted to meet contract
  - fair service among best effort flows
- Fair queueing
- Per-flow queueing
  - isolates flows from one another
Best Effort and Differentiated Service

Motivation

- Best effort *service* is provided by the Internet
  - attempt to eventually deliver most packets
- Motivation: differentiated service
  - allow service differentiation among classes of users
  - limit complexity of mechanisms
- Examples
  - Internet (best effort)
  - DiffServ differentiated service
Best Effort and Differentiated Service
Closed-Loop End-to-End Congestion Control

- Closed loop congestion
  - feedback from network necessary if no hard reservations
Best Effort and Differentiated Service

Congestion Control

- Keep queues from building
  - impacts entire port unless per flow queueing
  - bound steady state size of queues
  - throttle transmitter when necessary
    - RED: random early detection
    - ECN: explicit congestion notification
Best Effort and Differentiated Service

Output Scheduling

- **Output scheduling**
  - guaranteed service packets transmitted to meet contract
  - fair service among best effort flows
- **Fair queueing**
- **Per-flow queueing**
  - isolates flows from one another
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)

*operation?
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling

FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling

FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling

FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling

FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling

FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling

FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
- FIFO (first in first out)
  - send in order of arrival to queue when link available
• FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling

FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
Packet Scheduling

FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available

What happens when the queue fills?
Packet Scheduling
FIFO Queueing

- **FIFO** (first in first out)
  - send in order of arrival to queue when link available
- **Discard policy**: when full queue
  - **tail drop**: drop arriving packet
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
- Discard policy: when full queue
  - tail drop: drop arriving packet

*alternatives?*
Packet Scheduling
FIFO Queueing

- FIFO (first in first out)
  - send in order of arrival to queue when link available
- Discard policy: when full queue
  - tail drop: drop arriving packet
  - priority: drop/remove on priority basis
  - random: drop/remove randomly  *why? Lecture TL*
Packet Scheduling
FIFO Queueing

Advantages and Disadvantages?
Packet Scheduling

FIFO Advantages

- Simplest discipline
  - very simple software management
  - very simple hardware
    - shift register is a series of flip-flops
Packet Scheduling

FIFO Disadvantages

- **Simplest discipline**
  - very simple software management
  - very simple hardware
    - shift register is a series of flip-flops

- **All packets treated equally**
  
  *implications*?
Packet Scheduling
FIFO Disadvantages

- **Simplest discipline**
- **All packets treated equally**
  - flows not necessarily treated *fairly*
    - important for best effort service
  - flows not discriminated by service class
    - e.g. gold vs. silver vs. bronze
  - variable packet size not accounted for
    - small packets wait behind large packets
      (recall message switching)
Packet Scheduling

FIFO Disadvantages

- Simplest discipline
- All packets treated equally
  - flows not necessarily treated *fairly*
    - important for best effort service
  - flows not discriminated by service class
    - e.g. gold vs. silver vs. bronze
  - variable packet size not accounted for
    - small packets wait behind large packets
      (recall message switching)

*Alternatives?*
Packet Scheduling
Priority Queueing

- Priority queueing
  - separate traffic classes by priority
  - serve highest priority first
  - provides *relative* service differentiation

*Implementation?*
Packet Scheduling

Priority Queueing

- Priority queueing
  - FIFO queue for each traffic class
- Packet classifier sorts into proper queues
  - e.g. source address, port, or QoS field
Packet Scheduling

Priority Queueing

- Priority queueing
  - FIFO queue for each traffic class
- Packet classifier sorts into proper queues
  - e.g. source address, port, or QoS field
Packet Scheduling
Priority Queueing

- Priority queueing
  - FIFO queue for each traffic class
- Packet classifier sorts into proper queues
  - e.g. source address, port, or QoS field
Packet Scheduling

Priority Queueing

- Priority queueing
  - FIFO queue for each traffic class
- Packet classifier sorts into proper queues
  - e.g. source address, port, or QoS field
Priority queueing
- FIFO queue for each traffic class

Packet classifier sorts into proper queues
- e.g. source address, port, or QoS field
Packet Scheduling
Priority Queueing

- Priority queueing
  - FIFO queue for each traffic class
- Packet classifier sorts into proper queues
  - e.g. source address, port, or QoS field
Packet Scheduling
Priority Queueing

- Priority queueing
  - FIFO queue for each traffic class
  - queues served in priority order
- Packet classifier sorts into proper queues
  - e.g. source address, port, or QoS field
- Higher priority traffic classes receive better service
Packet Scheduling
Priority Queueing

Advantages and Disadvantages?
Packet Scheduling
Priority Queueing

- Advantages
  - relatively simple discipline
  - priority queues
    - multiple physical FIFO queues
    - memory and pointer management to partition between queues
Packet Scheduling
Priority Queueing

• Advantages
  – relatively simple discipline

• Disadvantages
  – provides service differentiation
  – but lower priorities may get no service: starvation
  – no mechanism for proportional service

*alternatives?
Packet Scheduling
Round Robin Queueing

- **Round robin**
  - FIFO queue for each class or flow
  - attempt to provide fair service among best-effort flows
- **Packet classifier** sorts into proper queues by *flow id*
- **Packets are served** *round-robin*
  - cyclically taking turns
Packet Scheduling
Round Robin Queueing

Advantages and Disadvantages?
Packet Scheduling
Round Robin Advantages

• Simple discipline
  – very simple software management
  – very simple hardware
    • round robin is a counter the points to a particular queue
  – in per flow round-robin senders *are* treated equally
    • in terms of number of packets served
Packet Scheduling
Round Robin Disadvantages

- Simple discipline
- All classes or flows treated equally

implications?
Packet Scheduling
Round Robin Disadvantages

- **Simplest discipline**
- **All classes or flows treated equally**
  - senders not necessarily treated *fairly*
  - variable packet size not accounted for
    - small packets wait behind large packets (recall message switching)
  - senders not discriminated by service class
    - e.g. gold vs. silver vs. bronze
Packet Scheduling
Round Robin Disadvantages

- **Simplest discipline**
- **All classes or flows treated equally**
  - senders not necessarily treated *fairly*
  - variable packet size not accounted for
    - small packets wait behind large packets
      (recall message switching)
  - senders not discriminated by service class
    - e.g. gold vs. silver vs. bronze

*Alternatives?*
Packet Scheduling
Weighted Round Robin Queueing

- Weighted round robin
  - FIFO queue for each flow or class (flow aggregates)
- Packet classifier sorts into proper queues
  - by flow id or traffic class
- Packets are served cyclically: *round-robin*
  weights determine relative service: benefits of priority and RR
Packet Scheduling

Fair Queueing

- **Motivation**
  - fairness among best effort flows

- **GPS: generalised processor sharing**
  - ideal fluid model of sharing among flows

- **Bit-by-bit round robin**
  - would serve a bit at a time from each queue
  - fair regardless of packet sizes
  - not practical to implement directly
Fair Queueing

Weighted Fair Queueing

- WFQ (weighted fair queueing)
  - FIFO queue flow or flow class
- Simulates bit-by-bit round robin
  - schedules packets based on simulated departure time
  - relatively complex to implement
Deficit round robin: simple hardware implementation
- FIFO queue for each class or flow
- each flow given a quantum of service

Packets are served *round-robin*
- as long as packet can be served within quantum + deficit
- unused service (with packet waiting) added to deficit
- provides *long term fairness*
Fair Queueing
Overengineering vs. Optimality

- Traffic management can be very complex
- Overengineering can dramatically simplify
  - network engineering or traffic engineering to help
- Support for mixed traffic
  - distinct networks (1\textsuperscript{st} and 2\textsuperscript{nd} generations through 1980s)
  - virtual network partitioning
  - differentiated services: coarse grained service grades
  - integrated services: fine grained traffic classes
    - e.g. ATM-TM and intserv with RSVP
Per Flow Queueing

Hardware Complexity

- **Moore's law**
  - dramatically increases the processing capability per packet

- **IntServ**
  - dismissed in the late 1990s as not scalable to core
    - now need 1M flows/router at 40Gb/s
  - per flow queuing was considered too complex
  - a decade later hardware experts agree that it *can* be done
    - using lower complexity approximations (e.g. DRR)
    - but the community seems to have dismissed it
Advanced Networking

4.5 Network Processors

AN4.1 Switch fabrics
AN4.2 Fast IP routers
AN4.3 Lookup and classification
AN4.4 Packet output scheduling
AN4.5 Network processors
AN4.6 End-to-end arguments
Network Processors
Overview and Motivation

• Flexibility of programmability
  – embedded controller software
  – reduce design cycle for vendors
  – field upgradeability and migration for service providers

vs.

• Performance of specialised hardware
  – custom VLSI and ASICS

Middle ground for networking?
Network Processors
Overview and Motivation

- Flexibility of programmability
  - embedded controller software
  - reduce design cycle for vendors
  - field upgradeability and migration for service providers

vs.

- Performance of specialised hardware
  - custom VLSI and ASICS

- Middle ground:
  - network processors: special type of embedded controller
  - programmable hardware: FPGAs
Network Processors
Overview and Motivation

• Network processors
  – designed for high-performance packet processing
  – compromise among cost, flexibility, performance

• Advantages of input/output programmability
  – reduce switch design and debug time
  – allow field upgrades
  – allow service providers to deploy new protocols/services
  – enable per port active networking
Network Processors
Use and Functionality

- Use for network processors
  - switch and router line cards
  - host-network interfaces

- Example NP functionality
  - header parsing and manipulation
  - longest prefix match lookup
  - multidimensional classification
  - error control (CRC/checksum generation and checking)
  - scheduling
    - policing, traffic shaping, fair queueing
Switch Implementation
Programmable Input/Output Processing

- NPs replace
  - input processing
    - lookup
    - classification
  - output processing
    - scheduling
  - additional
    - encryption
    - policing
  - large buffers
    - off NP chip
Network Processors

Typical Organisation

- **Embedded control processor core + memory**
  - performs overall NP and packet dispatching
    - e.g. PowerPC, XScale, ARM

- **Packet processing engines**
  - typically called *μengines* (Intel) or *picoprocessors* (IBM)
  - interconnection between packet *μ*engines
    - fixed or flexible; parallel or pipeline
  - packet buffers (shared or per *μ*engine)

- **Functional units** (shared or per *μ*engine)
  - hardware assists for critical-path functions
    - e.g. crypto, FEC, CRC, lookup, classification, scheduling
NP Standard Interfaces

Motivation

- Multiple chip vendors
- Multiple switch and network-interface building blocks
  - network processors
  - search engines and TCAMs
  - classifiers
  - schedulers
  - switch fabric chips
  - queues
  - serdes (serialiser/deserialiser)
  - FIFOs, queues, buffers

*Problem*?
Standard NP Interfaces

Motivation

- Multiple chip vendors
- Multiple switch and network-interface building blocks
  - with different internal architectures
  - with different external interfaces
- Standard interfaces permit multivendor solutions
  - common in PC world
  - new to switch and network interface domain
- Industry consortiums: implementation agreements
  - NPF: Network Processing Forum
  - OIF: Optical Internetworking Forum
Standard NP Interfaces
Line Card and NIC Interfaces

- **SxI**: common electrical characteristics
- **SPI** (system packet interface)
  - interface: link framer & switch fabric or host interconnect
- **SFI** (serdes (serial-deserial) – framer interface)
  - interface among components in physical/link layer pipeline
- **NPSI** (network processor streaming interface)
  - network processing element (NPE) & ...
  - framing element, another NPE, or switch fabric
- **LA** (lookaside) interface
  - between NPE & memory or coprocessor
Standard NP Interfaces
Line Card and NIC Interfaces

- Optical receive and transmit
- Data status: [16b] [2 or 4b]
- SONET, OTN, Ethernet, 802.16
- FEC
- Serdes 1:16
- Framer
- Memory co-processor
- Switch fabric

NPE
- NPSI [NPE-NPE]
- SPI [NPE-Framer]
- SFI
- LA bus

Advanced Networking: Fabrics and IP Routers
NP Design Alternatives & Tradeoffs

Degree of Specialisation

- Set of relatively general purpose processors
  - e.g. Intel IXP

- Highly specialised functional units
  - checksum, lookup, classification, crypto etc.
  - in addition to general purpose packet processors
  - e.g. IBM NP4GS3

Tradeoffs?
NP Design Alternatives & Tradeoffs
Degree of Specialisation

• Set of relatively general purpose processors
  – e.g. Intel IXP
• Highly specialised functional units
  – checksum, lookup, classification, crypto etc.
  – in addition to general purpose packet processors
  – e.g. IBM NP4GS3
• Tradeoffs
  – ease of programming vs. performance
NP Design Alternatives & Tradeoffs

Scalability: Performance Measures

*Performance measures?*
NP Design Alternatives & Tradeoffs

Scalability: Performance Measures

- Performance measures
  - "line speed" or data rate
    - typically in SONET equivalents: 2.5, 10, 40 Gbps
  - packet processing rate in [packets/sec]
    - maximum necessary derived from rate and 40B packets
  - latency
    - generally not an issue except for very reach networks
NP Design Alternatives & Tradeoffs

NP Scalability: Techniques

*Scaling of a single NP?*
NP Design Alternatives & Tradeoffs

NP Scalability: Techniques

- Scaling of a single NP
  - higher clock rate
  - functional units

more?
NP Design Alternatives & Tradeoffs

NP Scalability: Techniques

- Scaling of a single NP
  - higher clock rate
  - functional units
    - greater number
    - more specialised
  - packet pipeline
    - each µengine stage operates on all packets
    - each µengine stage performs a fraction of packet processing
  - packet parallelism
    - each µengine operates on fraction of packets
    - each µengine performs processing functions
  - combination of techniques
NP Design Alternatives & Tradeoffs

NP Scalability: Techniques

*Scaling performance with multiple NPs?*
NP Design Alternatives & Tradeoffs

NP Scalability: Techniques

- Scaling performance with multiple NPs
  - specialised coprocessors
  - packet pipeline
    - each µengine stage operates on all packets
    - each µengine stage performs a fraction of packet processing
  - packet parallelism
    - each µengine operates on fraction of packets
    - each µengine performs processing functions
  - combination of techniques
Network Processors

NP.4 Example Network Processors

NP.1 Overview and Motivation
NP.2 Standard interfaces and system architecture
NP.3 Design alternatives and tradeoffs
NP.4 Example network processors
NP.5 Higher layer and active processing
NP.6 Outlook and future prospects
### Network Processors Examples

<table>
<thead>
<tr>
<th></th>
<th>Interface Capacity</th>
<th>Packet Processors</th>
<th>Co-processors</th>
<th>Control Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td></td>
<td>µengines</td>
<td></td>
<td>ARM</td>
</tr>
<tr>
<td>IXP 1200</td>
<td>OC-12</td>
<td>6 @ 166 MHz</td>
<td>hash</td>
<td>StrongARM 232 MHz 8MB + 256MB+ external</td>
</tr>
<tr>
<td></td>
<td>14Mpps 40B</td>
<td>4K × 40b I mem</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IXP 2400</td>
<td>OC-48</td>
<td>8 @ 600 MHz</td>
<td>hash</td>
<td>Xscale 600MHz 32K$ [32+2]KD$ 2GB+32MB external</td>
</tr>
<tr>
<td></td>
<td>156Mpps 40B</td>
<td>4K × 40b I mem</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pipeline interconnection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IXP 2850</td>
<td>OC-192</td>
<td>16 @ 1.4 GHz</td>
<td>hash crypto</td>
<td>Xscale 700MHz 32K$ [32+2]KD$ Ext 6GB+64MB ext</td>
</tr>
<tr>
<td></td>
<td>60Mpps 40B</td>
<td>4K×40b I+996w D mem</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pipeline interconnection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-Port Motorola C-5</td>
<td>4 × OC-12</td>
<td>16 @ 266 MHz</td>
<td>queue, buffer, fabric, table lookup</td>
<td>XP 266 MHz 32KI + 32KD mem</td>
</tr>
<tr>
<td></td>
<td>15Mpps</td>
<td>64KI + 12KD mem</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pipeline/parallel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM PowerNP 4GS3</td>
<td>OC-48</td>
<td>8 × 2 @ 133 MHz</td>
<td>8×10 e.g. tree search, checksum</td>
<td>PowerPC 405 133 MHz 16K$ + 16KD$</td>
</tr>
<tr>
<td></td>
<td>4.5Mpps</td>
<td>picoprocessors</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Others are proprietary and require NDA for details.
Network Processors
Example: Intel IXP 2800/2850

- Moderately specialised embedded controller
  - organised for packet processing
  - XScale ARM core controller + 16 microengines
  - some functional units
    - hash, crypto (AES, DES, SHA-1 on 2850)
    - CRC, pseudorandom generator per microengine

- Programming
  - requires *significant* knowledge of hardware organisation
Network Processors
Example: Intel IXP 2800/2850

- Processors
  - XScale core @ 700MHz
  - 16 microengines
- 16KB scratchpad mem
- Controllers
  - PCI
  - SRAM, DRAM
- Media/switch interface
- Hash unit
- Crypto (2850 only)
Network Processors
Example: Intel IXP 2800/2850 Microengine

- 16 @ 1.4 GHz
- 2 × 128 GPRs (A,B)
- ALU (A • B)
- 640 longword local mem
- {S|D}RAM regs
- 16 × 32+4b CAM
- CRC-{16,32} unit
Network Processors
Example: IBM PowerNP 4GS3

- Highly specialised embedded controller
  - organised for packet processing; many functional units
    - PowerPC 405 core @ 133 MHz
    - completion unit, dispatch unit, control store arbiter
      enqueue/dequeue unit, traffic scheduler/shaper
    - 8 DPPUs (dyadic protocol processor units)
      - 2 picoprocessors each @ 133 MHz
      - 10 coprocessors: checksum, bus control, ext coprocessor control,
        counter, data store, enqueue, policer, string copy, tree search,
        semaphore manager

- Programming
  - requires significant knowledge of hardware organisation

IBM has sold the NP to Hifn – datasheets and manuals no longer online
Network Processors
Example: IBM PowerNP 4GS3

- **Processors**
  - PowerPC @ 133 MHz
  - 16K I + 16K D cache
  - 8 DPPUs

- **Memory**
  - 32 KB instruction
  - 128 KB SRAM buffer
  - 113 KB SRAM ctl store

- **Interface controllers**
  - switch ingress+egress
  - SRAM, SDRAM
  - PCI

- **Coprocessors/accelerators**
  - traffic scheduler/shaper
  - enqueue/dequeue
  - completion, dispatch, ctl store
Network Processors
Example: IBM PowerNP 4GS3 EPC

- 8 DPPUs
  - 2 picoprocessors
    - 133 MHz
    - 16|32 GPRs
    - ALU
  - 10 coprocessors
- $8 \times 4KB$ memory
Network Processors
Example: IBM PowerNP 4GS3 DPPU
Advanced Networking

4.6 End-to-End Arguments

AN4.1 Switch fabrics
AN4.2 Fast IP routers
AN4.3 Lookup and classification
AN4.4 Packet output scheduling
AN4.5 Network processors
AN4.6 End-to-end arguments
E2E Functions and Mechanisms
End-to-End Semantics

- **Hop-by-hop functions**
  - link layer protocols
  - link compression and FEC
  - network forwarding
  - link / subnet error control
  - embedded protocols (e.g. protocol boosters)

- **End-to-end functions**
  - transport protocols
  - source routing
  - end-to-end encryption
  - session protocols
  - application protocols
E2E Functions and Mechanisms
End-to-End Argument

- Hop-by-hop function composition $\neq$ end-to-end
- Examples
  - HBH encryption has data in clear inside network nodes
  - HBH link error control doesn’t cover network layer errors
E2E Functions and Mechanisms
Hop-by-Hop Performance Enhancement

- E2E functions replicated HBH if performance benefit
- Example
  - per link error control in high bandwidth-delay networks
    reduce E2E control loop when error

**Hop-by-Hop Performance Enhancement Corollary**

*It is beneficial to duplicate an end-to-end function hop-by-hop if the result is an overall (end-to-end) improvement in performance.*
End-to-End vs. Hop-by-Hop Performance Enhancement Example

- E2E vs. HBH error control for reliable communication
  - E2E argument says error control *must* be done E2E
    - e.g. E2E ARQ (error check code and retransmit if necessary
  - but should HBH error control *also* be done?

100 m wireless LAN
Univ. Kansas

15 000 km fiber WAN

100 m fiber LAN
Univ. Sydney
End-to-End vs. Hop-by-Hop Performance Enhancement Example

- E2E vs. HBH error control for reliable communication
  - E2E argument says error control *must* be done E2E
    - e.g. E2E ARQ (error check code and retransmit if necessary
    - but should HBH error control *also* be done?

- Effect of high loss rate on wireless link
  - ~300 ms RTT retransmission for every corrupted packet
- Error control on wireless link reduces to ~1μs
  - shorter control loop results in dramatically lower E2E delay
E2E Functions and Mechanisms

E2E Argument Misinterpretations

- **E2E-only**
  - do not replicate E2E services or features HBH
  - violated HBH performance enhancement corollary

- **Everything E2E**
  - implement as many services or feature E2E as possible
  - misstatement of Internet design philosophy: simple stateless network for resilience and survivability
E2E Functions and Mechanisms
Meaning of “In the Network”

• **Functional** (general use in this tutorial)
  – layers 1 – 3
    • physical
    • MAC
    • link
    • network

• **Topological**
  – colocated with network nodes

• **Administrative**
  – owned by network service provider