Slide 0

Support for Quality of Service in High-Speed Switch Architectures

Jennifer Rexford

`http://www.eecs.umich.edu./~jrexford`
Real-Time Computing Laboratory
University of Michigan

Slide 1

Outline

- Background and motivation
- Buffer architectures
- Link scheduling
- Traffic shaping
- Hierarchical sorting schemes
- Conclusions

Real-Time Computing Laboratory
The University of Michigan

Slide 2

Emerging High-Speed Networks

Network technology trends

- High-speed links (e.g., 155 megabits/second; 1 gigabit/second)
- Small data transfer unit (e.g., 53-byte ATM cell)

→ cell arrival/departure every few microseconds (or faster)

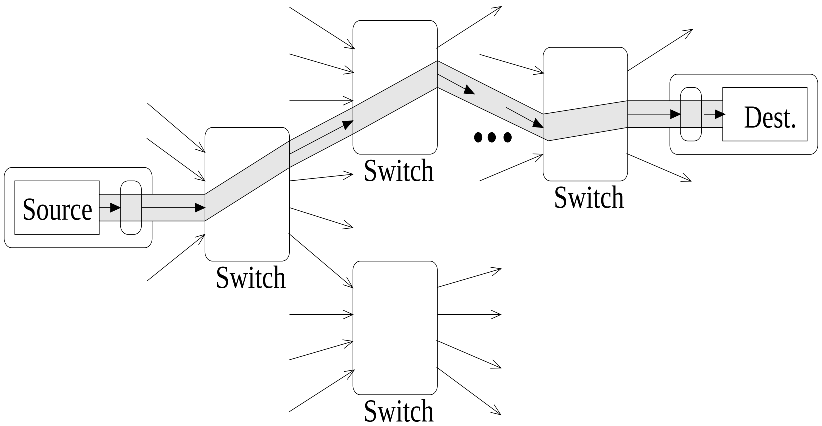
Network traffic sharing a single link

- Thousands of connections
- Diverse burst and bandwidth parameters (video, audio, data)
- Diverse performance requirements (delay, loss, throughput)

→ effective policies for managing bandwidth and buffer resources

Slide 3

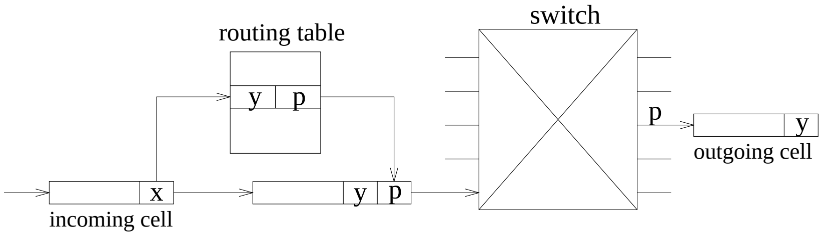
High-Speed Network of Switches



Real-Time Computing Laboratory
The University of Michigan

Slide 4

Switch Architecture



- **Traffic shaping:** hold back excess traffic
  (prevent overload at downstream switches)
- **Buffer management:** store transient congestion in switch
  (avoid loss, particularly for loss-sensitive traffic)
- **Link scheduling:** arbitrate access to bandwidth
  (guarantee performance; preserve traffic parameters)

Slide 5

Buffer Architectures

Supporting multiple high-speed input and output links requires:

- Large memory bandwidth (e.g., wide buffers; fast RAM)
- Small, bounded number of operations per cell arrival/departure

Architectural issues

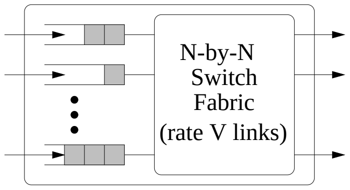
- Location of buffers: input vs. output ports
- Degree of sharing: partitioned, fully shared, or hybrid
- Per-connection state: buffer thresholds, logical FIFOs

Implementation issues

- Memory speed and size: number of cells, required bandwidth
- Control complexity: data structures, per-cell overheads

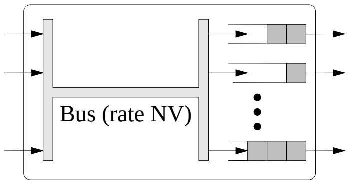
Slide 6

Input Queueing



- Pro: memory bandwidth V
- Con: head-of-line blocking; switch fabric complexity
- Fix: window selection $w > 1$; multistage networks

Output Queueing

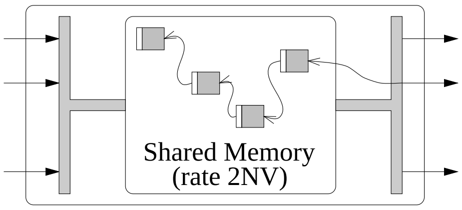


- Pro: no head-of-line blocking; simpler multicast
- Con: memory bandwidth NV
- Fix: knockout with $L < N$

Real-Time Computing Laboratory
The University of Michigan

Slide 7

Shared Memory Switch



Performance issues:

- Pro: better memory utilization
- Con: complex implementation; memory bandwidth $2NV$; intrusion between links
- Fix: per-link thresholds; overwrite schemes

Implementation:

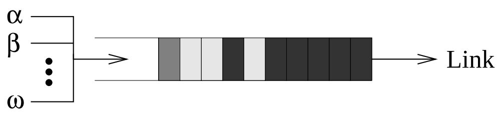
- Bus to multiplex cells to shared memory
- Logical queue for each output link (e.g., head/tail pointers)
- Idle-address FIFO

Real-Time Computing Laboratory
The University of Michigan

Slide 8

Limitations of FIFO Link Scheduling

FIFO scheduling causes *interference* between connections:



Link sharing goals:

- Guaranteed minimum bandwidth for each connection
- Preservation of connection traffic parameters
- Bounded access delay for high-priority connections
- High link utilization
- "Fair" division of excess link bandwidth

Slide 9

Link Scheduling

- **First-in-first-out scheduling**
  - Simple implementation
  - Guarantees bandwidth only on large time scale
- **Priority arbitration between traffic classes**
  - Relatively simple implementation
  - Protection between traffic classes, but not connections
- **Frame/template scheduling**
  - Connections have guaranteed slots in frame/template
  - Large frame/template: large delay, jitter, buffer space
  - Small frame/template: coarser bandwidth granularity

Slide 10

Link Scheduling (continued)

- **Fair queueing algorithms**
  - Connections receive bandwidth on relatively small time scale
  - Provable throughput bounds (consequently, delay bounds)
  - Traditionally hard to implement (priority queue)
- **Deadline-based scheduling**
  - Flexible specification of bandwidth and delay requirements
  - Provable delay and throughput bounds
  - Traditionally hard to implement (priority queue)

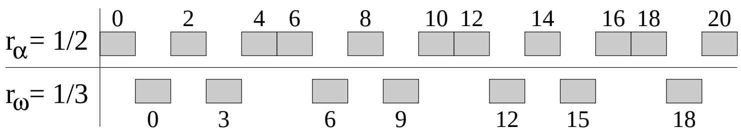
Real-Time Computing Laboratory
The University of Michigan

Slide 11

Fair Link Scheduling

- Abstraction of continuous bandwidth to each connection
- Division of bandwidth according to connection weights r_α
- Sharing of excess bandwidth amongst *backlogged* connections

e.g., weights $r_\alpha = 1/2$ and $r_\omega = 1/3 \Longrightarrow 3/5 : 2/5$ split

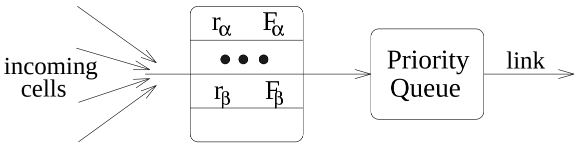


Real-Time Computing Laboratory
The University of Michigan

Slide 12

Implementing Fair Link Scheduling

- **Recurrences:** compute cell keys F based on $1/r$, resetting keys for idle connections
- **Cell sorting:** transmit cell with smallest key

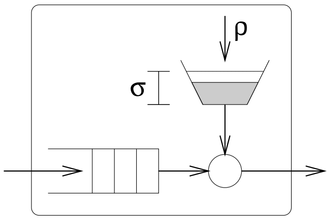


- **Weighted fair queueing**: $F = \max\{F, R(t)\} + 1/r$
- **Virtual clock**: $F = \max\{F, t\} + 1/r$
- **Self-clocked fair queueing**: $F = \max\{F, F^{serv}\} + 1/r$

Real-Time Computing Laboratory
The University of Michigan

Slide 13

Leaky-Bucket Traffic Shaping



- Connection burstiness
  → bucket size σ
- Connection bandwidth
  → token generation rate ρ
- Buffer for storing early cells

Scalable architecture

- **Recurrences**: compute *conformance time* t_c of incoming cell, based on connection's leaky-bucket status
- **Cell sorting**: transmit cell with smallest conformance time, as long as $t_c \leq t$ (non-work-conserving)
- **Buffer architecture**: share cell memory amongst connections

Real-Time Computing Laboratory
The University of Michigan

Slide 14

Algorithm to Compute t_c

$X = X + 1/\rho$;	`// Estimated cell arrival time`
`if` $(X \le t)$	`// Full token bucket: reset` X
$\quad t_c = X = t$;	
`else if` $(X \le t + \sigma/\rho)$	`// Partially full token bucket`
$\quad t_c = t$;	
`else`	`// Empty bucket: delay cell`
$\quad t_c = X - \sigma/\rho$;	

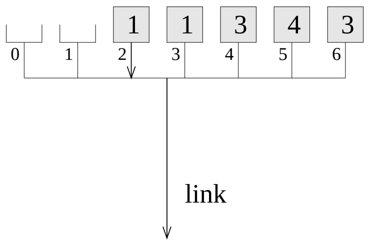
For peak-rate shaping (cell *spacing*),

$$t_c = X = \max\{X + 1/\rho, t\}$$

Real-Time Computing Laboratory
The University of Michigan

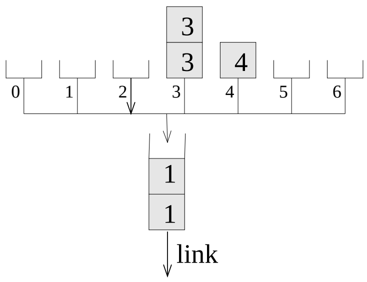
Slide 15

CNET Spacer Controller



– at most one cell per bin
– logic to locate first empty bin at (or after) t_c
– sequence through bins at link rate and transmit

Siemens Spacing Policer



– linked list in each bin
– cell joins bin t_c (enqueue cell in bin's linked list)
– concatenate bins into transmission FIFO

Real-Time Computing Laboratory
The University of Michigan

Slide 16

Scalability of Cell Sorting

Conflicting implementation trade-offs

- Low-rate traffic: large *range* of sorting keys to avoid loss
- High-rate traffic fine *grain* of sorting to avoid inaccuracy

→ large number of sorting bins (with head/tail state)

Performance trade-offs

- Effects of ties (multiple cells in same sorting bin)
- Low-rate connections can deny service to high-rate traffic
- Bursty connections can deny service to non-bursty traffic

→ cell delay and traffic distortions

Real-Time Computing Laboratory
The University of Michigan

Slide 17

Hierarchical Sorting Architectures

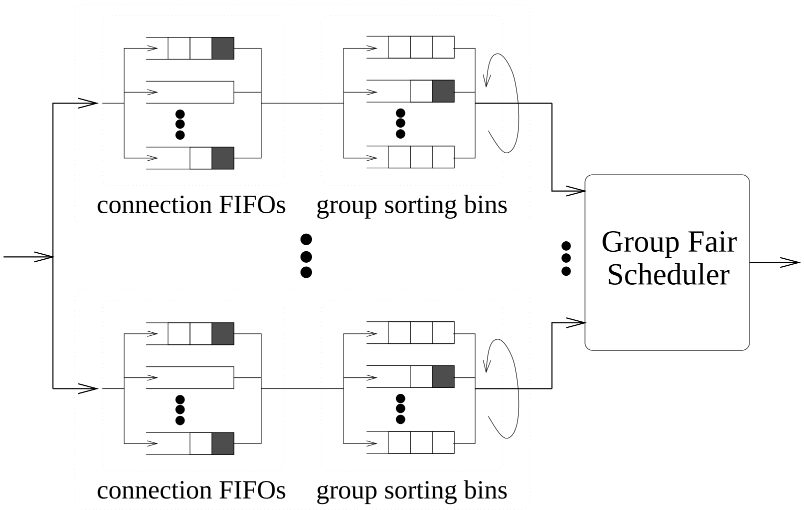
- Reduce range of keys by sorting amongst only *head-of-line* cells
  → per-connection FIFO queues
- Divide connections into groups based on rate parameters
  → e.g., factor of 8 or 16 apart
- Tailor sorting bin range and grain to each group
  - Low rate: coarse grain and large range
  - High rate: fine grain and small range

  → small number of bins in each group
- Perform "fair" arbitration amongst *small* number of groups
  → affordable implementation of fair queueing;
  limit interference from low-rate connections

Real-Time Computing Laboratory
The University of Michigan

Slide 18

Hierarchical Sorting Architecture (continued)



Real-Time Computing Laboratory
The University of Michigan

Slide 19

Conclusions

Quality of service requires efficient support for

- Traffic shaping
- Buffer management
- Link scheduling

Switch implementations are complicated by

- High-speed links and small cells
- Large number of connections
- Diverse rate and burstiness parameters
- Diverse performance requirements

→ scalable (approximate) sorting schemes

Real-Time Computing Laboratory
The University of Michigan