Administrivia

Reader now available at Copy Central
(only 5 papers)

Reading for Monday

- TLL93: ATM remote-memory access (online)
- Portals paper in reader

Outline

Review

Active messages

Software costs

Return to flow control

Flow Control Review

Traditional flow control requires feedback

Network capacity = bandwidth * delay

- ATM $\Rightarrow$ huge capacity

Huge capacity $\Rightarrow$

- End-to-end feedback is too slow
- Use link-to-link (back pressure) or rate-based flow control

Adaptive routing conflicts with back pressure:

- Back pressure along *all* paths before sender notices
- Requires some form of rate-based flow control...

Flow Control Review

Link-based vs. rate-based control:

- LB requires more complicated switches
- LB prevents buffer overflow $\Rightarrow$ *no dropped packets*
- RB artificially limits senders
- LB can easily reach the maximum throughput

LB alone depends on a fixed path:

- No adaptive routing except at setup

Bimodal traffic is the worst case for RB:

- Bursts will overrun buffers $\Rightarrow$ dropped packets
- Bandwidth allocated during idle periods...

Working Alternatives

	Connections	No Connections
Link Based	AN2 Adaptive routing at setup Dropping merely painful	Cray T3D No adaptive routing Dropping prohibited
Rate Based	Standard ATM Adaptive routing at setup Dropping likely & painful	CM-5 (!) Adaptive Routing Dropping prohibited

What does rate-based without connections mean?

Dropping Packets

1. What does it cost?
   - Copy at sender, sequence numbers, acks
   - Requires connections (perhaps implicit)
   - 40-50% of total overhead if done well
2. Causes: buffer overflow (>99%), bit errors
3. What does a dropped packet mean?
   - TCP answer: congestion

An aside:

- one dropped packet $\Rightarrow$ bit error
- consecutive dropped packets $\Rightarrow$ congestion (reduce rate)

Active Messages

Basic idea: three views

- Building block for all protocols
- Remote procedure invocation
- 1/2 of an RPC (other half is another active message)

The remote procedure is the *handler:*

- Handlers typically named by absolute address
- Handlers run atomically (and must be short)
- Handlers may send other active messages (with some restrictions)

Beautifully simple:

- No buffering or ordering required
- Easy dispatch

RPC with Active Messages

Sender:

- Package up handler name + args into message
- Send message
- Block on condition variable (or lock)

Receiver:

- Move arguments into registers
- Execute handler (local procedure call)
- Handler computes result
- Handler sends message with result to sender

Sender:

- Reply handler moves result to prearranged location
- Wakes up blocked thread

Simple Block Transfer

Assumptions:

- ◆ Target buffer preallocated & known to sender
- ◆ Receiver knows the size of the transfer
- ◆ Receiver executes “foo” when transfer completes

Basic handler: put(address, word1, word2, ..., word4)

- ◆ Each message transfers 4 words (CM-5)
- ◆ Basic algorithm:
    1. Copy data to address
    2. Update count of received words
    3. If count matches expected, execute “foo”

Sender just breaks buffer into a sequence of packets

Simple Block Transfer

Observations:

- ◆ Out -of-order delivery is no problem
- ◆ Buffer management is left to application
- ◆ Often need a round trip to setup size and target address

Traditional Send-Receive communication:

- ◆ Receiver specifies target buffer dynamically
- ◆ Receiver blocks for arriving data
- ◆ Sender blocks for receiver’s buffer info (RPC)
- ◆ Block transfer after RPC

Active messages < block transfer < send-receive

Software Costs

Function	Sender	Receiver
One AM	20	27
Data transfer	5w	6.75w
Transfer ordering	0.5w	0.75w
Buffer management	47	101
Final ack	27	20
Real reordering	1.25w	7.25w
Retransmission	7.25w	5w

Potential flaw: messages are too small

Block Transfer Flow Control

Observation: receiver takes longer than sender
  ⇒ path is always full (with back pressure)
  ⇒ with adaptive routing, entire network fills up!

This is where we need rate-based flow control...

Idea: statically limit injection rate to match receiver
  bandwidth matching

If sender takes 20 units and receiver takes 27:
  Make sender wait 7 units between packets

Eliminates congestion, worth 25-200%

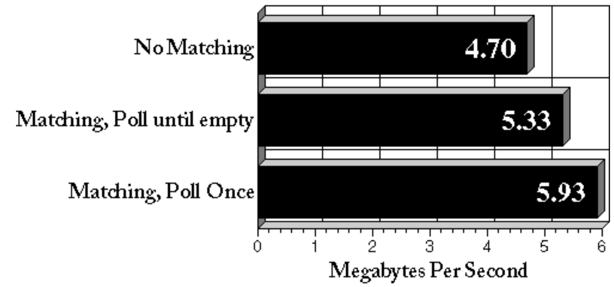
Bandwidth Matching

Bidirectional traffic: (normal case)
    Stable state: alternate send & receive

Repeat until done:
1. Send
2. If there's an arrival, handle all arrivals
3. Else wait R-S+ε

Properties:
- ◆ Correct stable-state behavior
- ◆ Receiver can catch up ⇒ self synchronizing (!)
- ◆ Incoming packets don't have to come from target
- ◆ ε affects send-only bandwidth and catch-up delay

Bandwidth Matching



Matching reduces standard dev. of transfer time by 50x

Bandwidth Matching

Pros:
- ◆ Eliminates congestion
- ◆ No connections (?)
- ◆ Zero overhead in the normal case
- ◆ Works well with adaptive routing

Cons:
- ◆ Requires one-on-one transfers (not really new)
- ◆ Requires polling not interrupts
- ◆ Requires knowledge of the bottleneck rate

Three common rates:
- ◆ Unidirectional
- ◆ Bidirectional (with or without paired transfers)
- ◆ Bidirectional across the bisection

RB Flow Control Revisited

What happened to all of the RB problems?
    Answer: we're also using LB flow control

LB offers:
- ◆ No dropped packets
- ◆ Robust behavior if we get the rate wrong

RB offers:
- ◆ True congestion and flow control
- ◆ Reduces need for virtual channels/virtual circuits

Best performance requires both!
- ◆ Optimal overhead (just block-transfer overhead)
- ◆ Minimal congestion for given end-to-end rate

Connections?

Bandwidth matching requires one-on-one transfers
(undivided attention of the receiver)

One method: set up a connection

- Requires round trip
- Similar to send-receive communication

Alternative: statically manage the “right to send”

- Implicit connections
- Barriers technique from BK94
- More on this in future lectures

Feedback Revisited

Back pressure allows feedback,
but fills network buffers

RB flow control gives up on feedback,
but keeps network clear

Using both is strictly better $\Rightarrow$
Why not augment RB with feedback from LB?

Fact: LB $\Rightarrow$ sender stalls if the network is backed up

Proposal: stalling $\Rightarrow$ congestion $\Rightarrow$ reduce rate

Q: can we build an adaptive RB block transfer?