Flow Control

Digital Communications II

Michaelmas Term 2007
Based on Prof. Jon Crowcroft's notes, and thus transitively on
S. Keshav's "An Engineering Approach to Computer Networking"

DigiComms I

- Explored the fundamentals
  - What to ensure receiver can take in information at the rate the transmitter is sending it
- Circuit switching
  - Easy – agree channel rate
- Packet switching
  - Harder
  - Interactions in intermediate switches
- Sliding windows for flow control
  - Receiver opens and closes the window
  - Still a delay in taking effect

Flow control problem

- Consider file transfer
- Sender sends a stream of packets representing fragments of a file
- Sender should try to match rate at which receiver and network can process data
- Can't send too slow or too fast
- Too slow
  - Wastes time
- Too fast
  - Can lead to buffer overflow
- How to find the correct rate?
  - Particularly given delays on control information

Other considerations

- Simplicity
- Overhead
- Scaling
- Fairness
- Stability

- Many interesting tradeoffs
  - Overhead for stability
  - Simplicity for unfairness

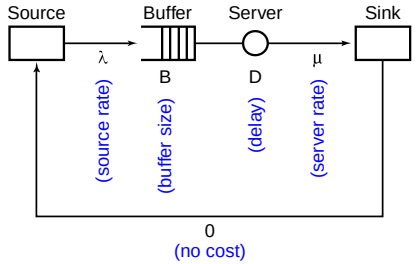
Where?

- Usually at transport layer
- Also, in some cases, in datalink layer

5

Model

- Source, sink, server, service rate, bottleneck, round trip time



6

Classification

- Open loop
  - Source describes its desired flow rate
  - Network *admits* call
  - Source sends at this rate
- Closed loop
  - Source monitors available service rate
    - Explicit or implicit
  - Sends at this rate
  - Due to speed of light delay, errors are bound to occur
- Hybrid
  - Source asks for some minimum rate
  - But can send more, if available

7

Open loop flow control

- Two phases to the flow being controlled
  - Call (i.e. connection) setup
  - Data transmission
- Call setup
  - Network prescribes parameters
  - User chooses parameter values
  - Network admits or denies call
- Data transmission
  - User sends within parameter range
  - Network *polices* users
  - Scheduling policies give user QoS

8

Hard problems

- Choosing a descriptor at a source
- Choosing a scheduling discipline at intermediate network elements
- Admitting calls so that their performance objectives are met (*call admission control*).

9

Traffic descriptors

- Usually an *envelope*
  - Constrains worst case behaviour
- Three uses
  - Basis for traffic contract
  - Input to *regulator*
  - Input to *policer*

10

Descriptor requirements

- Representativity
  - Adequately describes flow, so that network does not reserve too little or too much resource
- Verifiability
  - Verify that descriptor holds
- Preservability
  - Doesn't change inside the network
- Usability
  - Easy to describe and use for admission control

11

Examples

- Representative, verifiable, but not useable
  - Time series of inter-arrival times
- Verifiable, preservable, and useable, but not representative
  - Peak rate

12

Some common descriptors

- Peak rate
- Average rate
- Linear bounded arrival process

13

Peak rate

- Highest ‘rate’ at which a source can send data
- Two ways to compute it
- For networks with fixed-size packets
  - Minimum inter-packet spacing
- For networks with variable-size packets
  - Highest rate over *all* intervals of a particular duration
- Regulator for fixed-size packets
  - Timer set on packet transmission
  - If timer expires, send packet, if any
- Problem
  - Sensitive to extremes

14

Average rate

- Rate over some time period (*window*)
- Less susceptible to outliers
- Parameters: *t* and *a*
- Two types: jumping window and moving window
- Jumping window
  - Over consecutive intervals of length *t*, only *a* bits sent
  - Regulator reinitializes every interval
- Moving window
  - Over all intervals of length *t,* only *a* bits sent
  - Regulator forgets packet sent more than *t* seconds ago

15

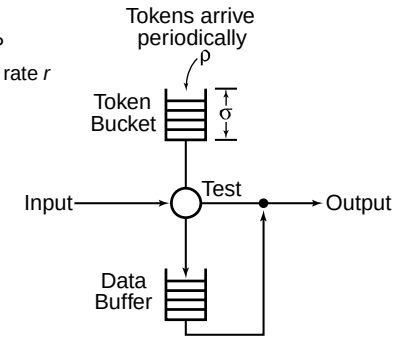
Linear Bounded Arrival Process

- Source bounds # bits sent in any time interval by a linear function of time
- The number of bits transmitted in any active interval of length *t* is less than *rt* + *s*
- *r* is the long term rate
- *s* is the burst limit
- Insensitive to outliers

16

Leaky bucket

- A regulator for an LBAP
- Token bucket fills up at rate *r*
- Largest # tokens < *s*



Variants

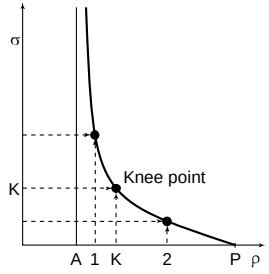
- Token and data buckets
  - Sum is what matters
- Peak rate regulator
  - Previously *s* limits the size of the burst, not its rate

Choosing LBAP parameters

- Trade-off between *r* and *s*
- Minimal descriptor
  - Doesn't simultaneously have smaller *r* and *s*
  - Presumably costs less
- How to choose minimal descriptor?
- Three way trade-off
  - Choice of *s* (data bucket size)
  - Loss rate
  - Choice of *r*

Choosing minimal parameters

- Keeping loss rate the same
  - If *s* is more, *r* is less (smoothing)
  - for each *r* we have least *s*
- Choose knee of curve



Linear Based Arrival Process conclusion

- Popular in practice and in academia
  - Sort of representative
  - Verifiable
  - Sort of preservable
  - Sort of usable
- Problems with multiple time scale traffic
  - Large burst messes up things

21

Open loop vs. closed loop

- Open loop
  - Describe traffic
  - Network admits/reserves resources
  - Regulation/policing
- Closed loop
  - Can't describe traffic or network doesn't support reservation
  - Monitor available bandwidth
    - Perhaps allocated using GPS-emulation
  - Adapt to it
  - If not done properly either
    - Too much loss
    - Unnecessary delay

22

Taxonomy

- First generation
  - Ignores network state
  - Only match receiver
- Second generation
  - Responsive to state
  - Three choices
    - State measurement
      - Explicit or implicit
    - Control
      - Flow control window size or rate
    - Point of control
      - Endpoint or within network

23

Explicit vs. Implicit

- Explicit
  - Network tells source its current rate
  - Better control
  - More overhead
- Implicit
  - Endpoint figures out rate by looking at network
  - Less overhead
- Ideally, want overhead of implicit with effectiveness of explicit

24

Flow control window

- Recall error control window
- Largest number of packet outstanding (sent but not ACKed)
- If endpoint has sent all packets in window, it must wait => slows down its rate
- Thus, window provides *both* error control and flow control
- This is called *transmission* window
- Coupling can be a problem
  - Few buffers are receiver → slow rate!

25

Window vs. rate

- In adaptive rate, we directly control rate
- Needs a timer per connection
- Plusses for window
  - No need for fine-grained timer
  - Self-limiting
- Plusses for rate
  - Better control (finer grain)
  - No coupling of flow control and error control
- Rate control must be careful to avoid overhead and sending too much

26

Hop-by-hop vs. end-to-end

- Hop-by-hop
  - First generation flow control at each link
    - Next server = sink
  - Easy to implement
- End-to-end
  - Sender matches all the servers on its path
- Plusses for hop-by-hop
  - Simpler
  - Distributes overflow
  - Better control
- Plusses for end-to-end
  - Cheaper

27

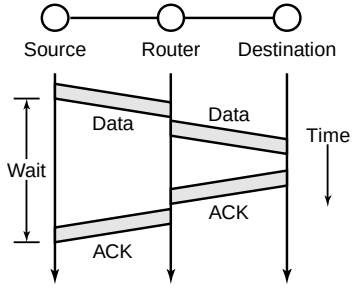
On-off

- Receiver gives ON and OFF signals
- If ON, send at full speed
- If OFF, stop
- OK when RTT is small
- What if OFF is lost?
- Bursty
- Used in serial lines or LANs

28

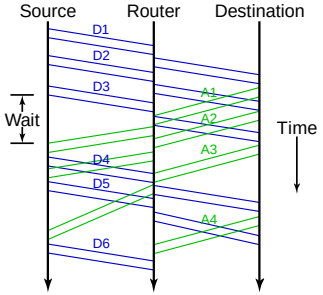
Stop and Wait

- Send a packet
- Wait for ACK before sending next packet



Static window

- Stop and wait can send at most one packet per RTT
- Here, we allow multiple packets per RTT (= transmission window)



What should window size be?

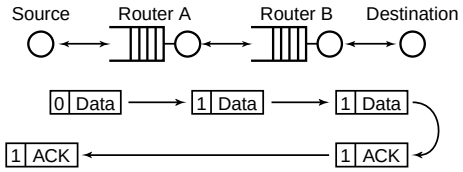
- Let bottleneck service rate along path = b packets/sec
- Let round trip time = R sec
- Let flow control window = w packets
- Sending rate is w packets in R seconds = w/R
- To use bottleneck w/R > b → w > bR
- This is the *bandwidth delay product* or *optimal window size*

Static window

- Works well if b and R are fixed
- But, bottleneck rate changes with time!
- Static choice of w can lead to problems
  - Too small – wasted potential
  - Too large – congestion
- So, need to adapt window
- Always try to get to the *current* optimal value

DECbit flow control

- Intuition
  - Every packet has a bit in header
  - Intermediate routers set bit if queue has built up => source window is too large
  - Sink copies bit to ACK
  - If bits set, source reduces window size
  - In steady state, oscillate around optimal size



33

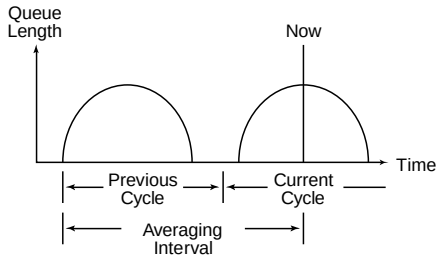
DECbit

- When do bits get set?
- How does a source interpret them?

34

DECbit details: router actions

- Measure *demand* and *mean queue length* of each source
- Computed over queue regeneration cycles
- Balance between sensitivity and stability



35

Router actions

- If mean queue length > 1.0
  - Set bits on sources whose demand exceeds fair share
- If it exceeds 2.0
  - Set bits on everyone
  - Panic!

36

Source actions

- Keep track of bits
- Can't take control actions too fast!
- Wait for past change to take effect
- Measure bits over past + present window size
- If more than 50% set, then decrease window, else increase
- Additive increase, multiplicative decrease

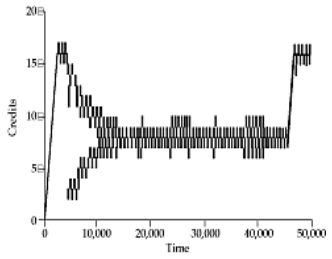
37

Evaluation

- Works with FIFO
  - but requires per-connection state (demand)
- Simple implementation
- But:
  - Assumes cooperation!
  - Has conservative window increase policy

38

Sample trace



39

TCP Flow Control

- Implicit
- Dynamic window
- End-to-end
- Very similar to DECbit, but
  - No support from routers
  - Increase if no loss (usually detected using timeout)
  - Window decrease on a timeout
  - Additive increase, multiplicative decrease

40

TCP details

- Window starts at 1
- Increases exponentially for a while, then linearly
- Exponentially → doubles every RTT
- Linearly → increases by 1 every RTT
- During exponential phase, every ACK results in window increase by 1
- During linear phase, window increases by 1 when # ACKs = window size
- Exponential phase is called *slow start*
- Linear phase is called *congestion avoidance*

41

More TCP details

- On a loss, current window size is stored in a variable called *slow start threshold* or *ssthresh*
- Switch from exponential to linear (slow start to congestion avoidance) when window size reaches threshold
- Loss detected either with timeout or *fast retransmit* (duplicate cumulative ACKs)
- Many versions of TCP
  - Tahoe: in both cases, drop window to 1
  - Reno: on timeout, drop window to 1, and on fast retransmit drop window to half previous size (also, increase window on subsequent ACKs)
  - Vegas, New Reno
  - (CU)BIC: Used in Linux 2.6.8 (2.6.19)

42

TCP vs. DECbit

- Both use dynamic window flow control and additive-increase multiplicative decrease
- TCP uses implicit measurement of congestion
  - Probe a black box
- Operates at the *cliff*
- Source does not filter information

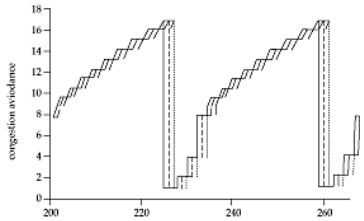
43

Evaluation

- Effective over a wide range of bandwidths
- A lot of operational experience
- Weaknesses
  - Loss → overload? (not always: e.g. wireless)
  - Overload → self-blame, problem with FCFS
  - Overload detected only on a loss
    - In steady state, source *induces* loss
  - Needs at least bR/3 buffers per connection

44

Sample trace



TCP Vegas

- Expected throughput = transmission_window_size/propagation_delay
- Numerator: known
- Denominator: measure *smallest* RTT
- Also know *actual* throughput
- Difference = how much to reduce/increase rate
- Algorithm
  - Send a special packet
  - On ACK, compute expected and actual throughput
  - (expected - actual)* RTT packets in bottleneck buffer
  - Adjust sending rate if this is too large
- Works better than TCP Reno

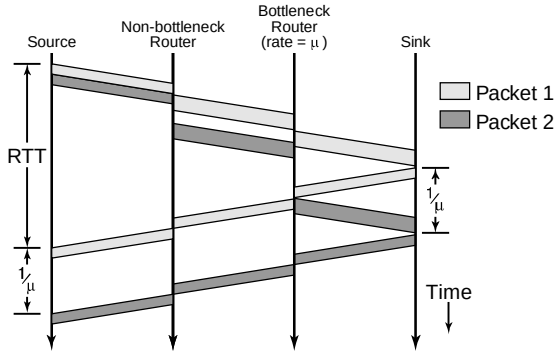
NETwork Block Transfer (NETBLT)

- First rate-based flow control scheme
- Separates error control (window) and flow control (no *coupling*)
- So, losses and retransmissions do not affect the flow rate
- Application data sent as a series of buffers, each at a given rate
- Rate = (burst size at burst rate) so granularity of control = burst
- Initially, no adjustment of rates
- Later, if received rate < sending rate, multiplicatively decrease rate
- Change rate only once per buffer → slow

Packet pair

- Improves basic ideas in NETBLT
  - Better measurement of bottleneck
  - Control based on prediction
  - Finer granularity
- Assume all bottlenecks serve packets in round robin order
- Then, spacing between packets at receiver (= ACK spacing) = 1/(rate of slowest server)
- If *all* data sent as paired packets, no distinction between data and probes
- Implicitly determine service rates if servers are round-robin-like

Packet pair



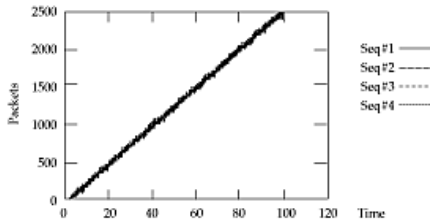
Packet-pair details

- ACKs give time series of service rates in the past
- We can use this to predict the next rate
- Exponential averager, with fuzzy rules to change the averaging factor
- Predicted rate feeds into flow control equation

Packet-pair flow control

- Let X = # packets in bottleneck buffer
- S = # outstanding packets
- R = RTT
- b = bottleneck rate
- Then, X = S - Rb (assuming no losses)
- Let l = source rate
- l(k+1) = b(k+1) + (setpoint -X)/R

Sample trace



ATM Forum End-to-End Rate-based Flow Control (EERC)

- Similar to DECbit, but send a whole cell's worth of info instead of one bit
- Sources periodically send a Resource Management (RM) cell with a *rate request*
  - Typically once every 32 cells
- Each server fills in RM cell with current share, if less
- Source sends at this rate

53

ATM Forum EERC details

- Source sends Explicit Rate (ER) in RM cell
- Switches compute source share in an unspecified manner (allows competition)
- Current rate = Allowed Cell Rate = ACR
- If ER > ACR then ACR = ACR + RIF × PCR else ACR = ER
  - (PCR is Peak Cell Rate, RIF is Rate Increase Factor)
- If switch does not change ER, then use DECbit idea
  - If CI bit set, ACR = ACR (1 - RDF)
- If ER < AR, AR = ER
- Allows interoperability of a sort
- If idle 500 ms, reset rate to Initial cell rate
- If no RM cells return for a while, ACR = ACR × (1-RDF)

54

Comparison with DECbit

- Sources know exact rate
- Non-zero initial cell-rate → conservative increase can be avoided
- Interoperation between ER/CI switches

55

Problems

- RM cells sitting in the data path is a mess
- Updating sending rate based on RM cell can be hard
- Interoperability comes at the cost of reduced efficiency (as bad as DECbit)
- Computing ER is difficult

56

Comparison among closed-loop schemes

- On-off, stop-and-wait, static window, DECbit, TCP, NETBLT, Packet-pair, ATM Forum EERC
- Which is best? No simple answer
- Some rules of thumb
  - Flow control easier with RR scheduling
    - Otherwise, assume cooperation, or police rates
  - Explicit schemes are more robust
  - Hop-by-hop schemes are more responsive, but more complex
  - Try to separate error control and flow control
  - Rate based schemes are inherently unstable unless well-engineered

Hybrid flow control

- Source gets a minimum rate, but can use more
- All problems of both open loop and closed loop flow control
- Resource partitioning problem
  - What fraction can be reserved?
  - How?