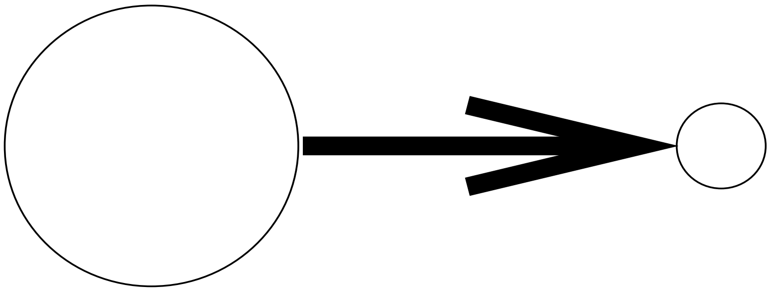
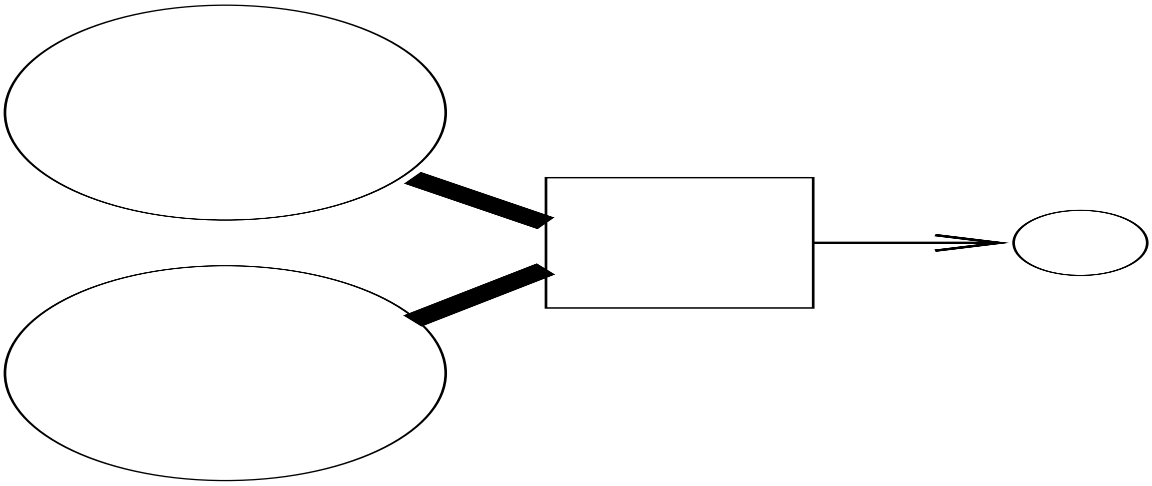
Flow and Congestion Control

- Flow Control
  - Problem: *senders may overwhelm receiver*
  - Solution: limit a sender's rate or burst size



- Congestion Control
  - Problem: *senders may overwhelm intermediate nodes*
  - Solution: limit senders' rates or sizes



- Often, these two goals are referred to simply as "flow control"

Flow Control

- When service rates do not change with time and buffers are sufficiently large, the End-to-End system appears as a single queue/server where
  - delay D is sum of the individual link delays $D_1, \ldots, D_n$
  - buffer B is buffer at the slowest gateway
  - service rate is the minimum $\mu = \min_n(\mu_n)$
- Observations: want to keep bottleneck gateway *just full enough*
  - if λ too small $\Rightarrow$ underutilization
  - if λ too large $\Rightarrow$ excess packet loss
- The Flow Control Problem
  - the bottleneck gateway drains at a variable rate
  - we do not know that rate instantaneously
  - with *feedback*, we can know it after the *Round Trip Time* (RTT)
  - RTT will be of great importance

Classification

- Flow Control Schemes may be: *open-loop*, *closed-loop*, or *hybrid*
- Open Loop
  - source may describe traffic with a *traffic descriptor*
  - source and/or network regulate traffic to the descriptor
  - generally, if resources are reserved and the source is compliant, overload and congestion can be avoided
- Closed Loop
  - source dynamically adapts its flow to match available resources
  - feedback is required to trigger adjustments
  - two types of feedback: *implicit* and *explicit*
- Hybrid
  - some combination of closed and open loop
  - may require some specification at call setup (if setup is required)
  - subsequent dynamic adaptation

Open Loop Flow Control

- source must somehow regulate its own flow
- may involve a *traffic descriptor*
- Traffic Descriptors:
  - Representativity – should represent long-term traffic behavior
  - Verifiability – can be used to verify a source's compliance
  - Preservability – can be preserved across switches with different service capabilities
  - Usability – easy for source to describe its traffic and for switches to perform admission control
- Open Loop Control will usually involve regulating a source's average data rate and its maximum single burst size

Peak Rate Traffic Descriptor

- refers to the highest data rate a source can generate
- Two possible definitions:
  - Constant Sized Packets: inverse of closest inter-packet time
  - Variable Sized Packets: measure peak rate over some time window and take the greatest value over all such time windows
- Example:
  - 50 byte packets, 10ms min spacing
  - Peak Rate $\Rightarrow \frac{50bytes}{.01secs} = 5000bytes/s$
  - Assuming a peak rate of 8Mbps over 15ms intervals, in 75ms we could generate $8Mbps \cdot 75ms = 0.6Mbits$
- Implementation:
  - buffer and timer
  - timer is used to "meter" packets at the smalles inter-departure time
- this bound is very loose, and is vulnerable to "outliers"

Average Rate Traffic Descriptor

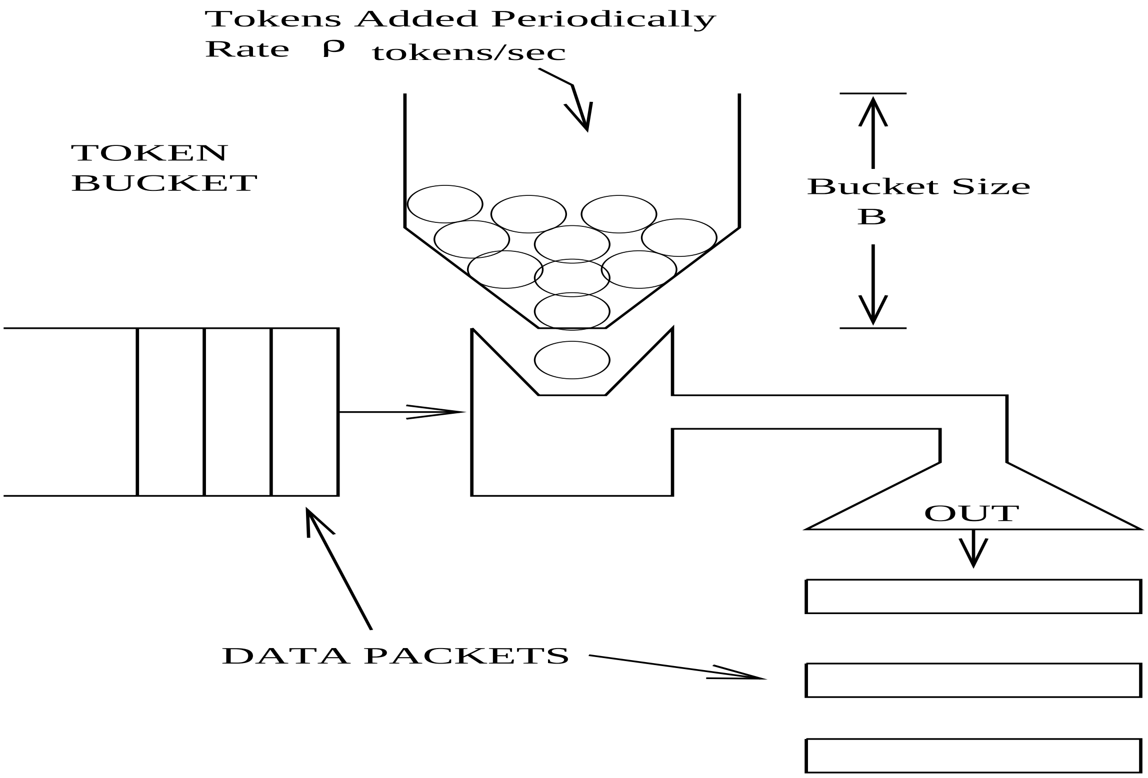
- the peak rate descriptor was subject to outliers
- Two types of averages: *jumping window* and *moving window*
- Jumping Window
  - measured over consecutive windows of length w
  - at most a bits can be sent in w seconds
  - sensitive to position of initial window
- Moving Window
  - window moves continuously
  - at most a bits can be sent in *all* windows of length w
  - insensitive to position of initial window
- Implementation:
  - Jumping: counter is incremented by size of each departing packet. If this size exceeds a in a window, delay until next window.
  - Moving: use above scheme, plus store time and sizes of departing packets. A counter keeps track of the number of bits sent in the last w seconds. When delaying, can determine when to send by inspecting the list.

Linear Bounded Arrival Process

- Abbreviation: LBAP
- a generalization of the average-rate descriptor
- Characteristics
  - source rate is bounded by $f(t)$ over time t, f is linear
  - let $f(t) = \rho t + \sigma$
  - ρ is long-term average rate
  - σ is maximum burst size
- Can be regulated by a *Leaky Bucket*

Leaky Bucket

- Leaky-Bucket Regulator



- limits maximum burst size to bucket size B
- limits average packet rate to token fill rate ρ
- limits average bit rate to ρT, where T is token size

Leaky Bucket

- Observation:
  - assume we wish to send a packet of P bits
  - assume there are N tokens in the bucket
  - $(P - NT)$ is the number of "left over bits"
  - we can send them in time ρT
  - so, $\lceil (P - NT)/(\rho T) \rceil$ is the minimum delay introduced by the regulator
- Example:
  - 100-byte tokens are added to a token of size 500 bytes periodically at 2 per second
  - Average Rate $\Rightarrow 2x100 = 200 bytes/s = 1600 b/s$
  - Peak Rate $\Rightarrow$ *unbounded* (500 bytes in arbitrarily small time)
  - Maximum Burst $\Rightarrow$ 500 bytes (5 tokens worth)
- Variants:
  - peak-rate regulator
  - inclusion of finite-length buffer for delayed packets (packets dropped if delay buffer full)

Closed Loop Flow Control

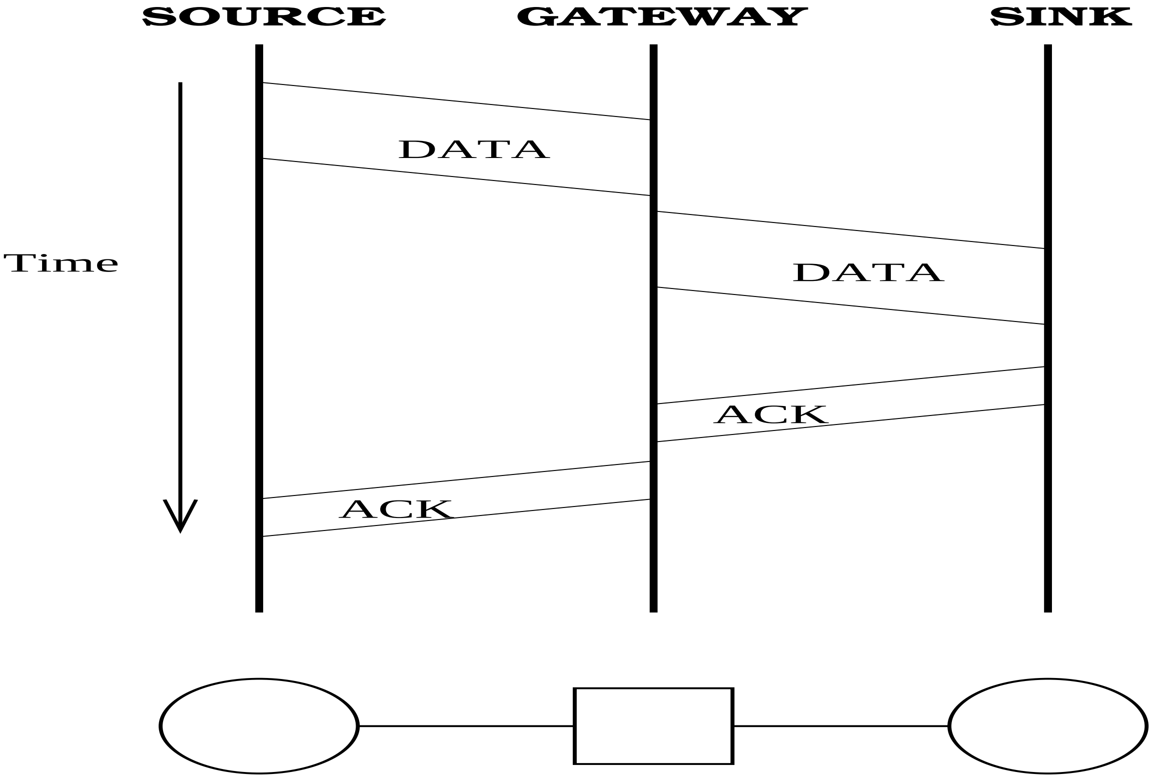
- Assumption: sources do not make *reservations*
  - reservation may not be supported
  - network may overbook
- Operation
  - use *feedback* to adapt to available resources
  - ineffective adaptation leads to underutilization or excess loss
- Generations of Protocols
  - first generation handled only "flow control"
  - second generation also handles "congestion control"

Closed Loop Design Space

- Design Space for Closed-Loop Schemes
  1. Implicit or Explicit state measurement
  2. Dynamic Window or Dynamic Rate
  3. Hop-by-Hop or End-to-End
- Implicit or explicit
  - implicit infers congestion state by some network measurement
  - explicit requires special messages or indicators
- Dynamic Window or Dynamic Rate
  - dynamic window regularly adjusts *volume* of data
  - dynamic rate regularly adjusts *rate* of data
- Hop-by-Hop or End-to-End
  - hop-by-hop involves flow control at each intermediate
  - end-to-end only involves flow control at perimeters

Stop and Wait

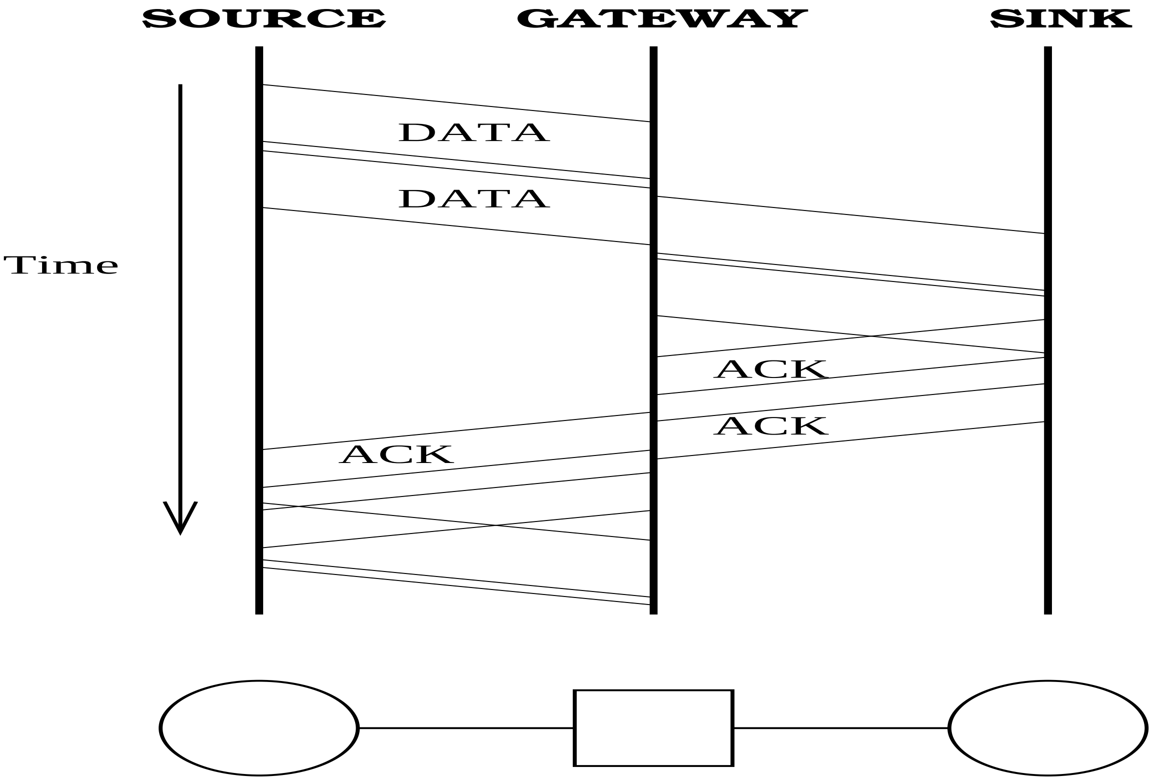
- The earliest/simplest of window-based flow control
- Also provides error control



- in SW, the window size $w = 1$
- throughput is limited to $\frac{w}{RTT} = \frac{1}{RTT}$
- degrades seriously with long-RTT networks

Fixed Window

- SW fails to "keep the pipe full"
- so, allow more data outstanding in the pipe



- window size $w = k$ (k unacknowledged packets allowed)
- throughput is limited to $\frac{w}{RTT} = \frac{k}{RTT}$
- *issue*: what if network is non-static

Window Sizes

- so, how big should the window be
- The Optimal Window Size
  - assume the departure rate at bottleneck is μ, and RTT R
  - in one RTT, the bottleneck services $R\mu$ packets
  - assume the sources sends packets $1, 2, \ldots, w$ at time 0
  - we want $w = R\mu$ to keep the bottleneck full but not overfull
  - if $w > R\mu$, then $w - R\mu$ packets will be buffered
  - if $w < R\mu$, then the bottleneck will go idle
  - $R\mu$ is a critical value called the *bandwidth-delay product*
- we can compute the optimal window size as given above
- in practice, μ and R may vary, sometimes within the same connection context

The DECbit Scheme

- paper: *A Binary Feedback Scheme for Congestion Avoidance in Computer Networks*
- *explicit* congestion notification using a header bit
- a receiver receiving a set "DECbit" copies it to ACKs
- sources adjusts w based on the series of bits it sees
- w will oscillate around the optimal value until a change
- Major Issues:
  - when to set bits, and on what connections
  - how the source responds to bits

DECbit

- When to set bits, and on which connections
  - compute *mean aggregate queue length* of output queue
  - average over *queue regeneration cycles*
- How to Take the Average
  - instantaneous queue length may not represent long-term situation
  - instantaneous could result in unfairness
  - need a *low-pass filter*
  - tried averaging over fixed time period; works ok when interval is close to the RTT
  - also tried running weighted exponential average
  - both show inconsistencies in feedback data
- Use and Adaptive Averaging Algorithm
  - uses *regeneration cycles*
  - *busy* (0-1) and *idle* (1-0) periods
  - bit is set as soon as average queue length exceeds 0
  - measurements taken over current (partial) and last cycle

DECbit

- End-System Policy:
  - transport updates w after receiving ACKS for W_p+W_c packets
  - only bits for W_c packets are examined
  - if more than 1/2 of the bits are "on", w is reduced, otherwise is increased
  - w is increased additively by α
  - w is decreased multiplicatively by β
  - (suggest $\alpha = 1, \beta = \frac{7}{8}$)
- Additive Increase/Additive Decrease

$$Increase : W_c^i = W_p^i + b, b \geq 0;$$

$$Decrease : W_c^i = W_p^i - d, d \geq 0.$$

  - leads to unfair allocation
- Additive Increase/Multiplicative Decrease

$$Increase : W_c^i = W_p^i + b, b \geq 0;$$

$$Decrease : W_c^i = dW_p^i, 0 < d \leq 1.$$

  - fair allocation, shown as only stable source behavior
- DECbit is unfair for non-cooperating source

TCP (Jacobson/Karels)

- achieve network stability by obeying *packet conservation* principal
- uses *implicit* signals to indicate network state
- also: fix a number of other performance problems (timer management, error recovery)
- New Algorithms:
  - slow-start
  - round-trip-time *variance* estimation
  - exponential retransmit timer backoff
  - more aggressive receiver ACK policy
  - dynamic window sizing on congestion
  - “fast retransmit”
- Slow-Start
  - technique to get to equilibrium in packet conservation
  - Details:
    * add a *congestion window (cwnd)* per connection
    * initialize $cwnd = 1$ on loss
    * increment *cwnd* by 1 for each ACK received
    * growth: 1 seg $\Rightarrow$ 2 segs $\Rightarrow$ 4 segs
    * time $R \log_2 W$ to reach the operating window

TCP

- Original Round-Trip Timing
  - a good RTT estimate lets us assign a good RTO
  - original TCP specification suggested estimating RTT as follows:
    $$R \leftarrow \alpha R + (1 - \alpha) M$$
  - R is RTT estimate, M is a new RTT measurement, α is a gain constant (0.90 recommended)
  - RTO set to βR
  - β accounts for variation, with recommended value 2
- Using a constant for β when estimating RTT doesn't adapt to wide var iance
- Need to replace β with a variance estimator of M (new samples)

TCP RTT Estimation

- Popular estimator is the sample *standard deviation* S
- Assuming a is the estimated (time-averaged) RTT:

$$S^2 = \frac{\Sigma(m-a)^2}{n-1}$$

- This requires squaring and a square root operation to get S
- Try *mean prediction error* or *mean deviation* $mdev = |m - a|$

$$mdev^2 = \left(\sum |m-a|\right)^2 \geq \sum(m-a)^2 = S^2$$

- So, $mdev$ is more conservative than S and cheaper to compute
- Set the retransmit timer timeout rto to $a + 4 * mdev$.
- Adapts much better to RTT variance

Slow-Start and Congestion Avoidance

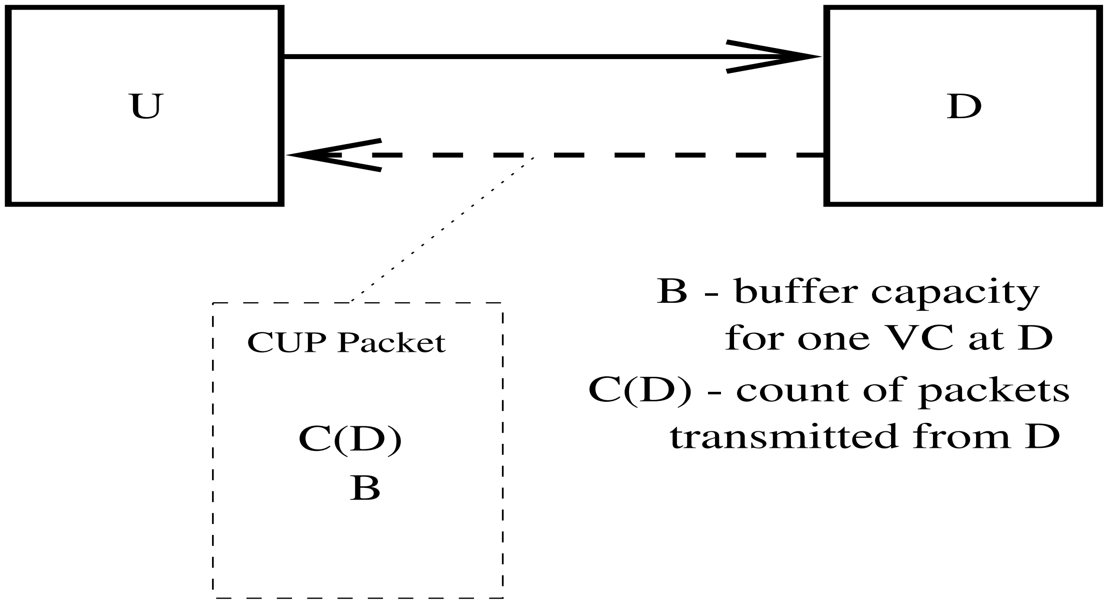
- Congestion Avoidance is usually combined with Slow-Start
  - slow-start grows window exponentially
  - congestion avoidance grows window linearly
  - variable *ssthresh* indicates level at which congestion avoidance takes over
- Action on timer expiration
  - $ssthresh = \frac{cwnd}{2}$
  - $cwnd = 1$
  - double retransmission timer
- Action on dup ACKS (*tcprexmtthresh*)
  - Tahoe: same as on timer expiration
  - Reno: $cwnd = \frac{cwnd}{2}, ssthresh = cwnd$
- General Observations:
  - TCPs cooperate to shed load during congestion periods
  - any loss is assumed to indicate congestion
  - steady-state operation keeps intermediate buffers near overrun

Credit-Based ATM Flow Control

- an explicit, dynamic window, hop-by-hop, credit-based scheme for ATM
- was eventually *not* adopted by the ATM Forum
- per-connection, per-link FC window
- Major Components:
  - *credit update protocol* (CUP)
  - adaptation of buffer allocation based on source rate
- CUP is used to convey buffer occupancy information to upstream nodes
- buffer adaptation moves buffering resources to flows that are using them
- separates flow control from error control
- guarantees zero loss, but complicates switch implementation
- assumes a fair allocation at switches

CUP

CB(U) - credit balance at U
C(U) - count of packets transmitted from U



- source transmits only if credit balance $CB(U) > 0$
- $C(U) - C(D)$ measures the number of packets in transit or cached at D
- thus, overflow can be prevented by sending at most $B-(C(U)-C(D))$ packets

A Word on Fairness

- most efforts use the *max-min fair share* criteria
- Max-Min Fairness:
  - to divide a local (single) switch's resources evenly among flows
  - label sources $1, \ldots, n$
  - with resource demands $x(1), \ldots, x(n)$
  - WOLOG, order them such that $x(1) \leq x(2) \ldots x(n)$
  - let a resource have some capacity C
  - initially give C/n to $x(1)$ (might be in excess $C - x(1)$)
  - distribute the rest evenly as $C/n + (C - x(1))/(n-1)$
  - this might be more than $x(2)$ wants, so continue iterating
- iteration completes when no source gets more than it asked for and no source gets less than what a source of higher index got
- used in DECbit, for example

Dropping Packets

- when a switch's buffers overflow, packets are dropped
- (note: congestion may occur before packets are dropped)
- In a Switch:
  - *scheduling* indicates which packet to serve next
  - *packet drop policy* dictates which packets to drop next
- Drop Policy Parameters:
  - Degree of aggregation (isolation between queues)
  - Priority
  - Early or overloaded drop
  - Drop position
- Simple Example: FIFO
  - typically no isolation (single shared buffer pool)
  - may support priority
  - typically drops on overload
  - drops most recent packets arriving to full queue (also called *drop-tail*)