Traffic Management

Delivery of Quality of Service to specific packet flows with efficient use of network resources

• Packet Level
  – Queueing & scheduling at switching and multiplexing points
  – Determines relative performance offered to packets over a short time scale (microseconds)

• Flow Level
  – Management of traffic flows & resource allocation to ensure delivery of QoS (milliseconds to seconds)
  – Matching traffic flows to resources available; congestion control

• Flow-Aggregate Level
  – Routing of aggregate traffic flows across the network for efficient utilization of resources and meeting of service levels
  – “Traffic Engineering”, at scale of minutes to days
Need for packet-level control for end-to-end QoS

- A packet traversing network encounters delay and possible loss at various multiplexing points
- End-to-end performance is accumulation of per-hop performances

FIFO and Priority Queues

- *First-In-First-Out*: all arriving packets placed in a queue
  - Transmitted in order of arrival
  - Packets discarded when buffer full
  - Delay and loss of packets depends on interarrival times and packet lengths
  - Subject to *hogging*
    - One flow sends packets at a high rate and fills all the buffers
  - Not possible to provide differentiated services in basic FIFO scheme
    - since all packets treated in the same fashion
- Different services can be provided with a FIFO variation with *priority classes*:
  - Packets from the higher priority class are buffered as long as there is space
  - Packets from the lower priority class are discarded when number of packets buffered reaches a threshold
Head-of-Line priority queuing

- Separate queues for each of a number of priority classes:
  - High priority queue for packets with low delay requirements and vice versa
  - Buffer sizes for different priority classes according to loss probabilities

![Diagram of Head-of-Line priority queuing]

- Pre-emptively always select next packet from highest priority queue
  - can lead to starvation of lower priority queues
- or allow a flow of packets from lower priority queues at less frequent rates
- Still no discrimination between flows at same priority level
  - hogging can still occur

Earliest Due Date Scheduling

- Queue sorted in order of some priority tag
  - but still discard when buffers full

![Diagram of Earliest Due Date Scheduling]

- Defining priority can be very flexible
  - Could be dynamic as conditions and flows change
- Priority could reflect a time due
  - The time by which the packet should be transmitted
  - Packets with no delay requirements get indefinite or long times-due
  - Time-critical packets get a short time due
Fair Queueing

- Attempts to provide equitable access to transmission bandwidth
- Each packet flow has its own queue
- Bandwidth is divided equally between flows
  - Each flow transmitted in turn in round robin fashion

Fairness and Interleaving Granularity

- With packet interleaving, bandwidth not necessarily equally shared
  - Packet lengths of different flows may vary
  - Flows with larger packet sizes will get more of the bandwidth
- Byte-by-byte, or any other fixed size, interleaving is fair but sorting out which byte belongs to which packet at the other end is a problem
- A fairer system takes packet length into account
- Compute packet completion time in ideal system: packet length / fair share of bandwidth
  - Add tag to packet
  - Sort packet in queue according to tag
  - Serve according to HOL
- Weighted Fair Queueing
  - Used for flows with different delay requirements
  - Weight applied to each flow to calculate a fair share of bandwidth
  - If flow $i$ has a weight $w_i$, its proportion of the bandwidth $= \frac{w_i}{\sum w}$
Congestion Control

Congestion occurs when too many packets arrive at the same switch
- If the arrival rate can be greater than the onward transmission rate
  » the pool of buffers in the switch can be used up
  » can happen irrespective of how big the pool is
    - a larger pool just postpones the congestion
- When buffers full, packets will be discarded
  » the destination will detect missing packets and request retransmission
    - probably along the same path as before
  » this generates even more traffic at the congested node
- If uncontrolled, the effect is a rapid drop in throughput:

Approaches to Congestion Control

• Open Loop - prevent congestion by controlling traffic flow at source
  - If the QoS cannot be guaranteed, the flow must be rejected
  - This is admission control and involves some type of resource reservation

• Closed Loop - react to congestion when it happens
  - Typically by regulating the traffic flow according to the state of the network
  - Closed loop because the network state has to be fed back to the point where traffic is regulated, usually the source
  - Typically do not use resource reservation

Congestion control is effective for temporary network overloads
- If the congestion persists (~secs), adaptive routing is probably needed to reroute around the congested part of the network
- For longer periods of congestion, the network needs to be upgraded
  » higher capacity links, switches etc.
Open Loop control

• Network performance is guaranteed to all traffic flows that have been admitted into the network
• Assumes that once a source is accepted it will not overload the network
• Initially for connection-oriented networks

Key mechanisms:

• Admission control
  – The source declares its requirements at connection set-up
  – The admission control processes computes the required resources, checks their availability and accepts or rejects the connection

• Policing
  – Monitoring of the flow to ensure it complies with its promises

• Traffic shaping
  – Altering the flow to ensure conformance

Connection Admission Control (CAC)

• Initially proposed for connection-oriented networks such as ATM
  – but also for datagram networks in terms of flow rather than a connection

• Each source requesting a connection must supply a traffic descriptor:
  – Peak, avg. bit rate, delay, loss requirements, maximum burst size
  – Maximum burst size relates to the maximum time traffic is generated at the peak rate

• The amount of bandwidth the CAC will allocate will normally lie somewhere between the average and the peak rates
  – Called the effective bandwidth

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Bits per second</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
</tr>
<tr>
<td>20</td>
<td>300</td>
</tr>
<tr>
<td>30</td>
<td>400</td>
</tr>
<tr>
<td>40</td>
<td>500</td>
</tr>
<tr>
<td>50</td>
<td>600</td>
</tr>
</tbody>
</table>

Peak rate
Average rate
Policing

- Network monitors traffic flows continuously to ensure they meet their traffic contract
- When a packet violates the contract, network can discard or tag the packet giving it lower priority
- If congestion occurs, tagged packets are discarded first
- *Leaky Bucket Algorithm* is the most commonly used policing mechanism
  - Bucket has specified leak rate for average contracted rate
  - Bucket has specified depth to accommodate variations in arrival rate
  - Arriving packet is *conforming* if it does not result in overflow

Leaky Bucket Overview

- Water flows into the bucket and leaks out at a constant rate
- The bucket overflows if too much water is poured into it
- The bucket absorbs irregularities in water supply
  - Bucket can be made shallow if the flow is expected to be smooth
  - deep if the flow may be bursty
- Packets are equivalent to dollops of water
  - a packet is *conforming* if the bucket does not overflow when it is added
  - *non-conforming* if it does overflow
- The drain-rate corresponds to the traffic rate that is to be policed
Leaky Bucket algorithm

Arrival of a packet at time \( t_a \)

Depletion rate: 1 packet per unit time

\( \text{Netflow rate: } L + I = \text{Bucket Depth} \)

\( I = \text{increment per arrival, nominal interarrival time} \)

\( X' = X - (t_a - \text{LCT}) \)

Interarrival time

Yes

\( X' < 0 ? \)

Non-empty

Yes

\( X = X' + I \)

Nonconforming packet

\( L + I \)

Conforming packet

No

\( X' = 0 \)

Empty

\( X > L ? \)

Arriving packet would cause overflow

\( X' = 0 \)

Conforming packet

\( X = X' = 0 \)

LCT = last conformance time

\( X = \text{value of the leaky bucket counter} \)

\( X' = \text{auxiliary variable} \)

Leaky Bucket Example

\( I = 4 \quad L = 6 \)

Packet arrival

Nonconforming

Time

\( L + I \)

Bucket content

Non-conforming packets not allowed into bucket & hence not included in calculations
Policing Parameters

- The inverse of $I$ is the sustainable rate
  - i.e. the long term average rate allowed for conforming traffic
- if the peak arrival rate is $R$ and $T = 1/R$, then maximum burst size is found to be:
  $$1 + \left\lfloor \frac{L}{I - T} \right\rfloor$$

- The leaky bucket algorithm can be used to police other traffic parameters
- It is common to have a number of leaky buckets in series to police all the required parameters

Traffic Shaping

- The source need to shape its traffic flow to what is specified for the line
- Shaping can be used to make the traffic smoother
  - e.g. an application that periodically generates 10kbits of data per second
- The destination may wish the traffic in fast short bursts or continuously:

  ![Graphs](attachment:graphs.png)

  - (a) least likely to stress the network, but destination may prefer (c)
**Traffic Shaping**

- Uses a buffer whose content is read out periodically at a constant rate
  - The bucket in this case is a buffer that stores packets
- The buffer smooths out the stream of packets
- Buffer size defines the maximum burst that can be accommodated
  - if the buffer is full, incoming packets are discarded

- Rather restrictive since the output rate is constant when the buffer is not empty
  - Delays may be unnecessarily long due to buffering
- For variable rate traffic, it might be better to allow bursts of traffic through when it is still conforming
  - Traffic does not need to be smooth to be conforming
  - The policing allows burstiness as long as it is under a certain limit

---

**Token-Bucket Traffic Shaper**

- The bucket now holds *tokens* which are generated at a constant rate
- A packet can only be taken from the buffer and transmitted if there is a token in the token bucket to be removed
- When the buffer is empty, the token bucket can accumulate tokens

- As long as there are tokens available, packets are transmitted as soon as they arrive (in bursts)
  - The size of the token bucket determines the amount of burstiness
- If the token bucket is empty, the packets have to wait in the buffer
  - Packets will be transmitted at the rate of generating tokens
- When the size of the token bucket is zero, the token bucket shaper reduces to just a leaky bucket shaper
Closed Loop Control

- Relies on feedback information to regulate the source flow

Three tasks:
- Detect congestion
  - Monitor queue sizes, drop rates, increases in delay, etc.
- Alert entities that can take action
  - Implicit, e.g. time-out
  - Explicit, e.g. control message
- Adjust system operation to correct the problem

- End-to-end vs. Hop-by-hop
  - Notifying the source (end-to-end) means that corrective action is delayed
  - If the intermediate switches are notified they can relieve the congestion spot faster (e.g. drop packets that would pass through the trouble spot)

TCP congestion control outline

- TCP uses a closed-loop implicit congestion control mechanism, the congestion window
  - Similar to *sliding window ARQ protocol* for end-to-end flow control
  - At any time the *maximum* amount of data the source can transmit is the *minimum of the advertised window and the congestion window*

- TCP congestion control algorithm *dynamically* controls the size of the congestion window according to the network state

- phase 1: *slow start*
  - Start with window size 1
  - Double size for each ACK on time until you reach the threshold

- phase 2: *congestion avoidance*
  - Increase window linearly until a time-out

- phase 3 : adaptation
  - Calculate new threshold (half of current window size) and go to phase 1